POST GRADUATE DIPLOMA IN COMPUTER APPLICATIONS

CSP-43
Database Management System

BLOCK
3

TRANSACTION PROCESSING & CONCURRENCY CONTROL

UNIT-1 TRANSACTION PROCESSING
UNIT-2 CONCURRENCY CONTROL
UNIT-3 DATABASE RECOVERY SYSTEM
EXPERT COMMITTEE

Dr. P.K Behera(Chairman)
Reader in Computer Science
Utkal University
Bhubaneswar, Odisha

Dr. J.R Mohanty(Member)
Professor and HOD
KIIT University
Bhubaneswar, Odisha

Sri Pabitrananda Pattnaik(Member)
Scientist-E, NIC
Bhubaneswar, Odisha

Sri Malaya Kumar Das (Member)
Scientist-E, NIC
Bhubaneswar, Odisha

Dr. Bhagirathi Nayak (Member)
Professor and Head (IT & System)
Sri Sri University
Bhubaneswar, Odisha

Dr. Manoranjan Pradhan(Member)
Professor and Head (IT & System)
G.I.T.A
Bhubaneswar, Odisha

Sri Chandrakant Mallick(Convener)
Consultant (Academic)
School of Computer and Information Science
Odisha State Open University
Sambalpur, Odisha

PG DIPLOMA IN COMPUTER APPLICATIONS

Course Writers
Mr. Aseem Kumar Patel
Academic Consultant
Odisha State Open University, Sambalpur, Odisha

Course Editor
Mr. Satya Sobhan Panigrahi
Asst. Prof, SIT, BBSR
UNIT-01 TRANSACTION PROCESSING

Learning Objective

After going through this unit, you should be able to:

- describe the term Transaction;
- define the term transaction and concurrent transactions;
- discuss about concurrency control mechanism;
- describe the principles of locking and Serializability, and
- Describe concepts of deadlock & its prevention.

Structure

1.1 Introduction
1.2 Transaction Procession
   1.2.1 What is a Transaction?
   1.2.2 Operations of Transaction
1.3 States of Transaction
1.4 Properties of a Transaction (ACID)
1.5 Schedules in DMBS
   1.5.1 Types of Schedules
1.6 Testing of Serializability
1.7 Conflict Serializability
1.8 View Serializability
1.9 Let us Sum Up
1.10 Check your Progress
1.11 Multiple Choice Questions
1.12 Multiple Choice Questions Key
1.13 References
1.1 Introduction

One of the main advantages of storing data in a database is to allow sharing of it among multiple users. Several users access the database or perform operations at the same time. What if a user try to access a data item that is being used /modified by another user? So unit try to give you a detail idea on how concurrent transactions are executed under the control of DBMS. However, in order to explain the concurrent transactions, first we must describe the term transaction.

Concurrent execution of user programs is essential for better performance of DBMS, as concurrent running of several user programs keeps utilizing CPU time efficiently. DBMS is only concerned about what data is being read /written from/ into the database. This unit discusses the issues of concurrent transactions in more detail.

1.2 Transaction Procession

In a single-user application, the user can modify data without concern for other users modifying the same data at the same time. However, in a multi-user application, the statements within multiple simultaneous transactions can update the same data. Transactions executing at the same time need to produce meaningful and consistent results.

Therefore, a multi-user application must provide the following:

**Data concurrency**: which ensures that users can access data at the same time.

**Data consistency**: which ensures that each user sees a consistent view of the data, including visible changes made by the user's own transactions and committed transactions of other users.

1.2.1 What is a Transaction?

A transaction is a set of changes that must all be made together. It is a program unit whose execution may or may not change the contents of a database. Transaction is executed as a single unit. If the database was in consistent state before a transaction, then after execution of the transaction also, the database must be in a consistent state.

For example, some of the transactions at a bank may be withdrawal or deposit of money; transfer of money from A’s account to B’s account etc. A transaction would involve manipulation of one or more data values in a database. Thus, it may require reading and writing of database value.

**Example:**
Suppose a customer goes to the ATM and instructs it to transfer Rs. 1000 from its account (Account ‘A’) to his father’s account (Account ‘B’). This simple transaction requires two steps:

- Subtracting the money i.e. Rs. 1000 from his account balance (from Account A).
- Adding the money i.e. Rs. 1000 to his father’s account (from Account B)

Thus, it may require reading and writing of database value. For example, the withdrawal transactions can be written in pseudo code as:

```plaintext
TRANSACTION (A, B, transfer_amount)
Begin transaction
  IF A AND B exist then
    READ A’s Acct. balance
    IF A’s balance > transfer_amount THEN
      A’s balance = A’s balance – transfer_amount
      READ B’s Acct. balance
      B’s balance = B’s balance + transfer_amount
      COMMIT
    ELSE DISPLAY (“Insufficient Balance in Account A”)
    ROLLBACK
  ELSE DISPLAY (“ACCOUNT A OR B DOES NOT EXIST”)
End_transaction
```

So, here the set of subtraction and addition operation is called a single transaction. Likewise withdraw of money from bank account or any single or set of operation perform in the database for a particular task is called transaction. If a transaction involving only data retrieval without any data update is called read-only transaction.

Here in the above pseudo code two keyword i.e. COMMIT and ROLLBACK has been used. Commit makes sure that all the changes made by transactions are made permanent. ROLLBACK terminates the transactions and rejects any change made by the transaction.

### 1.2.2 Operations of Transaction

Basically there are six operation takes place in a transaction that is

- Begin transaction
- Read operation
- Write operation
- End transaction
- Rollback and commit
**Begin Transaction**: it is a sign that indicates the start of a transaction execution.

**Read operation**: it is used to read the value from the database and stores it in a buffer in main memory.

**Write Operation**: it is used to write the value back to the database from the buffer.

**End transaction**: it is the symbol that indicates end of a transaction.

**Commit**: it makes sure that all the changes made by transactions are made permanent.

**Rollback**: it’s a Sign to specify that the transaction has been unsuccessful and terminates the transactions and rejects any change made by the transaction. A committed transaction cannot be rolled back.

