Concurrency Control Mechanism In Distributed Database System

Prof. Dimple Kanani[1] Prof. Brijal Patel[2]
[1, 2] Assistant Professor, Dept of Information Technology
[1, 2] Vadodara Institute of Engineering, Vadodara, India

Abstract- *the need and improvement in distributed database system is of paramount importance in today's world. Today's high-volume data storage is increasing online transaction processing that will lead to concurrency control; each transaction should follow ACID Property. The difficulties mostly faced in distributed database system is Protecting the ACID property i.e. when concurrent transactions perform read and write atomicity, consistency, integrity and durability of the database should be preserved and Recovery method to be used when distributed database crashes. Ideas that are used in the design, development, and performance of concurrency control mechanisms have been summarized. The ACID Property, issues in concurrency control and locking mechanisms are included.*

Keywords- concurrency control, Distributed database system, ACID Property, Lock

I. INTRODUCTION

A database transaction is a unit of work performed against a database management system or similar system that is treated in a coherent and reliable way independent of other transactions. A transaction is Logical Units of Work that made of read (read(x)) and write (write(x)) operation.

A database transaction, by definition, must be atomic, consistent, isolated and durable. These properties of database transactions are often referred to by the ACID.

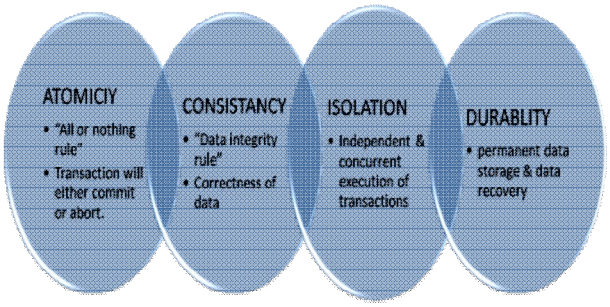
The *ACID* model is one of the oldest and most important concepts of database theory. It sets forward four goals that every database management system must strive to achieve: atomicity, consistency, isolation and durability. No database that fails to meet any of these four goals can be considered reliable.

Atomicity states that database modifications must follow an "all or nothing" rule. Each transaction is said to be "atomic." If one part of the transaction fails, the entire transaction fails. It is critical that the database management system maintain the atomic nature of transactions in spite of any DBMS, operating system or hardware failure.

Consistency checks correctness of data. It states that only valid data will be written to the database. If, for some reason, a transaction is executed that violates the database's consistency rules, the entire transaction will be rolled back and the database will be restored to a state consistent with those rules. On the other hand, if a transaction successfully executes, it will take the database from one state that is consistent with the rules to another state that is also consistent with the rules.

Isolation requires that multiple transactions occurring at the same time not impact each other's execution. For example, if Joe issues a transaction against a database at the same time that Mary issues a different transaction, both transactions should operate on the database in an isolated manner. The database should either perform Joe's entire transaction before executing Mary's or vice-versa. This prevents Joe's transaction from reading intermediate data produced as a side effect of part of Mary's transaction that will not eventually be committed to the database. But the isolation property does not ensure which transaction will execute first, merely that they will not interfere with each other.

Durability works for permanent changes. It also ensures that any transaction committed to the database will not be lost. Durability is ensured through the use of database backups and transaction logs that facilitate the restoration of committed transactions in spite of any subsequent software or hardware failures.



BeginTrans - Commit Tran - Rollback Tran: Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the

tasks fails, the transaction fails. Therefore, a transaction has only two results: *success or failure.* Incomplete steps result in the failure of the transaction.

II. TRANSACTION STATES

- *Begin Transaction*
- *Rollback Transaction*
- *Commit Transaction*

Begin Transaction Marks the starting point of an explicit, local transaction. BEGIN TRANSACTION

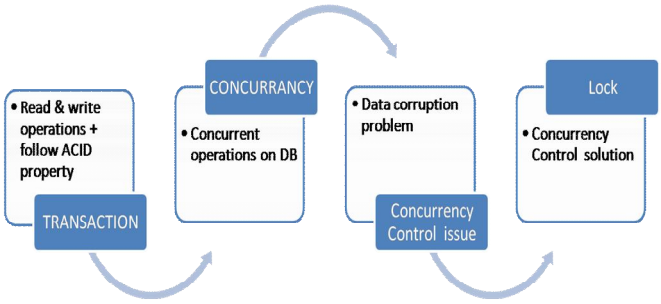
Rollback Transaction if anything goes wrong with any of the grouped statements, all changes need to be aborted. The process of reversing changes is called rollback in SQL Server terminology.

Commit Transaction If everything is in order with all statements within a single transaction, all changes are recorded together in the database. In SQL Server terminology, we say that these changes are committed to the database.

III. TRANSACTION MODES

1. *Autocommit transactions*: Each individual statement is a transaction.
2. *Explicit transactions*: Each transaction is explicitly started with the BEGIN TRANSACTION statement and explicitly ended with a COMMIT or ROLLBACK statement.
3. *Implicit transactions*: A new transaction is implicitly started when the prior transaction completes, but each transaction is explicitly completed with a COMMIT or ROLLBACK statement.

