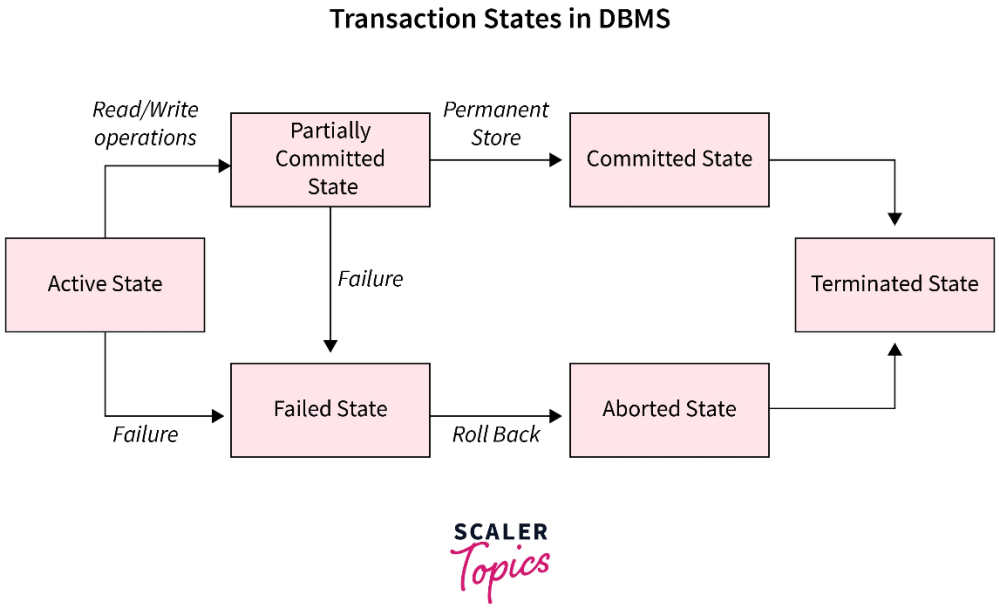
Q1. Draw and Explain the transaction state diagram

Ans)




We shall discuss all the different stages as can be seen in the diagram above.

Active state: This is the very first state of the transaction. All the read-write operations of the transaction are currently running then the transaction is in the active state. If there is any failure, it goes to the failed state. If all operations are successful then the transaction moves to the partially committed state. All the changes that are carried out in this stage are stored in the buffer memory.

Partially Committed state: Once all the instructions of the transaction are successfully executed, the transaction enters the Partially Committed state. If the changes are made permanent from the buffer memory, then the transaction enters the Committed state. Otherwise, if there is any failure, it enters the failed state. The main reason for this state is that every time a database operation is performed, a transaction can involve a large number of changes to the database, and if a power failure or other technical problem occurs when the system goes down because the transaction will result in Inconsistent changes to the database.

Committed state: Once all the operations are successfully executed and the transaction is out of the partially committed state, all the changes become permanent in the database. That is the Committed state. There's no going back! The changes cannot be rolled back and the transaction goes to the terminated state.