**Example**:

TRANSACTION (A, B, transfer_amount)

Begin transaction

IF A AND B exist then

READ A’s Acct. balance

IF A’s balance > transfer_amount THEN

A’s balance = A’s balance – transfer_amount

READ B’s Acct. balance

B’s balance = B’s balance + transfer_amount

COMMIT

ELSE DISPLAY (“Insufficient Balance in Account A”)

ROLLBACK

ELSE DISPLAY (“ACCOUNT A OR B DOES NOT EXIST”)

End_transaction

---

**1.3 States of Transaction**

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

- Active
- Partially committed
- Failed
- Aborted
- Committed

**Active**: In this state, the transaction is being executed. This is the initial state of every transaction.

**Partially Committed**: When a transaction executes its final operation, it is said to be in a partially committed state.
**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.

**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 or Kill the transaction.

**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.

![Transaction State Diagram](image)

**(transaction states)**

We say that a transaction has committed only if it has entered the committed state. Similarly, we say that a transaction has aborted only if it has entered the aborted state. A transaction is said to have terminated if has either committed or aborted.

A transaction starts in the active state. When it finishes its final statement, it enters the partially committed state. At this point, the transaction has completed its execution, but it is still possible that it may have to be aborted, since the actual output may still be temporarily hiding in main memory and thus a hardware failure may preclude its successful completion.

The database system then writes out enough information to disk that, even in the event of a failure, the updates performed by the transaction can be recreated when the system restarts after the failure. When the last of this information is written out, the transaction enters the committed state.

Transactions have certain desirable properties. Let us look into those properties of a transaction.
1.4 Properties of a Transaction (ACID)

In order to guarantee the integrity of the DBMS, the integrity of the transaction's ACID properties must be maintained. A transaction has four basic properties which is also called ACID property.

ACID (atomicity, consistency, isolation, durability) is a set of properties that guarantee that database transactions are processed reliably.

These are:

- Atomicity
- Consistency
- Isolation
- Durability

**Atomicity**

It defines a transaction to be a single unit of processing. In other words either a transaction will be done completely or not at all. Atomicity requires that database modifications must follow an all or nothing rule. Each transaction is said to be atomic if when one part of the transaction fails, the entire transaction fails and database state is left unchanged.

**Consistency**

This property ensures that a complete transaction execution takes a database from one consistent state to another consistent state. If a transaction fails even then the database should come back to a consistent state.

![Figure 1: A Transaction execution](image)

**Isolation**

The isolation property states that the updates of a transaction should not be visible till they are committed. Isolation refers to the requirement that other operations cannot access or see data that has been modified during a transaction that has not yet completed. Each transaction must remain unaware of other
concurrently executing transactions, except that one transaction may be forced to wait for the completion of another transaction that has modified data that the waiting transaction requires.

**For example**, if another transaction that is a withdrawal transaction which withdraws an amount of Rs. 5000 from X account is in progress, whether fails or commits, should not affect the outcome of this transaction. Only the state that has been read by the transaction last should determine the outcome of this transaction.

**Durability**
Durability is the DBMS's guarantee that once the user has been notified of a transaction's success, the transaction will not be lost even if the computer crashes.

This property necessitates that once a transaction has committed, the changes made by it be never lost because of subsequent failure. Many DBMSs implement durability by writing transactions into a transaction log that can be reprocessed to recreate the system state right before any later failure.

### 1.5 Schedules in DMBS

As we know a transaction is a collection of or set of or sequence of instructions that perform a logical work. When more than one transactions are executing parallel the order of execution of various instructions is called as a schedule.

A schedule is important in a database transaction because when more than one transactions execute in parallel, they may alter the result of the transaction means if one transaction is updating the values which is in use by the other transaction, then the order of these two transactions will change the result of second transaction. Hence a schedule is created to execute the transactions.

#### 1.5.1 Types of Schedules

1. Serial Schedules
2. Non-serial Schedules
3. Serialisable Schedules

**Serial Schedules**
The serial schedule is a type of schedule where the transactions are executed one after another, i.e., a serial schedule is one in which no transaction starts until a running transaction has ended are called serial schedules.

Suppose there are two transactions T1 and T2 having some operations. If these two has no interleaving of operations, then there will be two possible results:

Execute all the instructions of T1 which was followed by all the operations of T2.
Execute all the operations of T2 which was followed by all the operations of T1.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A);</td>
<td></td>
</tr>
<tr>
<td>A=A-100;</td>
<td></td>
</tr>
<tr>
<td>Write(A);</td>
<td></td>
</tr>
<tr>
<td>Read(B);</td>
<td></td>
</tr>
<tr>
<td>B=B+100;</td>
<td></td>
</tr>
<tr>
<td>Write(B);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(A);</td>
</tr>
<tr>
<td></td>
<td>A=A+150;</td>
</tr>
<tr>
<td></td>
<td>Write(A);</td>
</tr>
</tbody>
</table>

**Schedule-01**
Here, in the above two schedule (1 and 2) are serial schedule because to the transaction T1 and T2 are performed in serial order so they can execute in any manner i.e. T1-->T2 or T2-->T1.

**Another Example:**
Suppose there are two transactions T1 and T2 where one transaction (T1) is updating the marks of a student in one subject i.e. A; for the meantime another transaction (T2) is calculating the total marks of the same student. If the second transaction (T2) is executed after first transaction (T1) is complete, then both the transaction will be correct. But what if second transaction (T2) runs first? It will get wrong result.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadMark(A);</td>
<td></td>
</tr>
<tr>
<td>A=A+20;</td>
<td></td>
</tr>
<tr>
<td>Write(A);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadMark(A,B,C);</td>
</tr>
<tr>
<td></td>
<td>Total Mark =A+B+C;</td>
</tr>
<tr>
<td></td>
<td>Write(Total Mark);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadMark(A);</td>
<td></td>
</tr>
<tr>
<td>A=A+20;</td>
<td></td>
</tr>
<tr>
<td>Write(A);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadMark(A,B,C);</td>
</tr>
<tr>
<td></td>
<td>Total Mark =A+B+C;</td>
</tr>
<tr>
<td></td>
<td>Write(Total Mark);</td>
</tr>
</tbody>
</table>

**Schedule-A**
Here, in the above example though both the transaction executed in Serial Schedule still in Schedule-B, we get wrong result.