If you want all your commands to require an explicit COMMIT or ROLLBACK in order to finish, you can issue the command SET IMPLICIT_TRANSACTIONS ON. By default, SQL Server operates in the autocommit mode; it does not operate with implicit transactions. Any time you issue a data modification command such as INSERT, UPDATE, or DELETE, SQL Server automatically commits the transaction. However, if you use the SET IMPLICIT_TRANSACTIONS ON command, you can override the automatic commitment so that SQL Server will wait for you to issue an explicit COMMIT or ROLLBACK statement to do anything with the transaction. This can be handy when you issue commands interactively, mimicking the behavior of other databases such as Oracle.



IV. CONCURRENCY

When many people attempt to modify data in a database at the same time, a system of controls must be implemented so that modifications made by one person do not adversely affect those of another person. This is called *concurrency control.*

4.1 Concurrency control theory has *two classifications* for the methods of instituting concurrency control:

4.1.1 Pessimistic concurrency control

A system of locks prevents users from modifying data in a way that affects other users. After a user performs an action that causes a lock to be applied, other users cannot perform actions that would conflict with the lock until the owner releases it. This is called pessimistic control because it is mainly used in environments where there is high contention for data, where the cost of protecting data with locks is less than the cost of rolling back transactions if concurrency conflicts occur.

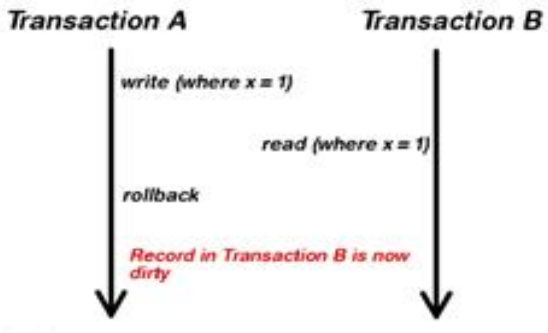
4.1.2 Optimistic concurrency control

in optimistic concurrency control, users do not lock data when they read it. When an update is performed, the system checks to see if another user changed the data after it was read. If another user updated the data, an error is raised. Typically, the user receiving the error rolls back the transaction and starts over. This is called optimistic because it is mainly used in environments where there is low contention for data, and where the cost of occasionally rolling back a transaction outweighs the costs of locking data when read.

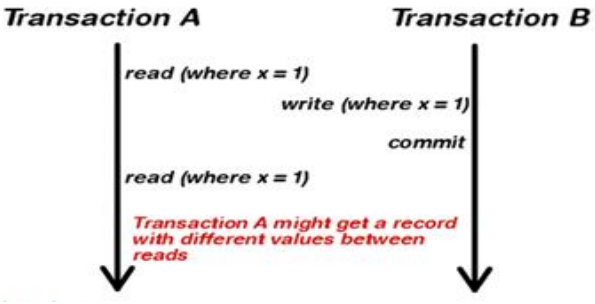
4.2 Issues:

4.2.1 Dirty reads Uncommitted dependency occurs when a second transaction selects a row that is being updated by another transaction. The second transaction is reading data that has not been committed yet and may be changed by the transaction updating the row.

In this example Transaction A writes a record. Meanwhile Transaction B reads that same record before Transaction A commits. Later Transaction A decides to rollback and now we have changes in Transaction B that are inconsistent. This is a dirty read. TransactionB was running in READ_UNCOMMITTED isolation level so it was able to read Transaction A changes before a commit occurred.



4.2.2 Unrepeatable reads: Inconsistent Analysis (Nonrepeatable Read) Inconsistent analysis occurs when a second transaction accesses the same row several times and reads different data each time. Inconsistent analysis is similar to uncommitted dependency in that another transaction is changing the data that a second transaction is reading. However, in inconsistent analysis, the data read by the second transaction was committed by the transaction that made the change. Also, inconsistent analysis involves multiple reads (two or more) of the same row and each time the information is changed by another transaction; thus, the term nonrepeatable read.

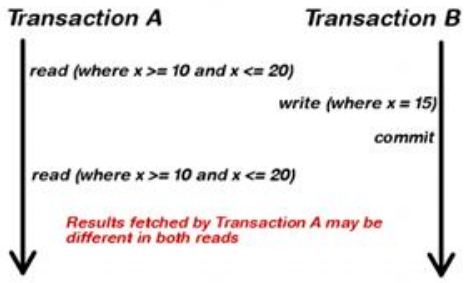


In this example Transaction A reads some record. Then Transaction B writes that same record and commits. Later Transaction A reads that same record again and may get different values because Transaction B made changes to that record and committed. This is a non-repeatable read.

4.2.3 Phantom problem**:**

Phantom reads occur when an insert or delete action is performed against a row that belongs to a range of rows being read by a transaction. The transaction's first read of the range of rows shows a row that no longer exists in the second or succeeding read, as a result of a deletion by a different transaction. Similarly, as the result of an insert by a different transaction, the transaction's second or succeeding read shows a row that did not exist in the original read.

