Concurrency control

6.2

Concurrency control

- Executing a single transaction at a time will increase the waiting time of the other transactions which may result in delay in the overall execution.
- Hence for increasing the overall throughput and efficiency of the system, several transactions are executed.
- Concurrency control is a very important concept of DBMS which ensures the simultaneous execution or manipulation of data by several processes or user without resulting in data inconsistency.
- Concurrency control provides a procedure that is able to control concurrent execution of the operations in the database.

Concurrency Control Problems

- The database transaction consist of two major operations “Read” and “Write”.
- It is very important to manage these operations in the concurrent execution of the transactions in order to maintain the consistency of the data.
- **Dirty Read Problem(Write-Read conflict)**
- Dirty read problem occurs when one transaction updates an item but due to some unconditional events that transaction fails but before the transaction performs rollback, some other transaction reads the updated value. Thus creates an inconsistency in the database.
- **Lost Update Problem**
- Lost update problem occurs when two or more transactions modify the same data, resulting in the update being overwritten or lost by another transaction.

Concurrency Control Protocols

- Two-phase locking Protocol
- Time stamp ordering Protocol
- Multi version concurrency control
- Validation concurrency control

Two-phase locking Protocol

- It's called "two-phase" because, during each transaction, there are two distinct phases: the Growing phase and the Shrinking phase.
- **Phases**:
  - **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.
  - In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.
  - **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.
  - In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.
- The primary purpose of the Two-Phase Locking protocol is to ensure conflict-serializability, as the protocol ensures a transaction does not interfere with others in ways that produce inconsistent results

2. Time stamp ordering Protocol

- The **Timestamp Ordering Protocol** is a concurrency control method used in database management systems to maintain the serializability of transactions.
- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps.
- The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first.
- To determine the timestamp of the transaction, this protocol uses system time or logical counter.

- Example:
- Let’s assume there are two transactions T1 and T2.
- Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times.
- T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data

- **Basic Timestamp ordering protocol works as follows:**

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:

- If W_TS(X) >TS(Ti) then the operation is rejected.
- If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

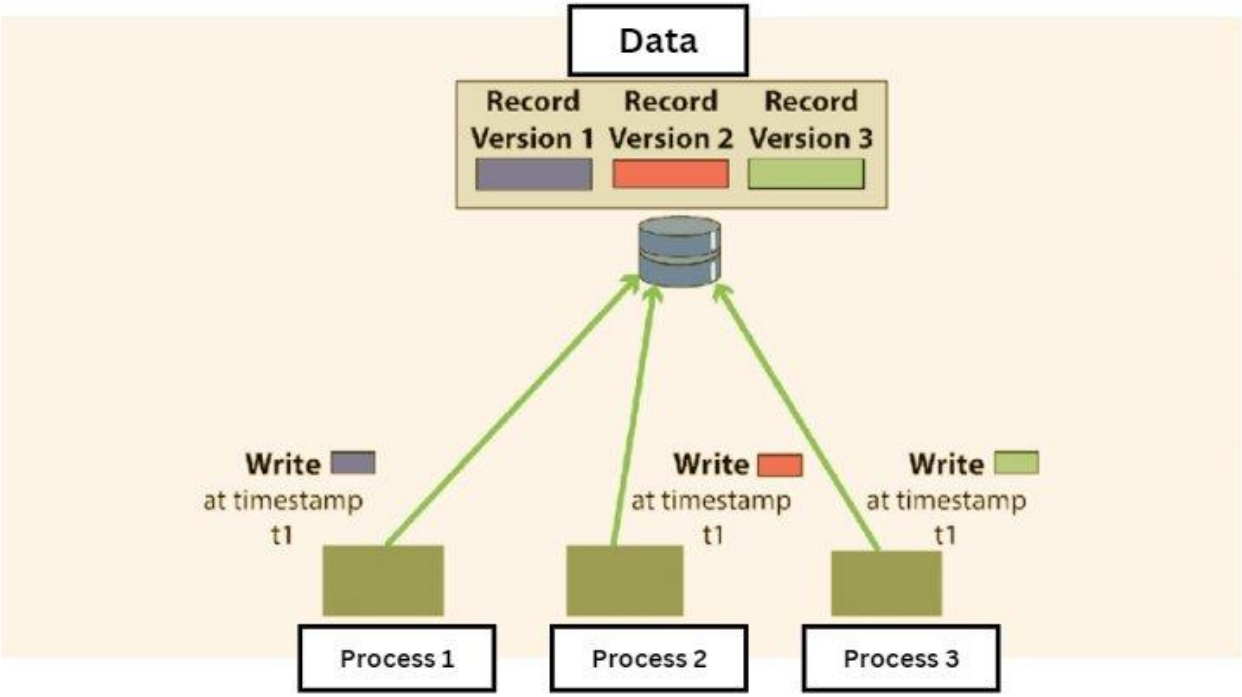
2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:

- If TS(Ti) < R_TS(X) then the operation is rejected.
- If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

- Where,
- **TS(TI)** denotes the timestamp of the transaction Ti.
- **R_TS(X)** denotes the Read time-stamp of data-item X.
- **W_TS(X)** denotes the Write time-stamp of data-item X.