**Non-serial Schedules**
A schedule is said to be non-serial if interleaving of operations is allowed means if the operations of more than one transaction are executed non-consecutively and operations overlap at time.
Example: Suppose

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A); A=A-100; Write(A); Read(B); B=B+100; Write(B);</td>
<td>Read(A); A=A+150; Write(A);</td>
</tr>
</tbody>
</table>

Schedule-01

Another Example:
Suppose there are two transactions T1 and T2 where one transaction (T1) is updating the marks of a student in one subject i.e. A; for the meantime another transaction (T2) is calculating the total marks of the same student.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadMark(A); A=A+20; Write(A); ReadMark(D); D=D+10; Write(D);</td>
<td>ReadMark(A,B,C); Total Mark =A+B+C; Write(Total Mark);</td>
</tr>
</tbody>
</table>

Schedule-02

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadMark(A); A=A+20; ReadMark(D); D=D+10; Write(D);</td>
<td>ReadMark(A,B,C); Total Mark =A+B+C; Write(Total Mark);</td>
</tr>
</tbody>
</table>

Schedule-A

Here, in the above example though both the Schedule-A and B are called non-serial because each sequence interleaves operations from the two transactions.

However, some non-serial schedules give the correct expected result, such as schedule B. We would like to determine which of the non-serial schedules always give a correct result and which may give erroneous results. The concept used to characterize schedules in this manner is that of serializability of a schedule.

Serialisable Schedules

A schedule of n transactions is said to be serializable if it is equivalent to some serial schedule of the same n transactions. In other words a non-serial schedule
will be serializable if its result is equal to the result of its transactions executed serially.

A Non-serial schedules that are equivalent to one (or more) of the serial schedules is called serializable, whereas that are not equivalent to any serial schedule are called not serializable. Then the question comes how to check whether it is serializable or not?

1.6 Testing of Serializability

Testing of serializability is the answer for the above question. Serializability is a concept which is used to find which non-serial schedules are correct and will maintain the consistency of the database.

To determine serializability of a schedule the best and effective method is to construct a directed graph, which is known as precedence graph of the Schedule.

- If the directed graph or precedence graph contain a cycle then it is not serialisable or not conflict serialisable.
- If the directed graph or precedence graph not contain a cycle then it is serialisable or conflict serialisable.

Algorithm: (Testing Serializability of a Schedule S)

The steps of constructing a precedence graph are:

1. Create a node for every transaction in the schedule.
2. Find the precedence relationships in conflicting operations. Conflicting operations are (read-write) or (write-read) or (write-write) on the same data item in two different transactions. But how to find them?
   2.1 For a transaction Ti which reads an item A, find a transaction Tj that writes A later in the schedule. If such a transaction is found, draw an edge from Ti to Tj.
   2.2 For a transaction Ti which has written an item A, find a transaction Tj later in the schedule that reads A. If such a transaction is found, draw an edge from Ti to Tj.
   2.3 For a transaction Ti which has written an item A, find a transaction Tj that writes A later than Ti. If such a transaction is found, draw an edge from Ti to Tj.
3. If there is any cycle in the graph, the schedule is not serialisable, otherwise, find the equivalent serial schedule of the transaction by traversing the transaction nodes starting with the node that has no input edge.

Let us use this above algorithm to check whether a schedule is Serialisable or not.
Example:-01
Assume a schedule S as below

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X)</td>
<td>X=X-100</td>
</tr>
<tr>
<td>Write(X)</td>
<td></td>
</tr>
<tr>
<td>Read(Y)</td>
<td>Y=Y+100</td>
</tr>
<tr>
<td>Write(Y)</td>
<td>Display(X+Y)</td>
</tr>
</tbody>
</table>

As per step 1, we draw the two nodes for T1 and T2.

![Diagram](image)

Explanation:

**Read(X):** In T2, no subsequent writes to X, so no new edges

**Read(X):** X is subsequently Write by T1, so add edge T2 → T1

**Write(X):** In T2, no subsequent read/writes to X, so no new edges

**Read(Y):** Y is subsequently Write by T1, so add edge T2 → T1

**Read(Y):** In T2, no subsequent writes to Y, so no new edges

**Write(X):** In T2, no subsequent read/writes to X, so no new edges

- The transaction T2 reads data item X, which is subsequently written by T1, thus there is an edge from T2 to T1 as per clause 2.1.
- Then T2 reads data item Y, which is subsequently written by T1, thus there is an edge from T2 to T1 as per clause 2.1. As, that edge already exists, so we do not need to redraw the edge again.

Please note that there are no cycles in the graph, thus, the above schedule is **serialisable or conflict serialisable.**
**Example-02**

Assume a schedule $S$ as below

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read($X$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write($X$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write($X$)</td>
<td></td>
<td>Write($X$)</td>
</tr>
</tbody>
</table>

As per step 1, we draw the three nodes for $T_1$ and $T_2$ and $T_3$.

**Explanation:**

- **Read($X$):** $X$ is subsequently Write by $T_2$ and $T_3$, so add edge $T_1 \rightarrow T_2$ and $T_1 \rightarrow T_3$
- **Write($X$):** $X$ is subsequently Write by $T_1$ and $T_3$, so add edge $T_2 \rightarrow T_1$ and $T_2 \rightarrow T_3$
- **Write($X$):** $X$ is subsequently Write by $T_3$, so add edge $T_1 \rightarrow T_3$
- **Write($X$):** In $T_2$ or $T_3$, no subsequent read/writes to $X$, so no new edges

Please note that the graph above contains a cycle, hence it is **not serialisable** or **not conflict serialisable**.

