UNIT IX – Transaction & Concurrency Control

Notes By: Raju Poudel (Mechi Multiple Campus)

Transaction Concept

- ✓ The transaction is a set of logically related operation. It contains a group of tasks.
- ✓ A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

- ✓ Let's take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A's Account

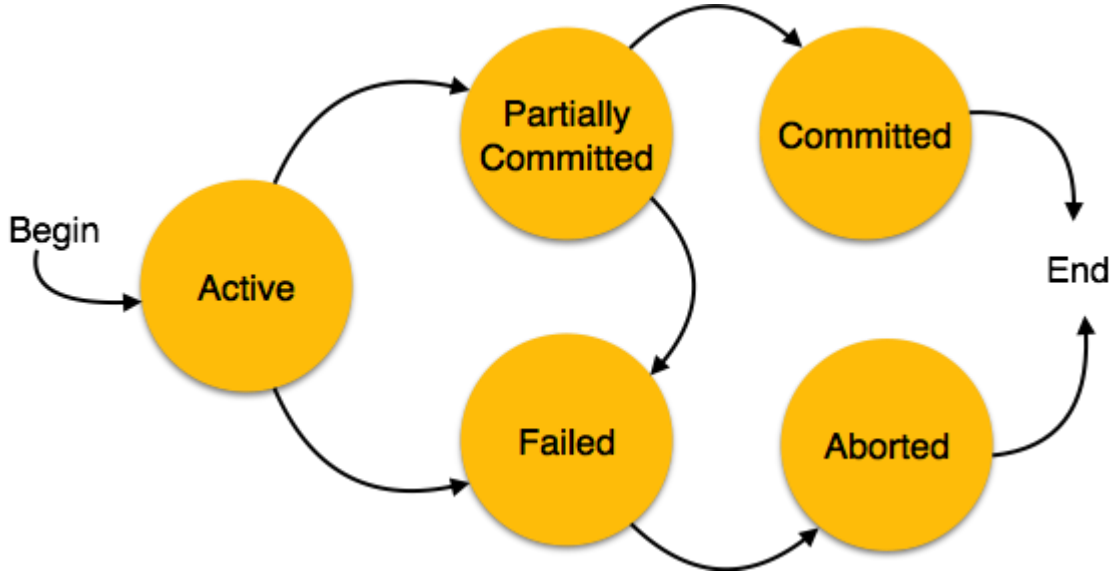
```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

B's Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

States of Transactions / Transaction Model

✓ A transaction in a database can be in one of the following states −



1) **Active −** In this state, the transaction is being executed. This is the initial state of every transaction.
2) **Partially Committed −** When a transaction executes its final operation, it is said to be in a partially committed state.

States of Transactions

3) **Failed –** A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
4) **Aborted –** If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts −
    - Re-start the transaction
    - Kill the transaction
5) **Committed –** If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

ACID Properties

- ✓ A transaction is a very small unit of a program and it may contain several low-level tasks.
- ✓ A transaction in a database system must maintain **Atomicity, Consistency, Isolation, and Durability** − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

1) <u>Atomicity</u>

- This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none.
- There must be no state in a database where a transaction is left partially completed.
- States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

ACID Properties

1) Atomicity

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then amount has been deducted from **X** but not added to **Y**.

This results in an inconsistent database state.

Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

ACID Properties

2) Consistency

- The database must remain in a consistent state after any transaction.
- No transaction should have any adverse effect on the data residing in the database.
- If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- Referring to the example above, The total amount before and after the transaction must be maintained.

Total **before T** occurs = **500 + 200 = 700**.

Total **after T occurs** = **400 + 300 = 700**.

Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result T is incomplete.

ACID Properties

3) Isolation

- In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.
- No transaction will affect the existence of any other transaction.

4) Durability

- The database should be durable enough to hold all its latest updates even if the system fails or restarts.
- If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data.
- If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

Isolation Example

Let **X= 500, Y = 500.**

Consider two transactions **T** and **T''.**

T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result , interleaving of operations takes place due to which **T''** reads correct value of **X** but incorrect value of **Y** and sum computed by

T'': (X+Y = 50, 000+500=50, 500)

is thus not consistent with the sum at end of transaction:

T: (X+Y = 50, 000 + 450 = 50, 450).

This results in database **inconsistency**, due to a **loss of 50 units**. Hence, transactions must take place in **isolation** and changes should be visible only after a they have been made to the main memory.

Serializability

- ✓ When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state.
- ✓ **Serializability** is a concept that helps us to check which schedules are serializable.
- ✓ A **serializable schedule** is the one that always leaves the database in consistent state.

- ✓ **Schedule −** A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
- ✓ **Serial Schedule −** It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

Types of Serializability

- ✓ There are two types of Serializability –
  1. Conflict Serializability
  2. View Serializability

1) Conflict Serializability

- ✓ Conflict Serializability is one of the type of Serializability, which can be used to check whether a non-serial schedule is conflict serializable or not.
- ✓ A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

Conflicting operations

Two operations are said to be in conflict, if they satisfy all the following three conditions:

1. Both the operations should belong to different transactions.
2. Both the operations are working on same data item.
3. At least one of the operation is a write operation.

Example of Conflict Serializability

Lets consider this schedule:

```
T1          T2
-----       ------
R(A)
R(B)
            R(A)
            R(B)
            W(B)
W(A)
```

To convert this schedule into a serial schedule we must have to swap the **R(A)** operation of transaction T2 with the **W(A)** operation of transaction T1.

However we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is **not Conflict Serializable**.

Example of Conflict Serializability

Lets take another example:

```
T1          T2
-----       ------
R(A)
            R(A)
            R(B)
            W(B)
R(B)
W(A)
```

Lets **swap non-conflicting operations**:

After swapping R(A) of T1 and R(A) of T2 we get:

```
T1          T2
-----       ------
            R(A)
R(A)
            R(B)
            W(B)
R(B)
W(A)
```

After swapping R(A) of T1 and R(B) of T2 we get:

```
T1          T2
-----       ------
            R(A)
            R(B)
R(A)
            W(B)
R(B)
W(A)
```

After swapping R(A) of T1 and W(B) of T2 we get:

```
T1          T2
-----       ------
            R(A)
            R(B)
            W(B)
R(A)
R(B)
W(A)
```

We finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is **Conflict Serializable**.

View Serializability

- ✓ View Serializability is a process to find out that a given schedule is view serializable or not.
- ✓ Two schedules S1 and S2 are said to be view equal if below conditions are satisfied :

1) Initial Read

If a transaction T1 reading data item A from initial database in S1 then in S2 also T1 should read A from initial database.

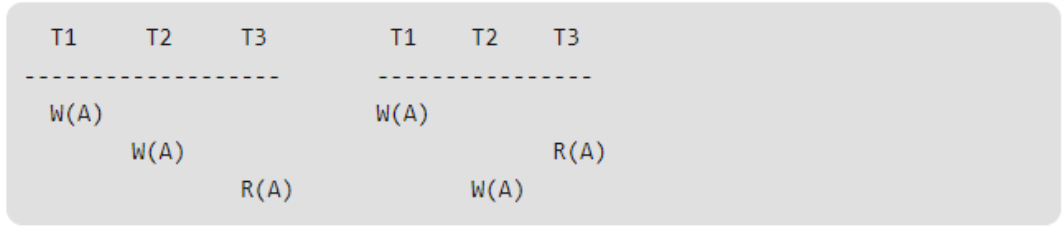
```
  T1      T2      T3
------------------
         R(A)
 W(A)
                 R(A)
         R(B)
```

Transaction T2 is reading A from initial database.

View Serializability

2) Updated Read

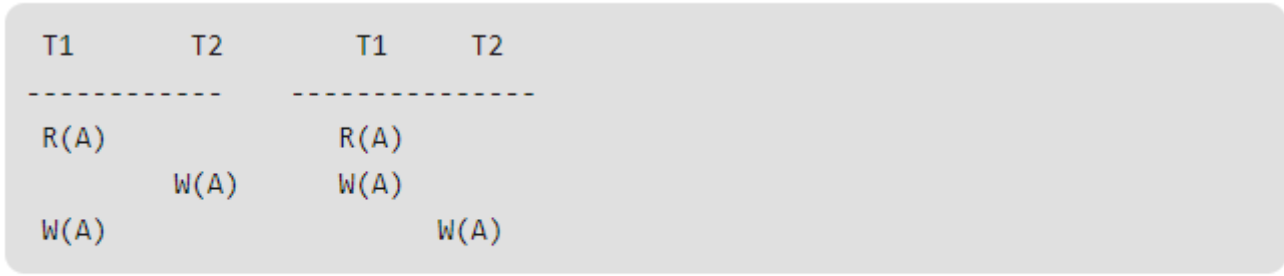
If Ti is reading A which is updated by Tj in S1 then in S2 also Ti should read A which is updated by Tj.



Above two schedule are not view equal as in S1 :T3 is reading A updated by T2, in S2 T3 is reading A updated by T1.

3) Final Write operation

If a transaction T1 updated A at last in S1, then in S2 also T1 should perform final write operations.



Above two schedule are not view as Final write operation in S1 is done by T1 while in S2 done by T2.

Concurrency Control

- ✓ Concurrency control is the procedure in DBMS **for managing simultaneous operations** without conflicting with each another.
- ✓ Concurrent access is quite **easy** if all users are **just reading data**. There is no way they can interfere with one another.
- ✓ Though for any practical database, would have a **mix of reading and WRITE operations** and hence the concurrency is a challenge.

- ✓ Concurrency control is used to address such conflicts which mostly occur with a **multi-user system**.
- ✓ It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases.
- ✓ Therefore, concurrency control is a most important element for the proper functioning of a system where two or multiple database transactions that require access to the same data, are executed simultaneously.

Why use Concurrency method?