Multi version concurrency control

- **It is** a technique used in database management systems to handle concurrent operations without conflicts, using multiple versions of a data item. Instead of locking the items for write operations (which can reduce concurrency and lead to bottlenecks or deadlocks), MVCC will create a separate version of the data item being modified.



Breakdown of the Multi version concurrency control (MVCC)

- Multiple Versions: When a transaction modifies a data item, instead of changing the item in place, it creates a new version of that item. This means that multiple versions of a database object can exist simultaneously.
- Reads aren't Blocked: One of the significant advantages of MVCC is that read operations don't get blocked by write operations. When a transaction reads a data item, it sees a version of that item consistent with the last time it began a transaction or issued a read, even if other transactions are currently modifying that item.
- Timestamps or Transaction IDs: Each version of a data item is tagged with a unique identifier, typically a timestamp or a transaction ID. This identifier determines which version of the data item a transaction sees when it accesses that item. A transaction will always see its own writes, even if they are uncommitted.
- Garbage Collection: As transactions create newer versions of data items, older versions can become obsolete. There's typically a background process that cleans up these old versions, a procedure often referred to as "garbage collection."
- Conflict Resolution: If two transactions try to modify the same data item concurrently, the system will need a way to resolve this. Different systems have different methods for conflict resolution. A common one is that the first transaction to commit will succeed, and the other transaction will be rolled back or will need to resolve the conflict before proceeding

Validation Based Protocol

- Validation phase is also known as optimistic concurrency control technique.
- In the validation based protocol, the transaction is executed in the following three phases:
  - **Read phase:**
  - In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
  - **Validation phase:**
  - In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.
  - **Write phase:**
  - If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

- Here each phase has the following different timestamps:
- **Start(Ti):** It contains the time when Ti started its execution.
- **Validation (T_i):** It contains the time when Ti finishes its read phase and starts its validation phase.
- **Finish(Ti):** It contains the time when Ti finishes its write phase

- This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.
- Hence TS(T) = validation(T).
- The serializability is determined during the validation process. It can't be decided in advance.
- While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.
- Thus it contains transactions which have less number of rollbacks.

Deadlock

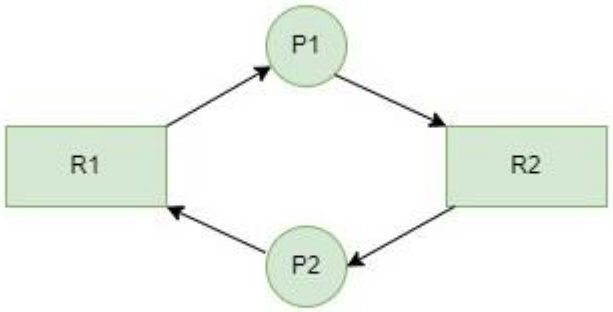
- A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks.
- Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.
- **Example:**
- If there are three processes p1,p2 and p1 are acquiring r1 resource and that r1 is needed by p2 which is acquiring another resource r2 and that is needed by p1. Here cycle occurs. It is called a deadlock.

Deadlock Avoidance

- When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

Resource allocation graph

- P1,p2 –process
- R1,r2- resources



Deadlock Detection

- In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not.
- The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

- The deadlock Detection Algorithm is of two types:
  - Wait-for-Graph Algorithm (Single Instance)
  - Banker's Algorithm (Multiple Instance)

Wait-for-Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.
- It is a variant of the Resource Allocation graph. In this algorithm, we only have processes as vertices in the graph. If the Wait-for-Graph contains a cycle then we can say the system is in a Deadlock state.
- The Resource Allocation graph will be converted into Wait-for-Graph in an Algorithmic Approach.
- We need to remove resources while converting from Resource Allocation Graph to Wait-for-Graph.