**Example-03**

Assume a schedule $S$ as below

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read($X$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read($Z$)</td>
<td>Read($Y$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write($Z$)</td>
<td>Write($Y$)</td>
</tr>
<tr>
<td></td>
<td>Read($Y$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write($Y$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write($X$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write($X$)</td>
</tr>
<tr>
<td></td>
<td>Write($X$)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

11
As per step 1, we draw the three nodes for \( T_1 \) and \( T_2 \) and \( T_3 \).

**Explanation:**

- **Read(\( X \))**: \( X \) is subsequently written by \( T_3 \), so add edge \( T_1 \rightarrow T_3 \)
- **Read(\( Z \))**: \( Z \) is subsequently written by \( T_2 \), so add edge \( T_3 \rightarrow T_2 \)
- **Write(\( Z \))**: \( Z \) is subsequently written by \( T_2 \), so add edge \( T_3 \rightarrow T_2 \)
- **Read(\( Y \))**: In \( T_1 \) or \( T_3 \), no subsequent writes to \( Y \), so no new edges
- **Read(\( Y \))**: \( Y \) is subsequently written by \( T_2 \), so add edge \( T_1 \rightarrow T_2 \)
- **Write(\( Y \))**: In \( T_1 \) or \( T_3 \), no subsequent writes to \( Y \), so no new edges
- **Write(\( X \))**: \( X \) is subsequently written by \( T_1 \), so add edge \( T_3 \rightarrow T_1 \)
- **Write(\( Z \))**: In \( T_1 \) or \( T_3 \), no subsequent read/writes to \( Z \), so no new edges
- **Write(\( X \))**: In \( T_2 \) or \( T_3 \), no subsequent read/writes to \( X \), so no new edges

Please note that the graph above contains a cycle, hence it is not serialisable or not conflict serialisable.

**Example-04**

Assume a schedule \( S \) as below

<table>
<thead>
<tr>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read(( X ))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read(( X ))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write(( Y ))</td>
<td>Read(( X ))</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(( X ))</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read(( Y ))</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write(( Y ))</td>
<td></td>
</tr>
</tbody>
</table>
As per step 1, we draw the three nodes for T1, T2, T3 and T4

![Precedence Graph](image)

**Explanation:**

**Read(X):** X is subsequently Write by T2, so add edge T4 → T2

**Read(X):** In T1, T3 and T4 no subsequent writes to X, so no new edges

**Read(X):** X is subsequently Write by T2, so add edge T3 → T2

**Write(Y):** Y is subsequently Read by T3, so add edge T1 → T3

**Write(Y):** Y is subsequently Write by T2, so add edge T1 → T2

**Write(X):** In T1, T3 and T4 no subsequent read/writes to X, so no new edges

**Read(Y):** In T1, T2 and T4 no subsequent writes to Y, so no new edges

**Write(Y):** In T1, T3 and T4 no subsequent read/writes to Y, so no new edges

Please note that there are no cycles in the precedence graph, thus, the above schedule is **serialisable or conflict serialisable**.

As per the definition of serializable, a schedule of n transactions is said to be serializable if it is equivalent to some serial schedule of the same n transactions.

But, we checked whether a schedule is serializable or not based on precedence graph. Then question comes in our mind how come our definition becomes true? And how shall we get a serial schedule which is equivalence to non-serial schedule?

All this possible by using topological sort. Let’s understand this by applying this topological sort in above four example.

To get an equivalence serial schedule for a non-serial schedule we have to use the precedence graph. So for that we need to sort the transactions based on their incoming edges. Sort the transactions having no incoming edges or minimum edges.
**Example-01**

This is the precedence graph for schedule in the above example-01 for testing for Serializability.

There are two transaction, i.e. T1 and T2.

Transaction T1 having one (01) incoming edges whereas T2 having no incoming edges so the serial schedule will be T2->T1.

We write T2 first, as it has no incoming edges. Then T1 as it has one incoming edges.

**Example-02**

Here in this graph, there are three transaction T1, T2 and T3 having incoming edges as follows

- T1 having 1 edge
- T2 having 1 edge
- T3 having 2 edges but as there is a cycle in this graph so it is a not conflict serialisable. So we don’t need to find the equivalent serial schedule.

**Example-03**

The graph of example-03 contains a cycle, hence it is not serialisable or not conflict serialisable. So we don’t need to find the equivalent serial schedule.

**Example-04**

In this graph there is no cycle, so it is a conflict serialisable. So we can find the conflict equivalent serial schedule.

- T1 having 0 incoming edges
- T2 having 3 incoming edges
- T3 having 1 incoming edges
- T4 having 0 incoming edges
See here there are two transaction having zero incoming edges so we got two option for conflict equivalent serial schedule.
We can start with T1 or T4 both.
If we start with T1 then T4 will be the next and if we start with T4 then T1 will be the transaction.
Then, Transaction T3 having 1 incoming edges so T3 will be the next transaction and then finally T2 as it has highest numbers of incoming edges i.e. 3.
So the final Schedule will be either T1->T4->T3->T2 or T4->T1->T3->T2.
So here in this example we have two possible schedule for example-04 non-serial schedule.
Let’s write all three schedule in a place as below.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Read(X)</td>
</tr>
<tr>
<td></td>
<td>Read(X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write(Y)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Read(Y)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write(Y)</td>
</tr>
</tbody>
</table>

(Non-Serial Schedule)

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write(Y)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(Y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(Y)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Serial Schedule-01)

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Read(X)</td>
</tr>
<tr>
<td></td>
<td>Write(Y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write(Y)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Read(Y)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Read(X)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write(X)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write(Y)</td>
</tr>
</tbody>
</table>

(Serial Schedule-02)

1.7 Conflict Serializability

Serializability is a concept which is used to find which non-serial schedules are correct and will maintain the consistency of the database.