Reasons for using Concurrency control method is DBMS:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose.

- Lock-Based Protocols
- Two Phase
- Timestamp-Based Protocols
- Validation-Based Protocols

Lock-Based Protocols

- ✓ A **lock** is a **data variable** which is **associated** with a **data item**. This lock signifies that operations that can be performed on the data item.
- ✓ Locks help synchronize access to the **database items by concurrent transactions.**
- ✓ All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.

- ✓ **Binary Locks:** A Binary lock on a data item can either locked or unlocked states.

- ✓ **Shared/exclusive:** This type of locking mechanism separates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

Lock-Based Protocols

1. Shared Lock (S):

- ✓ A shared lock is also called a **Read-only lock**. With the shared lock, the data item can be shared between transactions.
- ✓ This is because you will never have permission to update data on the data item.

- ✓ For example, consider a case where two transactions are reading the account balance of a person.
- ✓ The database will let them read by placing a shared lock.
- ✓ However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

Lock-Based Protocols

2. Exclusive Lock (X):

- ✓ With the Exclusive Lock, a data item can be read as well as written.
- ✓ This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction.
- ✓ Transactions may unlock the data item after finishing the 'write' operation.

- ✓ For example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it.
- ✓ Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

Deadlock Handling

- ✓ Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

- ✓ 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.

Deadlock Example in DBMS

- ✓ For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table.
- ✓ Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

- ✓ Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock.
- ✓ All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

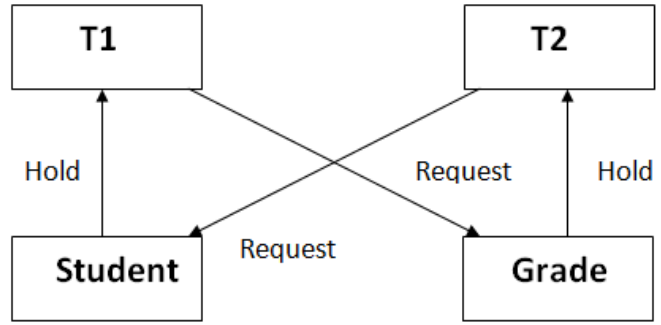


Figure: Deadlock in DBMS

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.

Deadlock Prevention

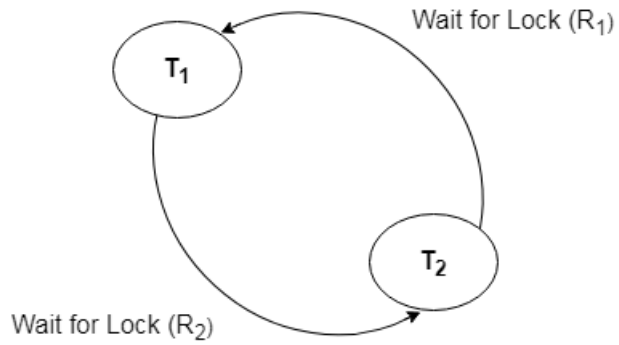
- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

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.

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.



Timestamp-based Protocols

- ✓ The timestamp-based algorithm uses a timestamp to serialize the execution of concurrent transactions. This protocol ensures that every conflicting read and write operations are executed in timestamp order.
- ✓ The protocol uses the **System Time or Logical Count as** a Timestamp.

- ✓ **The older transaction is always given priority** in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.
- ✓ Lock-based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created. **Example**,

```
Suppose there are there transactions T1, T2, and T3.
T1 has entered the system at time 0010
T2 has entered the system at 0020
T3 has entered the system at 0030
Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.
```