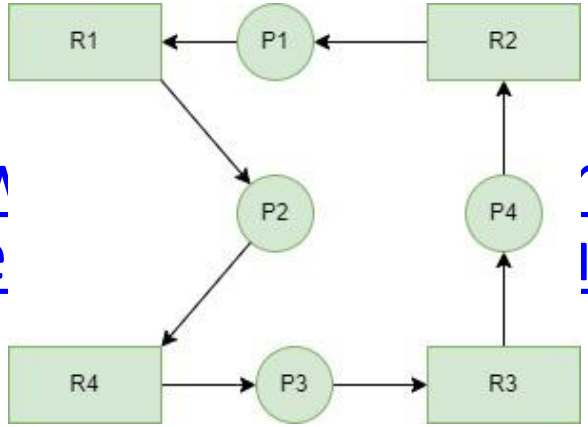
Wait for graph algorithm

- **Step 1:** Take the first process (Pi) from the resource allocation graph and check the path in which it is acquiring resource (R_i), and start a wait-for-graph with that particular process.
- **Step 2:** Make a path for the Wait-for-Graph in which there will be no Resource included from the current process (P_i) to next process (P_j), from that next process (P_j) find a resource (R_j) that will be acquired by next Process (P_k) which is released from Process (P_j).
- **Step 3:** Repeat Step 2 for all the processes.
- **Step 4:** After completion of all processes, if we find a closed-loop cycle then the system is in a deadlock state, and deadlock is detected.

Example

- Consider a Resource Allocation Graph with 4 Processes P1, P2, P3, P4, and 4 Resources R1, R2, R3, R4. Find if there is a deadlock in the Graph using the Wait for Graph-based deadlock detection algorithm.

- https://www[illegible]g/wait-for-graph-deadlock-de[illegible]ited-system/



- Step1:
- Process P1 which is waiting for Resource R1, resource R1 is acquired by Process P2, Start a Wait-for-Graph for the above Resource Allocation Graph.
- **Step 2:** Now we can observe that there is a path from P1 to P2 as P1 is waiting for R1 which is been acquired by P2. Now the Graph would be after removing resource R1 looks like.

- **Step 3:** From P2 we can observe a path from P2 to P3 as P2 is waiting for R4 which is acquired by P3. So make a path from P2 to P3 after removing resource R4 looks like.
-
- **Step 4:** From P3 we find a path to P4 as it is waiting for P3 which is acquired by P4. After removing R3 the graph looks like this.
-
- **Step 5:** Here we can find Process P4 is waiting for R2 which is acquired by P1. So finally the Wait-for-Graph is as follows:
-

- **Step 6:** Finally In this Graph, we found a cycle as the Process P4 again came back to the Process P1 which is the starting point (i.e., it's a closed-loop).
- So, According to the Algorithm if we found a closed loop, then the system is in a deadlock state. So here we can say the system is in a deadlock state.

Recovery system

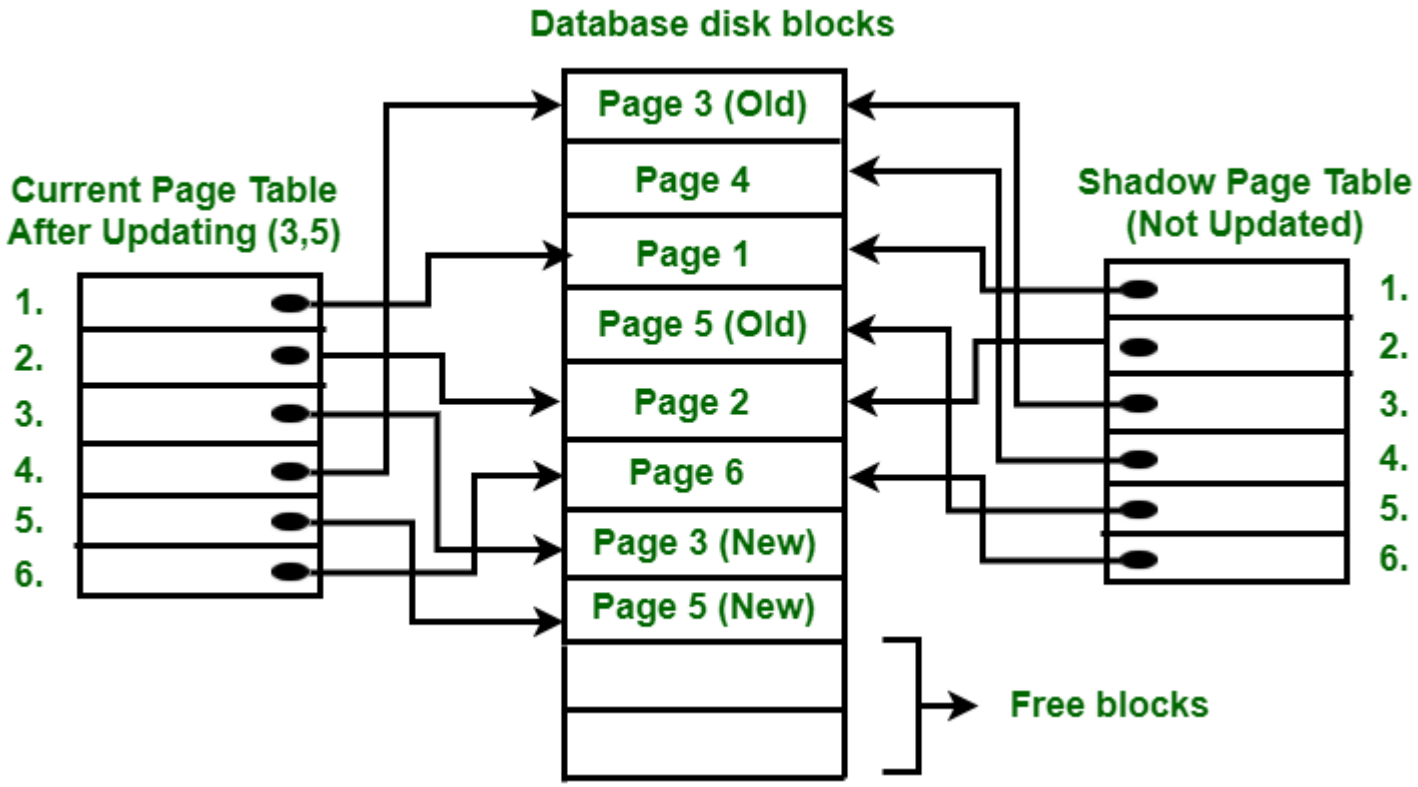
- There are two different approaches:
  - log-based recovery, and
  - shadow-paging

