**Concurrency Control in DBMS**

In a database management system (DBMS), allowing transactions to run concurrently has significant advantages, such as better system resource utilization and higher throughput. However, it is crucial that these transactions do not conflict with each other. The ultimate goal is to ensure that the database remains consistent and accurate. For instance, if two users try to book the last available seat on a flight at the same time, the system must ensure that only one booking succeeds. Concurrency control is a critical mechanism in DBMS that ensures the consistency and integrity of data when multiple operations are performed at the same time.

* Concurrency control is a concept in Database Management Systems (DBMS) that ensures multiple transactions can simultaneously access or modify data without causing errors or inconsistencies. It provides mechanisms to handle the concurrent execution in a way that maintains ACID properties.
* By implementing concurrency control, a DBMS allows transactions to execute concurrently while avoiding issues such as deadlocks, race conditions, and conflicts between operations.
* The main goal of concurrency control is to ensure that simultaneous transactions do not lead to data conflicts or violate the consistency of the database. The concept of serializability is often used to achieve this goal.

**Concurrent Execution and Related Challenges in DBMS**

In a multi-user system, several users can access and work on the same database at the same time. This is known as concurrent execution, where the database is used simultaneously by different users for various operations. For instance, one user might be updating data while another is retrieving it.

When multiple transactions are performed on the database simultaneously, it is important that these operations are executed in an interleaved manner. This means that the actions of one user should not interfere with or affect the actions of another. This helps in maintaining the consistency of the database. However, managing such simultaneous operations can be challenging, and certain problems may arise if not handled properly. These challenges need to be addressed to ensure smooth and error-free concurrent execution.

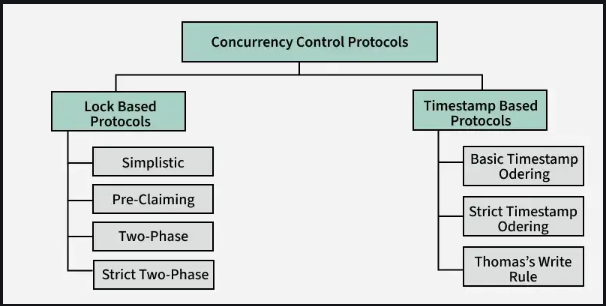
**Concurrent Execution can lead to various challenges:**

* **In-consistent Data:** When multiple users are accessing and modifying the data in a transaction, it may lead to inconsistent data states. For example, if there is inadequate concurrency management and two transactions are modifying the same record at the same time, the final state of the data may only reflect one of the updates, which could cause inconsistent results.
* **Lost Updates:**When two or more transactions read the same piece of data and then update it according to the real value, this happens. The final value represents only the most recent update, so “losing” the earlier updates since no transaction can observe the updates performed by prior transactions.
* **Uncommitted Data:**When a transaction accesses data that has been altered by another transaction but has not yet been committed, it can lead to uncommitted data issues, also known as dirty reads. The data that was read by the first transaction becomes invalid if the transaction that made the modifications is rolled back. A second transaction might be working with uncommitted data that could be rolled back, for instance, if it accesses a record that has been updated by the first transaction before it commits.
* **Inconsistent Retrievals:**When a transaction reads the same data again and receives different results each time due to another transaction changing the data in between the reads, this is known as an inconsistent retrieval or non-repeatable read. The operation of the transaction may become an anomaly as a result.

For example, if a transaction reads a collection of data, another transaction modifies that data, and then the first transaction reads the data again and observes different values, and this causes an inconsistent retrieval.

**Concurrency Control Protocols**

Concurrency control protocols are the set of rules which are maintained in order to solve the concurrency control problems in the database. It ensures that the concurrent transactions can execute properly while maintaining the database consistency. The concurrent execution of a transaction is provided with atomicity, consistency, isolation, durability, and serializability via the concurrency control protocols.



**Lock Based Concurrency Control Protocol in DBMS**

In a Database Management System (DBMS), lock-based concurrency control (BCC) is a method used to manage how multiple transactions access the same data. This protocol ensures data consistency and integrity when multiple users interact with the database simultaneously.

This method uses locks to manage access to data, ensuring transactions don’t clash and everything runs smoothly when multiple transactions happen at the same time. In this article, we’ll take a closer look at how the Lock-Based Protocol works.