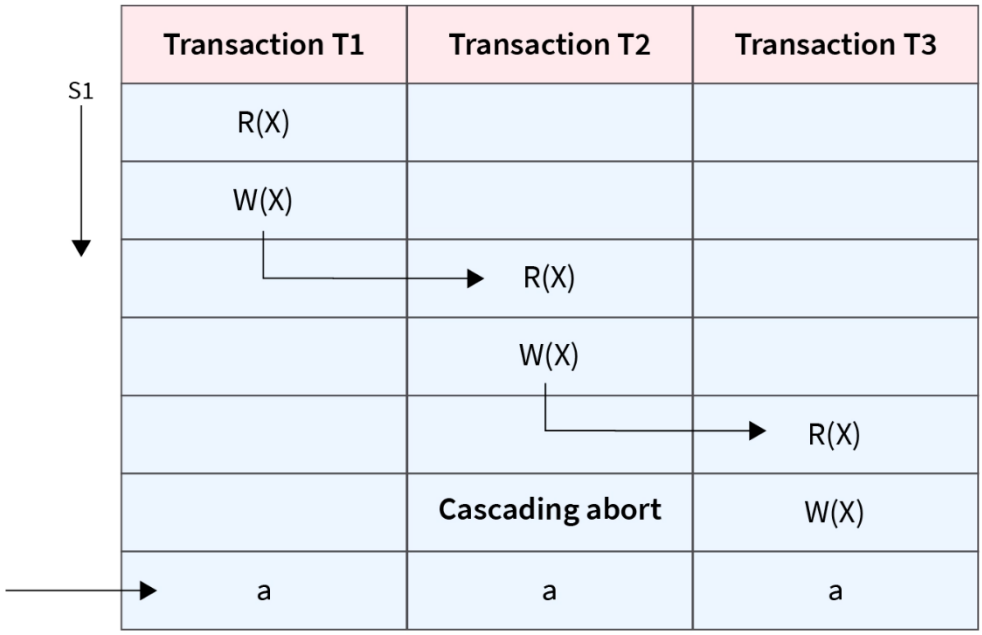
Failed state: In case there is any failure in carrying out the instructions while the transaction is in the active state, or there are any issues while saving the changes permanently into the database (i.e. in the partially committed stage) then the transaction enters the failed state.

Aborted state: If any of the checks fail and the transaction reaches a failed state, the database recovery system ensures that the database is in a previously consistent state. Otherwise, the transaction is aborted or rolled back, leaving the database in a consistent state. If a transaction fails in the middle of a transaction, all running transactions are rolled back to a consistent state before executing the transaction.

Terminated state: If a transaction is aborted, then there are two ways of recovering the DBMS, one is by restarting the task, and the other is by terminating the task and making itself free for other transactions. The latter is known as the terminated state.

Q2. Explain Cascading schedule.

Ans) If in a schedule, several other dependent transactions are forced to rollback/abort because of the failure of one transaction, then such a schedule is called a Cascading Schedule or Cascading Rollback or Cascading Abort. It simply leads to the wastage of CPU time.




Here, Transaction T2 depends on transaction T1, and transaction T3 depends on transaction T2. Thus, in this Schedule, the failure of transaction T1 will cause transaction T2 to roll back, and a similar case for transaction T3. Therefore, it is a cascading schedule. If the transactions T2 and T3 had been committed before the failure of transaction T1, then the Schedule would have been irrecoverable.

Q3. Illustrate with example conflict Serializability

Ans) A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. An operations pair become conflicting if all conditions satisfy:

Both belong to separate transactions.

They have the same data item.

They contain at least one write operation.

| Transaction T1 | Transaction T2 |
|---|---|
| Read(A) | |
| Write(A) | |
| | Read(A) |
| | Write(A) |
| Read(B) | |
| Write(B) | |
| | Read(B) |
| | Write(B) |

Before Swapping

| Transaction T1 | Transaction T2 |
|---|---|
| Read(A) | |
| Write(A) | |
| Read(B) | |
| Write(B) | |
| | Read(A) |
| | Write(A) |
| | Read(B) |
| | Write(B) |

After Swapping

In this schedule, Write(A)/Read(A) and Write(B)/Read(B) are called as conflicting operations. This is because all the above conditions hold true for them. Thus, by swapping(non-conflicting) 2nd pair of the read/write operation of 'A' data item and 1st pair of the read/write operation of 'B' data item, this non-serial Schedule can be converted into a serializable Schedule. Therefore, it is conflict serializable.

Q4. Explain Consistency requirement and Isolation with example of 2 transaction

Ans) Consistency: Consistency is one of the fundamental properties of transactions, ensuring that the database transitions from one valid state to another.

It guarantees the integrity of data by enforcing rules, constraints, and relationships defined in the database schema.

When a transaction completes successfully, the database remains in a consistent state.

For example, consider two transactions, T1 and T2:

**TRANSACTION 1(T1):**

BEGIN TRANSACTION;

UPDATE Customers SET Balance = Balance + 50 WHERE CustomerID = 1;

COMMIT;

**TRANSACTION 2(T2):**

BEGIN TRANSACTION;

UPDATE Customers SET Balance = Balance - 20 WHERE CustomerID = 3;

In this scenario, T2 modifies the same record that T1 is currently reading. To maintain consistency, T2 must wait until T1 completes its execution.