A schedule is called conflict serializability if it can be converted into a serial schedule by exchanging non-conflicting operations.
Two or more operations are said to be in conflict if:
   1. The actions belong to different transactions.
   2. The actions access the same object (read or write).
   3. At least one of the actions is a write operation.

**Example:**

**Schedule A**

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(X)</td>
</tr>
</tbody>
</table>

Schedule A is conflict Serializable because

1. Action belong to two different Transaction i.e. T1 and T2
2. Both the operation i.e. read and write operate on same data item X
3. The operation in T2 is a Write Operation

All above three condition satisfy so it is a conflict Serializable

Similarly, (Write1(X), Write2(X)) and (Write1(X), Read2(x)) pairs are also conflicting.

Whereas, (Read1(X), Write2(Y)) pair is non-conflicting because they operate on different data item. Likewise, (Write1(X), Write2(|Y)) pair is non-conflicting.

**Another Example**

In this schedule, T1(Write(X)) and T2(Read(X)) are called as conflicting operations because all the above conditions hold true for them.

**How to check this schedule is conflict serializable or not?**

We will solve by using precedence graph.

Let’s draw the precedence graph for Schedule P.

Please Note that this graph does not form a cycle, then the schedule is conflict serializable.

The corresponding serial schedule(s) can be found by Topological Sorting of this acyclic precedence graph.
As T1 have no incoming edges so T1 will come first then T2. So the serial schedule will be as follows.

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(Y)</td>
<td>Read(X)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 1.8 View Serializability

A schedule will view serializable if it is view equivalent to a serial schedule. If a schedule is conflict serializable, then it will be view serializable.

All conflict serializable schedules are view serializable but all view serializable schedules may or may not be conflict serializable.

It means if we have a non-serial schedule and we have to find the view of it using some conditions and check whether it is equal to its serial schedule or not. If it is equal then we can say it is view equivalent.

**Condition of View equivalent**

- Initial reads value must be same for all data items
- Final writers value must be same for all data items
- Write-read conflict must be same

**Example:**
Suppose we have two schedules S1 and S2 each consisting of two transactions T1 and T2. Schedules S1 and S2 are called view equivalent if the above three conditions hold true for them.

**Schedule S1 (Non-serial schedule)**

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X=X+10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(Y)</td>
<td>Y=Y+20</td>
</tr>
<tr>
<td></td>
<td>Write(Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(X)</td>
<td>X=X+10</td>
</tr>
<tr>
<td></td>
<td>Write(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(Y)</td>
<td>Y=Y*11</td>
</tr>
<tr>
<td></td>
<td>Write(Y)</td>
<td></td>
</tr>
</tbody>
</table>

**Schedule S2 (Serial schedule)**

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X=X+10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(Y)</td>
<td>Y=Y+20</td>
</tr>
<tr>
<td></td>
<td>Write(Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(X)</td>
<td>X=X+10</td>
</tr>
<tr>
<td></td>
<td>Write(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(Y)</td>
<td>Y=Y*11</td>
</tr>
<tr>
<td></td>
<td>Write(Y)</td>
<td></td>
</tr>
</tbody>
</table>
Let’s check the first condition for that we maintain a table
Here, first we check first two condition.

For Schedule S1

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T2</td>
</tr>
</tbody>
</table>

Initial value of X is first read by T1 in S1
Initial value of X is finally write by T2 in S1
Similarly
Initial value of Y is first read by T1 in S1
Initial value of Y is finally write by T2 in S1

For Schedule S2

<table>
<thead>
<tr>
<th></th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T2</td>
</tr>
</tbody>
</table>

Initial value of X is first read by T1 in S2
Initial value of X is finally write by T2 in S2
Similarly
Initial value of Y is first read by T1 in S2
Initial value of Y is finally write by T2 in S2

Here, the first value of X and Y read by the transaction is same in both the schedule. Likewise the final value of X and Y write by the transaction is same in both the schedule which is shown in the above tables.

Now our example satisfies first two conditions out of three. Now let’s check the third condition i.e. Write-read conflict must be same (W-R Conflict).

Write-read condition means, if a value is read by a transaction after written by another transaction.

In schedule S1 there is two W-R Conflict i.e. (Write(X) → Read(X)) and T1→ T2(Write(Y) → Read(Y)) which is indicated using arrow mark in schedule S1.

Similarly, In schedule S2 there is two W-R Conflict i.e. (Write(X) → Read(X)) and T1→ T2(Write(Y) → Read(Y)) which is indicated using arrow mark in schedule S2.

So for both the value X and Y the W-R conflict in S1 and s2 are same means it satisfy the third condition.

As these two schedule S1 and S2 satisfies all three condition of view equivalence.

So we can say S2 is view equivalent to S1. Means the non-serial schedule S2 is view equivalent to serial Schedule S1.
1.9 Let us Sum Up

A transaction is a set of changes that must all be made together. It is a program unit whose execution may or may not change the contents of a database. Transaction is executed as a single unit. If the database was in consistent state before a transaction, then after execution of the transaction also, the database must be in a consistent state.

A transaction in a database can be in one of the following states: Active, Partially committed, Failed, Aborted, Committed.

In order to guarantee the integrity of the DBMS, the integrity of the transaction's ACID properties must be maintained. A transaction has four basic properties which is also called ACID property. That are Atomicity, Consistency, Isolation and Durability.

As we know a transaction is a collection of or set of or sequence of instructions that perform a logical work. When more than one transactions are executing parallel the order of execution of various instructions is called as a schedule.

A schedule will view serializable if it is view equivalent to a serial schedule. If a schedule is conflict serializable, then it will be view serializable.

1.10 Check Your Progress

1. What is a transaction? What are its properties? Can a transaction update more than on data values? Can a transaction write a value without reading it? Give an example of transaction.

**Ans:**

2. What are the problems of concurrent transactions? Can these problems occur in transactions which do not read the same data values?