log-based recovery

- A log is kept on stable storage.
- The log is a sequence of log records, and maintains a record of update activities on the database.
- When transaction Ti starts, it registers itself by writing a < Ti start > log record
- Before Ti executes write(X), a log record < Ti, X, V1, V2 > is written, where V1 is the value of X before the write, and V2 is the value to be written to X.
- When Ti finishes it last statement, the log record < Ti commit > is written.
- We assume for now that log records are written directly to stable storage (that is, they are not buffered)

Shadow paging

- **Shadow Paging** is recovery technique that is used to recover database.
- In this recovery technique, database is considered as made up of fixed size of logical units of storage which are referred as **pages.**
- pages are mapped into physical blocks of storage, with help of the **page table** which allow one entry for each logical page of database.
- This method uses two page tables named **current page table** and **shadow page table**.
- The entries which are present in current page table are used to point to most recent database pages on disk.
- Another table i.e., Shadow page table is used when the transaction starts which is copying current page table.
- After this, shadow page table gets saved on disk and current page table is going to be used for transaction.
- Entries present in current page table may be changed during execution but in shadow page table it never get changed.
- After transaction, both tables become identical. This technique is also known as **Cut-of-Place updating.**


Database disk blocks
Page 3 (Old)
Page 4
Page 1
Page 5 (Old)
Page 2
Page 6
Page 3 (New)
Page 5 (New)
Current Page Table
After Updating (3,5)
1.
2.
3.
4.
5.
6.
Shadow Page Table
(Not Updated)
1.
2.
3.
4.
5.
6.
Free blocks

- To understand concept, consider above figure. In this 2 write operations are performed on page 3 and 5. Before start of write operation on page 3, current page table points to old page 3. When write operation starts following steps are performed :
- Firstly, search start for available free block in disk blocks.
- After finding free block, it copies page 3 to free block which is represented by Page 3 (New).
- Now current page table points to Page 3 (New) on disk but shadow page table points to old page 3 because it is not modified.
- The changes are now propagated to Page 3 (New) which is pointed by current page table.

- **COMMIT Operation :**

To commit transaction following steps should be done :

- All the modifications which are done by transaction which are present in buffers are transferred to physical database.
- Output current page table to disk.
- Disk address of current page table output to fixed location which is in stable storage containing address of shadow page table.
- This operation overwrites address of old shadow page table. With this current page table becomes same as shadow page table and transaction is committed.

- **Failure :** If system crashes during execution of transaction but before commit operation, With this, it is sufficient only to free modified database pages and discard current page table.
- Before execution of transaction, state of database get recovered by reinstalling shadow page table.
- If the crash of system occur after last write operation then it does not affect propagation of changes that are made by transaction.
- These changes are preserved and there is no need to perform redo operation.