Isolation: Isolation ensures that concurrent transactions do not interfere with each other.

It prevents intermediate transaction states from being visible to others until committed.

Different isolation levels (such as Read Uncommitted, Read Committed, Repeatable Read, and Serializable) control the degree of isolation.

For instance, consider the following scenario:

**TRANSACTION 2(T2):**

BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

UPDATE Customers SET Balance = Balance - 20 WHERE CustomerID = 3;

T2 tries to modify the same record, but it is blocked by T1. T2 has to wait until T1 completes. The Repeatable Read isolation level ensures that T2 cannot modify the data that T1 is currently reading

Q5. Demonstrate Relation Decomposition in detail with example.

Ans) When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.

In a database, it breaks the table into multiple tables.

If the relation has no proper decomposition, then it may lead to problems like loss of information.

Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.

The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.

The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

**Example:**

**EMPLOYEE_DEPARTMENT table:**

| EMP_ID | EMP_NAME | EMP_AGE | EMP_CITY | DEPT_ID | DEPT_NAME |
|---|---|---|---|---|---|
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |
| 60 | Jack | 40 | Noida | 678 | Testing |

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

Q6. Explain 2 phase locking protocol

Ans) Two-phase Locking is further classified into three types :

**Strict two-phase locking protocol :**

The transaction can release the shared lock after the lock point.

The transaction can not release any exclusive lock until the transaction commits.

In strict two-phase locking protocol, if one transaction rollback then the other transaction should also have to roll back. The transactions are dependent on each other. This is called Cascading schedule.

**Rigorous two-phase locking protocol :**

The transaction cannot release either of the locks, i.e., neither shared lock nor exclusive lock.

Serailizability is guaranteed in a Rigorous two-phase locking protocol.

Deadlock is not guaranteed in rigorous two-phase locking protocol.

**Conservative two-phase locking protocol :**

The transaction must lock all the data items it requires in the transaction before the transaction begins.

If any of the data items are not available for locking before execution of the lock, then no data items are locked.

The read-and-write data items need to know before the transaction begins. This is not possible normally.

Conservative two-phase locking protocol is deadlock-free.

Conservative two-phase locking protocol does not ensure a strict schedule.

Q7. Demonstrate 1NF and 3NF with example

Ans) **First Normal Form(1NF)**

A relation will be 1NF if it contains an atomic value.

It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.

First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

**EMPLOYEE table:**

| EMP_ID | EMP_NAME | EMP_PHONE | EMP_STATE |
|---|---|---|---|
| 14 | John | 7272826385, 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389, 8589830302 | Punjab |

The decomposition of the EMPLOYEE table into 1NF has been shown below:

| EMP_ID | EMP_NAME | EMP_PHONE | EMP_STATE |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

**Third Normal Form(3NF)**

A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.

3NF is used to reduce the data duplication. It is also used to achieve the data integrity.

If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

X is a super key.

Y is a prime attribute, i.e., each element of Y is part of some candidate key.

**Example:**

**EMPLOYEE_DETAIL table:**

| EMP_ID | EMP_NAME | EMP_ZIP | EMP_STATE | EMP_CITY |
|---|---|---|---|---|
| 222 | Harry | 201010 | UP | Noida |
| 333 | Stephan | 02228 | US | Boston |
| 444 | Lan | 60007 | US | Chicago |
| 555 | Katharine | 06389 | UK | Norwich |
| 666 | John | 462007 | MP | Bhopal |

**Super key in the table above:**

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Q8. What is Deadlock and explain Deadlock prevention strategies

Ans) 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.

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.