For example, an editor makes changes to a document submitted by a writer, but when the changes are incorporated into the master copy of the document by the production department, they find that new unedited material has been added to the document by the author. This problem could be avoided if no one could add new material to the document until the editor and production department finish working with the original document.



Lost Updates:

Lost updates occur when two or more transactions select the same row and then update the row based on the value originally selected. Each transaction is unaware of other transactions. The last update overwrites updates made by the other transactions, which results in lost data.

V. TYPES OF ISOLATION LEVELS

The transaction isolation levels help to determine whether the concurrently running transactions in a DB can affect each other or not. If there are 2 or more transactions concurrently accessing the same Database, then we need to prevent the actions of the transactions from interfering with each other. It can be achieved by the isolation levels.

5.1. READ COMMITTED: Specifies that shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in nonrepeatable reads or phantom data. This option is the SQL Server default.READ_COMMITTED isolation level states that a transaction can't read data that is not yet committed by other transactions.

READ UNCOMMITTED**:** Implements dirty read, or isolation level 0 locking, which means that no shared locks are issued and no exclusive locks are honored. When this option is set, it is possible to read uncommitted or dirty data; values in the data can be changed and rows can appear or disappear in the data set before the end of the transaction. This option has the same effect as setting NOLOCK on all tables in all SELECT statements in a transaction. This is the least restrictive of the four isolation levels.

In this example Transaction A writes a record. Meanwhile Transaction B reads that same record before Transaction A commits. Later Transaction A decides to rollback and now we have changes in Transaction B that are inconsistent. This is a dirty read. Transaction B was running in READ_UNCOMMITTED isolation level so it was able to read Transaction A changes before a commit occurred.

5.2 REPEATABLE READ: Locks are placed on all data that is used in a query, preventing other users from updating the data, but new phantom rows can be inserted into the data set by another user and are included in later reads in the current transaction. Because concurrency is lower than the default isolation level, use this option only when necessary.

5.3 SERIALIZABLE

Places a range lock on the data set, preventing other users from updating or inserting rows into the data set until the transaction is complete. This is the most restrictive of the four isolation levels. Because concurrency is lower, use this option only when necessary. This option has the same effect as setting HOLDLOCK on all tables in all SELECT statements in a transaction. SERIALIZABLE transaction isolation level is the default isolation level for the COM+ application

5.4

	dirty reads	non-repeatable reads	phantom reads
READ_UNCOMMITTED	yes	yes	yes
READ_COMMITTED	no	yes	yes
REPEATABLE_READ	no	no	yes
SERIALIZABLE	no	no	no

It provides the highest level of isolation. If set then it helps us to prevent all problems like dirty reads, non-repeatable reads and phantom reads. Transactions are executed with locking at all levels (read, range and write locking) so they appear as if they were executed in a serialized way.

Locking: A solution to problems arising due to concurrency. Locking of records can be used as a concurrency control technique to prevent the above mentioned problems. A transaction acquires a lock on a record if it does not want the record values to be changed by some other transaction during a period of time. The transaction releases the lock after this time.

VI. CONCLUSION

Distributed database system is considered to be more reliable than centralized database system. We also described the concurrency control mechanism. Many organizations are now deploying distributed database systems. Therefore, we have no choice but to ensure that these systems operate in a secure environment and integrity[1]. Security is concerned with the assurance of confidentiality, integrity, and availability of information in all forms. There are many tools and techniques that can support the management of distributed database security. We discuss the basic concept of concurrency control in distributed database systems and also issued the various techniques for concurrency control in distributed environments. ACID properties of database is of utmost importance and it has to be maintained while concurrently accessing the database.

REFERENCES

[1] Manoj Kumar Sah 1, Vinod Kumar 2, Ashish Tiwari , “Security and Concurrency Control in Distributed Database System”,International Journal of scientific research and management (IJSRM), Volume 2, Issue 12 2014,, Pages 1839-1845, ISSN (e): 2321-3418

[2] Gupta V.K., Sheetlani Jitendra, Gupta Dhiraj and Shukla Brahma Datta, “Concurrency Control and Security issues of Distributed Databases Transaction”, International Science Congress Association - Research Journal of Engineering, Vol. 1(2), 70-73, August 2012, ISSN 2278 – 9472.

[3] Daniel A. Menasce And Richard R. Muntz, Member, IEEE, “Locking And Deadlock Detection In Distributed Data Bases” IEEE Transactions On Software Engineering, Vol. Se-5, No. 3, May 1979

[4] Prof. Vinayak Sinde, Preeti A. Aware, “Concurrency Control In Distributed Database Systems ”, International Journal For Research In Engineering Application & Management (Ijream) ISSN : 2494-9150 Vol-01, Issue 10, Jan 2016.

[5] Gupta V.K., Sheetlani Jitendra, Gupta Dhiraj and Shukla Brahma Datta, "Concurrency control and Security issues in Distributed Database system", Vol. 1(2), 70-73, August (2012)