**Ans:**
3. What is a Commit state? Can you rollback after the transaction commits?
Ans: -

4. Consider the following two transactions, given two bank accounts having a balance A and B.
   Transaction T1: Transfer Rs. 100 from A to B
   Transaction T2: Find the multiple of A and B.
   Create a non-serialisable schedule.
Ans: -

5. Let the transactions T1, T2, T3 be defined to perform the following operations:
   T1: Add one to A
   T2: Double A
   T3: Display A on the screen and set A to one.
Suppose transactions T1, T2, T3 are allowed to execute concurrently. If A has initial value zero, how many possible correct results are there? Enumerate them.
Ans: -
1.11 Multiple Choice Questions

1. A _______ consists of a sequence of query and/or update statements.
   a) Transaction  
   b) Commit  
   c) Rollback  
   d) Flashback

2. Which of the following makes the transaction permanent in the database?
   a) View  
   b) Commit  
   c) Rollback  
   d) Flashback

3. In order to undo the work of transaction after last commit which one should be used?
   a) View  
   b) Commit  
   c) Rollback  
   d) Flashback

4. What does Rollback do?
   a) Undoes the transactions before commit  
   b) Clears all transactions
   c) Redoes the transactions before commit
   d) No action

5. In case of any shut down during transaction before commit which of the following statement is done automatically?
   a) View  
   b) Commit  
   c) Rollback  
   d) Flashback

6. _______ property will check whether all the operation of a transaction completed or none.
   a) Atomicity  
   b) Consistency  
   c) Isolation  
   d) Durability

7. A Transaction ends
   a) only when it is Committed.  
   b) only when it is Rolled-back
   c) when it is Committed or Rolled-back
   d) only when it is initialized

8. In which state, the transaction will wait for the final statement has been executed?
   a) Active  
   b) Failed  
   c) Aborted  
   d) partially committed

9. A sophisticated locking mechanism is known as 2-phase locking which includes Growing phase and _______.
   a) Shrinking Phase  
   b) Release phase  
   c) Commit phase  
   d) Acquire Phase

10. A Transaction ends
    a) only when it is Committed.  
    b) only when it is Rolled-back
    c) when it is Committed or Rolled-back
    d) only when it is initialized
1.12 Multiple Choice Questions Key
1. a) Transaction
2. b) Commit
3. c) Rollback
4. d) No action
5. c) Rollback
6. b) Consistency
7. c) when it is Committed or Rolled-back
8. d) partially committed
9. a) Shrinking Phase
10. c) when it is Committed or Rolled-back

1.13 References
UNIT-02 CONCURRENcy CONTROL

Learning Objective

After going through this unit, you should be able to:

- describe the term CONCURRENCY;
- define the term transaction and concurrent transactions
- discuss about concurrency control mechanism;
- describe the principles of locking and serialisability, and
- Describe concepts of deadlock & its prevention.

Unit Structure

2.1 Introduction
2.2 Concurrency Control
2.3 Advantage of Concurrent Execution of Transaction
2.4 Problems caused by Concurrency Control
2.5 Concurrency Control Protocol
   2.5.1 Lock Based Protocol
      2.5.1.1 Solution of Inconsistency Problem
      2.5.1.2 Problem of Starvation
      2.5.1.3 Two Phase Locking Protocol (2PL)
         2.5.1.3.1 Problems with Two Phase Locking Protocol
   2.5.2 Timestamp-Based Protocols
      2.5.2.1 Thomas Write Rule
2.6 Handling of Deadlock
   2.6.1 Deadlock Prevention
      2.6.1.1 Wait-die
      2.6.1.2 Wound-Wait
   2.6.2 Deadlock Detection and Recovery
      2.6.2.1 Deadlock Detection
      2.6.2.2 Deadlock Recovery
2.7 Let us Sum Up
2.8 Check your Progress
2.9 Multiple Choice Questions (MCQ)
2.10 Reference
2.1 Introduction

Concurrent execution of user programs is essential for better performance of DBMS, as concurrent running of several user programs keeps utilizing CPU time efficiently. DBMS is only concerned about what data is being read/written from/into the database. This unit discusses the issues of concurrent transactions in more detail.

2.2 Concurrency Control

Concurrent execution of user programs is essential for better performance of DBMS, as concurrent running of several user programs keeps utilizing CPU time efficiently, since disk accesses are frequent and are relatively slow in case of DBMS. Also, a user’s program may carry out many operations on the data returned from DB, but DBMS is only concerned about what data is being read/written from/into the database.

Concurrency is simultaneous occurrence. We could say something is concurrent when two processes happen at the same time. “Concurrent execution of transaction” means multiple transactions execute/run concurrently in RDBMS with each transaction doing its atomic unit of work for the operations encapsulated in the particular transaction.

Concurrency control in database management systems (DBMS) ensures that database transactions are performed concurrently without the concurrency violating the data integrity of a database. Executed transactions should follow the ACID rules.

2.3 Advantage of Concurrent Execution of Transaction

**Improved throughput:** Number of transactions that can be executed in a given amount of time is called as throughput. When we execute multiple transactions simultaneously that increases the transaction throughput.

**Resource utilization:** It is about using various system resources like disks and CPUs concurrently. We all know that most of the times these resources are not utilized properly. When a transaction is using CPU we may permit other transaction to use disk I/O to fetch the data from disks to main memory. This way the utilization of resources can be increased.

**Reduced waiting time:** Concurrent execution of transactions would reduce the waiting time of other transactions. Consider a situation where all transactions are executed serially. Transactions may be of any size. When executing serially, one long transaction is executing and a very small transaction may be on the queue for its turn. This increases the waiting time of small transaction. Hence,
if we execute transactions simultaneously, the waiting time would be much reduced when compared to the serial execution.

**Average response time of transaction increased**
The average time consumed by a transaction to complete since its start is called as the average response time. In concurrent execution, as multiple transactions are executing simultaneously by sharing the system resources, the waiting time is reduced which in turn increases the average response time.

### 2.4 Problems caused by Concurrency Control

If concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur.