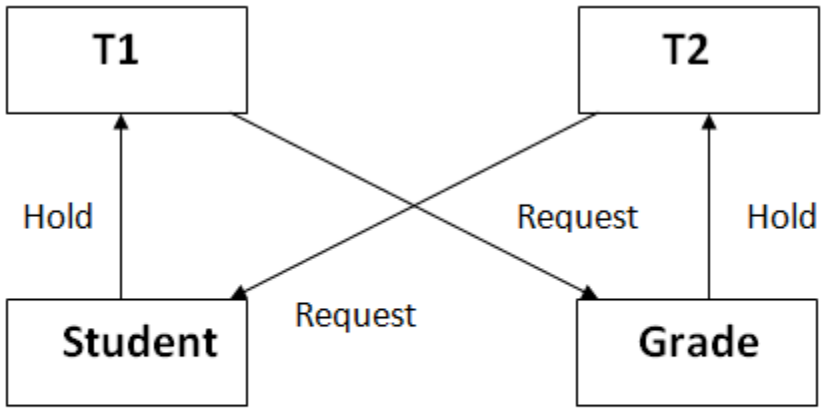



**Figure:** Deadlock in DBMS

**Deadlock Prevention Strategies**

Avoiding one or more of the above-stated Coffman conditions can lead to the prevention of a deadlock. Deadlock prevention in larger databases is a much more feasible situation rather than handling it.

The DBMS is made to efficiently analyze every database transaction, whether they can cause a deadlock or not, if any of the transactions can lead to a deadlock, then that transaction is never executed.

Wait-Die Scheme: When a transaction requests a resource that is already locked by some other transaction, then the DBMS checks the timestamp of both the transactions and makes the older transaction wait until that resource is available for execution.

Wound-wait Scheme: When an older transaction demands a resource that is already locked by a younger transaction (a transaction that is initiated later), the younger transaction is forced to kill/stop its processing and release the locked resource for the older transaction's own execution. The younger transaction is now restarted with a one-minute delay, but the timestamp remains the same. If a younger transaction requests a resource held by an older one, the younger transaction is made to wait until the older one releases the resource.Q9. What is deadlock and deadlock recovery

Ans) 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.

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.

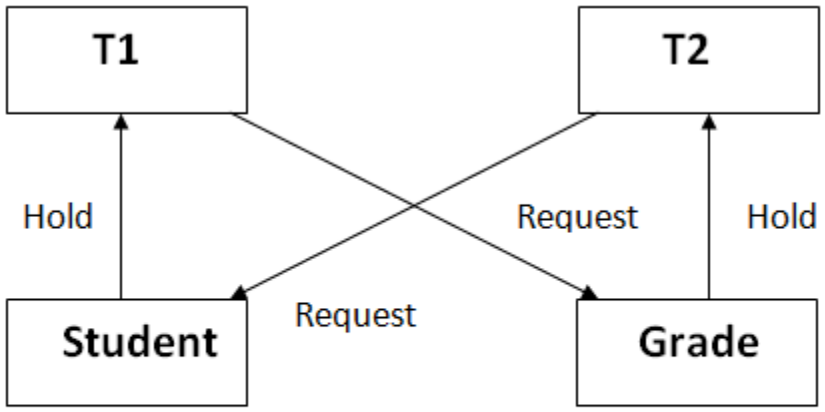



**Figure:** Deadlock in DBMS

**Recovery from Deadlock**

When a detection algorithm detects that there is a deadlock, the system must be able to recover from the deadlock.

The most common solution to recover from the dead-lock situation is to roll back one or more transactions and break the deadlock.

Here, three actions are needed to be taken :

**(a)Selection of a Victim :**

Given a set of deadlocked transactions, one must determine which transaction has to be rolled back to break the deadlock. Then, that transaction should be rolled back which occurs at minimum cost. Many other factors also play a role in deciding which transactions need to be rolled back.

**(b)Rollback :**

Once it is decided that a particular transaction must be rolled back, then determine how far this transaction should be rolled back.

The simplest solution is 'Total Rollback'. That is, we abort the transaction and then restart it.

Sometimes, it is possible to do 'partial rollback', which retains the consistency of the database. 'Partial Rollback' requires the system to maintain additional information about the state of all the running transactions.

**(c)Starvation:**

In a system where the selection of a transaction is based primarily on cost factors, then it is possible to pick the same transaction to be rolled back, known as 'victim'. As a result, the transaction never completes its designated task, thus there is a 'starvation'. So, here one should decide that a selected transaction be rolled back for a finite number of times only. Then 'starvation' does not occur.