**What is a Lock?**

A lock is a variable associated with a data item that indicates whether it is currently in use or available for other operations. Locks are essential for managing access to data during concurrent transactions. When one transaction is accessing or modifying a data item, a lock ensures that other transactions cannot interfere with it, maintaining data integrity and preventing conflicts. This process, known as locking, is a widely used method to ensure smooth and consistent operation in database systems.

**Lock Based Protocols**

Lock-Based Protocols in DBMS ensure that a transaction cannot read or write data until it gets the necessary lock. Here’s how they work:

* These protocols prevent concurrency issues by allowing only one transaction to access a specific data item at a time.
* Locks help multiple transactions work together smoothly by managing access to the database items.
* Locking is a common method used to maintain the serializability of transactions.
* A transaction must acquire a read lock or write lock on a data item before performing any read or write operations on it.

**Types of Lock**

**Shared Lock (S):** Shared Lock is also known as Read-only lock. As the name suggests it can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item. S-lock is requested using lock-S instruction.

**Exclusive Lock (X):** Data item can be both read as well as written. This is Exclusive and cannot be held simultaneously on the same data item. X-lock is requested using lock-X instruction.

### Rules of Locking

### The basic rules for Locking are given below :

### Read Lock (or) Shared Lock(S)

### If a Transaction has a Read lock on a data item, it can read the item but not update it.

### If a transaction has a Read lock on the data item, other transaction can obtain Read Lock on the data item but no Write Locks.

### So, the Read Lock is also called a Shared Lock.

### Write Lock (or) Exclusive Lock (X)

### If a transaction has a write Lock on a data item, it can both read and update the data item.

### If a transaction has a write Lock on the data item, then other transactions cannot obtain either a Read lock or write lock on the data item.

### So, the Write Lock is also known as Exclusive Lock.

**Concurrency Control Protocols**

Concurrency Control Protocols are the methods used to manage multiple transactions happening at the same time. They ensure that transactions are executed safely without interfering with each other, maintaining the accuracy and consistency of the database.

These protocols prevent issues like data conflicts, lost updates or inconsistent data by controlling how transactions access and modify data.

**Types of Lock-Based Protocols**

### 1. Simplistic Lock Protocol

### It is the simplest method for locking data during a transaction. Simple lock-based protocols enable all transactions to obtain a lock on the data before inserting, deleting, or updating it. It will unlock the data item once the transaction is completed.  Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

### Example:

### Consider a database with a single data item X = 10.

### Transactions:

### T1: Wants to read and update X.

### T2: Wants to read X.

### Steps:

(1)

### T1 requests an exclusive lock on X to update its value. The lock is granted.

### T1 reads X = 10 and updates it to X = 20.

### (2)

### T2 requests a shared lock on X to read its value. Since T1 is holding an exclusive lock, T2 must wait.

### (3)

### T1 completes its operation and releases the lock.

### (4)

### T2 now gets the shared lock and reads the updated value X = 20.

### This example shows how simplistic lock protocols handle concurrency but do not prevent problems like deadlocks or limits concurrency.

**2. Pre-Claiming Lock Protocol**

The Pre-Claiming Lock Protocol evaluates a transaction to identify all the data items that require locks. Before the transaction begins, it requests the database management system to grant locks on all necessary data elements. If all the requested locks are successfully acquired, the transaction proceeds. Once the transaction is completed, all locks are released. However, if any of the locks are unavailable, the transaction rolls back and waits until all required locks are granted before restarting.

**Example:**

Consider two transactions T1 and T2 and two data items, X and Y:

(1)

Transaction T1 declares that it needs:

* A write lock on X.
* A read lock on Y.

Since both locks are available, the system grants them. T1 starts execution:

* It updates X.
* It reads the value of Y.

(2)

While T1 is executing, Transaction T2 declares that it needs:

* A read lock on X.

However, since T1 already holds a write lock on X, T2’s request is denied. T2 must wait until T1 completes its operations and releases the locks.

(3)

Once T1 finishes, it releases the locks on X and Y. The system now grants the read lock on X to T2, allowing it to proceed.

This method is simple but may lead to inefficiency in systems with a high number of transactions.