The DBMS must ensure that two or more transactions do not get into each other's way, i.e., transaction of one user doesn’t affect the transaction of other or even the transactions issued by the same user should not get into the way of each other. Concurrency related problem may occur in databases only if two transactions are contending for the same data item and at least one of the concurrent transactions wishes to update a data value in the database. In case, the concurrent transactions only read same data item and no updates are performed on these values, then it does NOT cause any concurrency related problem.

There are four types of problem that might occur with concurrent execution of transaction i.e.

1. **Lost Update Problem (Write-Write Conflict)**
2. **The Temporary Update (Dirty Reads) problem (Write-Read Conflict)**
3. **Unrepeatable reads (Read-Write Conflict)**
4. **The incorrect summary problem or Inconsistent Analysis**

#### 1. Lost Update problem

The lost update problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect. That is, interleaved use of the same data item would cause some problems when an update operation from one transaction overwrites another update from a second transaction.

Suppose we have two transactions T1 and T2 run concurrently and they happen to be interleaved in the following way (assume the initial value of X as 50000):
The above interleaved operation will lead to an incorrect value for data item X, because at time step 2, T2 reads in the original value of X which is before T1 changes it in the database, and hence the updated value resulting from T1 is lost.

After the execution of both the transactions the value X is 7000 while the semantically correct value should be 5000-2500+2000=4500. The problem occurred as the update made by T1 has been overwritten by T2. The root cause of the problem was the fact that both the transactions had read the value of X as 5000. Thus one of the two updates has been lost and we say that a lost update has occurred.

There is one more way in which the lost updates can arise. Consider the following part of some transactions:

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>Value of X 2000</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update(X)</td>
<td></td>
<td>3000</td>
<td>T1 update the data value of X to 3000</td>
</tr>
<tr>
<td>Update(X)</td>
<td></td>
<td>4000</td>
<td>T1 update the data value of X to 3000</td>
</tr>
<tr>
<td>Rollback</td>
<td>2000</td>
<td></td>
<td>T1 rollback its operations and restore the original value of X to 2000</td>
</tr>
</tbody>
</table>

Here T1 & T2 updates the same item X. Thereafter T1 decides to undo its action and rolls back causing the value of X to go back to the original value that was 2000. In this case also the update performed by T2 had got lost and a lost update is said to have occurred.
2. The Temporary Update (or Dirty Read) Problem

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.

Occurs when one transaction can see intermediate results of another transaction before it has committed.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>Value of X 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>T1</td>
</tr>
<tr>
<td>Read(X)</td>
<td></td>
<td>2000</td>
</tr>
<tr>
<td>X=X+100</td>
<td></td>
<td>2000+100=2100</td>
</tr>
<tr>
<td>Write(X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read(X)</td>
<td></td>
<td>2000</td>
</tr>
<tr>
<td>Rollback</td>
<td></td>
<td>2000</td>
</tr>
</tbody>
</table>

T2 reads a value which has been updated by T1. This update has not been committed and T1 aborts. Here T2 reads a value that has been updated by transaction T9 that has been aborted. Thus T2 has read a value that would never exist in the database and hence the problem. Such a type of problem is called Dirty Read or temporary Update problem.

3. Unrepeatable reads

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>Value of X 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>T1</td>
</tr>
<tr>
<td>Read(X)</td>
<td></td>
<td>2000</td>
</tr>
<tr>
<td>Read(X)</td>
<td></td>
<td>2000</td>
</tr>
<tr>
<td>X=X+100</td>
<td></td>
<td>2000+100=2100</td>
</tr>
<tr>
<td>Write(X) committed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read(X)</td>
<td></td>
<td>2100</td>
</tr>
</tbody>
</table>

Suppose T1 reads X twice during its execution. If it did not update X itself it could be very disturbing to see a different value of X in its next read. But this could occur if, between the two read operations, another transaction modifies X. Thus, the inconsistent values are read and results of the transaction may be in error.

4. The incorrect summary problem

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

Occurs when transaction reads several values but second transaction updates some of them during execution of first.
Suppose that a transaction T1 is calculating the total value of A, B and C; meanwhile, transaction T2 is executing.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A)</td>
<td>Read(A)</td>
<td>100</td>
<td>50</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>S=S+A</td>
<td>A=A-10</td>
<td>100</td>
<td>50</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>Read(B)</td>
<td>Write(A)</td>
<td>90</td>
<td>50</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>S=S+B</td>
<td>Read(C)</td>
<td>90</td>
<td>50</td>
<td>25</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>Write(C)</td>
<td>90</td>
<td>50</td>
<td>35</td>
<td>150</td>
</tr>
<tr>
<td>Read(C)</td>
<td></td>
<td>90</td>
<td>50</td>
<td>35</td>
<td>150</td>
</tr>
<tr>
<td>S=S+C</td>
<td></td>
<td>90</td>
<td>50</td>
<td>35</td>
<td>185</td>
</tr>
</tbody>
</table>

The result of T1 is 185 whereas the original sum must be 175 because initially the value of A=100, B=50 and C=25. So the sum must be 100+50+25=175.

But during the sum operation of T1, T2 perform two operation on data item A and C and alter the value of A and C. Such a problem is called incorrect summery problem. This problem can avoided by preventing T1 from reading A, B and C until after T2 completed updates.

Thus, we can conclude that the prime reason of problems of concurrent transactions is that a transaction reads an inconsistent state of the database that has been created by other transaction.

Well one of the commonest techniques used for this purpose is to restrict access to data items that are being read or written by one transaction and is being written by another transaction. This technique is called locking.

### 2.5 Concurrency Control Protocol

Concurrency control protocols can be roughly divided into two types

1. Lock-Based Protocol
2. Timestamp Based Protocol

#### 2.5.1 Lock Based Protocol

To control concurrency related problems we use locking. A lock is basically a variable that is associated with a data item in the database. A lock can be placed by a transaction on a shared resource that it desires to use. When this is done, the data item is available for the exclusive use for that transaction, i.e., other transactions are locked out of that data item. When a transaction that has locked a data item does not desire to use it any more, it should unlock the data item so that other transactions can use it. If a transaction tries to lock a data item already
locked by some other transaction, it cannot do so and waits for the data item to be unlocked. The component of DBMS that controls and stores lock information is called the Lock Manager.