### Two-phase locking

Two-phase locking (2PL) is one of the most widely used concurrency control protocols, particularly in centralized, single-node [relational databases](https://aerospike.com/blog/what-is-a-relational-database/). It requires transactions to acquire locks before accessing data. Lock-based protocols, in general, obtain a lock before they operate on an item. 2PL is a type of lock-based protocol that employs two types of locks:

* Read lock (shared lock): Allows multiple transactions to read an item simultaneously.
* Write lock (exclusive lock): Prevents other transactions from acquiring any lock (read or write) on the same item.

This locking behavior aligns with the notion of conflicts introduced in the previous section.

2PL consists of two phases:

1. Growing phase: During this phase, a transaction can obtain (acquire) any number of locks as required but cannot release any. This phase continues until the transaction acquires all the locks it needs and no longer requests.
2. Shrinking phase: Once the transaction releases its first lock, the Shrinking phase starts. During this phase, the transaction can release but not acquire any more locks.
3. The moment when a transaction has acquired all its required locks is known as the lock point. The schedule produced by 2PL is equivalent to a serial schedule where transactions are ordered by their lock points.

### Strict two-phase locking (Strict 2PL)

In standard 2PL, a transaction releases locks as soon as the locks are no longer needed once the transaction has reached its lock point. These can lead to schedules that are not recoverable or cause cascading aborts as described below:

* The schedule is not recoverable: A committed transaction may have read values written by another transaction that later aborts. This is not a desirable situation.
* Schedule causes cascading aborts: The failure of one transaction may force multiple dependent transactions to abort.

Most databases implement strict 2PL to mitigate these issues, where all locks are held until the transaction commits or aborts.

**Cascadeless and Recoverable Schedules in Concurrency Control**

**1. Recoverable Schedules**

A recoverable schedule ensures that a transaction commits only if all the transactions it depends on have committed. This avoids situations where a committed transaction depends on an uncommitted transaction that later fails, leading to inconsistencies.

Concurrency control ensures recoverable schedules by keeping track of which transactions depend on others. It makes sure a transaction can only commit if all the transactions it relies on have already committed successfully. This prevents issues where a committed transaction depends on one that later fails.

Techniques like strict two-phase locking (2PL) enforce recoverability by delaying the commit of dependent transactions until the parent transactions have safely committed.

**2. Cascadeless Schedules**

A cascadeless schedule avoids cascading rollbacks, which occur when the failure of one transaction causes multiple dependent transactions to fail.

Concurrency control techniques such as strict 2PL or timestamp ordering ensure cascadeless schedules by ensuring dependent transactions only access committed data.

By delaying read or write operations until the transaction they depend on has committed, cascading rollbacks are avoided.

**Advantages of Concurrency**

In general, concurrency means that more than one transaction can work on a system. The advantages of a concurrent system are:

**Waiting Time:** It means if a process is in a ready state but still the process does not get the system to get execute is called waiting time. So, concurrency leads to less waiting time.

**Response Time:** The time wasted in getting the response from the CPU for the first time, is called response time. So, concurrency leads to less Response Time.

**Resource Utilization:** The amount of Resource utilization in a particular system is called Resource Utilization. Multiple transactions can run parallel in a system. So, concurrency leads to more Resource Utilization.

**Efficiency:** The amount of output produced in comparison to given input is called efficiency. So, Concurrency leads to more Efficiency.

**Disadvantages of Concurrency**

**Overhead:** Implementing concurrency control requires additional overhead, such as acquiring and releasing locks on database objects. This overhead can lead to slower performance and increased resource consumption, particularly in systems with high levels of concurrency.

**Deadlocks:** Deadlocks can occur when two or more transactions are waiting for each other to release resources, causing a circular dependency that can prevent any of the transactions from completing. Deadlocks can be difficult to detect and resolve, and can result in reduced throughput and increased latency.

**Reduced concurrency:** Concurrency control can limit the number of users or applications that can access the database simultaneously. This can lead to reduced concurrency and slower performance in systems with high levels of concurrency.

**Complexity:** Implementing concurrency control can be complex, particularly in distributed systems or in systems with complex transactional logic. This complexity can lead to increased development and maintenance costs.

**Inconsistency:** In some cases, concurrency control can lead to inconsistencies in the database. For example, a transaction that is rolled back may leave the database in an inconsistent state, or a long-running transaction may cause other transactions to wait for extended periods, leading to data staleness and reduced accuracy.