**Types of Locks**

There are two basic types of locks:

1. **Binary lock**
2. **Multiple-mode locks (Shared Lock and Exclusive Lock)**

**Binary lock:** This locking mechanism has two states for a data item: locked or unlocked (0 and 1 for simplicity).

Let us first take an example for binary locking and explain how it solves the concurrency related problems. Let us reconsider the transactions T1 and T2. T1 working on two data items A and B, in T1 value 10 will be subtracted from A and value 20 will be added with B whereas T2 finding the total value of A and B. Suppose initially Value of A=150 and B=200.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock(A)</td>
<td></td>
<td>A=140</td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A=A-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unlock A</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lock(A)</td>
<td>A=140</td>
</tr>
<tr>
<td></td>
<td>Lock(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(A)</td>
<td>B=200</td>
</tr>
<tr>
<td></td>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total=A+B</td>
<td>Total=340</td>
</tr>
<tr>
<td></td>
<td>Unlock(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lock(B)</td>
<td>B=200</td>
</tr>
<tr>
<td></td>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B=B+20</td>
<td>B=220</td>
</tr>
<tr>
<td></td>
<td>Write(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unlock(B)</td>
<td>A=140</td>
</tr>
</tbody>
</table>

Does the locking as done above solve the problem of concurrent transactions? No the same problems still remains. T2 reads the old value i.e. 200. Thus, locking should be done with some logic in order to make sure that locking results in no concurrency related problem. One such solution is that apply Lock on both data items A and B by transaction T1 so that T2 cannot read the data item B until it gets updated by T1.

Thus, the locking on A and B will obtain by T1 at the beginning of the transaction and release them at the end ensures that transactions are executed"
with no concurrency related problems. However, such a scheme limits the concurrency. Now this Schedule is become a Serial Schedule.

**Multiple-mode locks (Shared Lock and Exclusive Lock)**

It offers two locks: shared locks and exclusive locks. But why do we need these two locks? There are many transactions in the database system that never update the data values. These transactions can coexist with other transactions that update the database. In such a situation multiple reads are allowed on a data item, so multiple transactions can lock a data item in the shared or read lock. On the other hand, if a transaction is an updating transaction, that is, it updates the data items, it has to ensure that no other transaction can access (read or write) those data items that it wants to update. In this case, the transaction places an exclusive lock on the data items. Thus, a somewhat higher level of concurrency can be achieved in comparison to the binary locking scheme.

The properties of shared and exclusive locks are summarized below:

**Shared lock or Read Lock (Lock-S(A))**

- It is requested by a transaction that wants to just read the value of data item.
- A shared lock on a data item does not allow an exclusive lock to be placed but permits any number of shared locks to be placed on that item.
- This lock can be represented by Lock-S(A). Here ‘S’ indicates Shared Lock.

**Exclusive lock (Lock-X(A)) or Write Lock**

- It is requested by a transaction on a data item that it needs to update.
- No other transaction can place either a shared lock or an exclusive lock on a data item that has been locked in an exclusive mode.
- This lock can be represented by Lock-X(A). Here ‘X’ indicates Exclusive Lock.

<table>
<thead>
<tr>
<th>Lock</th>
<th>Shared</th>
<th>Exclusive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Exclusive</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Let us describe the above two modes with the help of an example. Let A and B be two accounts that are accessed by transactions T1 and T2. Transaction T1 transfers Rs.50 from account B to account A. Suppose the values of accounts A and B are initially Rs.200 and Rs.500, respectively.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock-X(B)</td>
<td>Read(B)</td>
<td>B=150</td>
</tr>
<tr>
<td>B=B-50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
31

Write(B)  Unlock(B)

Lock-S(A)  Read(A)  Unlock(A)  Lock-S(B)  Read(B)  Unlock(B)

A=150  B=500  Total=150+500  Total=650

A=150  B=500  Total=650

Lock-X(A)  Read(A)  A=A+20  Write(A)  Unlock(A)

Schedule-01

If both the transaction executed serially i.e. T1 → T2 or T2 → T1 then T2 will display 700 as a result. But if we run these two transaction concurrently as per above schedule then the result of Transaction T2 will be 650, which is an incorrect result. And the reason for this is that the transaction T1 unlock data item B too early. This leads the transaction into an inconsistent state.

2.5.1.1 Solution of Inconsistency Problem

As we discuss in the previous section 1.13 T2 shows an inconsistency state because of T1 unlock data item B too early. So we rewrite the above transaction T1 which delayed the unlocking of data item B to the end of the transaction. The Transaction T1 and T2 will looks like as below

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock-X(B)</td>
<td>Lock-S(A)</td>
</tr>
<tr>
<td>Read(B)</td>
<td>Read(A)</td>
</tr>
<tr>
<td>B=B-50</td>
<td>Lock-S(B)</td>
</tr>
<tr>
<td>Write(B)</td>
<td>Read(B)</td>
</tr>
<tr>
<td>Lock-X(A)</td>
<td>Total=A+B</td>
</tr>
<tr>
<td>Read(A)</td>
<td>Unlock(A)</td>
</tr>
<tr>
<td>A=A+20</td>
<td>Unlock(B)</td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
</tbody>
</table>

We can verify that the inconsistency problem in schedule-01 of above example is no longer with this schedule-02.

But, unfortunately, the use of delayed in locking leads to an undesirable situation.

Here in Transaction T1 of Schedule-02 hold an exclusive lock on data item B and T2 wants a shared lock on B means T2 is waiting for T1 to unlock B Similarly, T2 is holding a shared lock on A and T1 wants an exclusive lock on A, Thus T1 is waiting for T2 to unlock A.
Thus, now we are in a situation where no transaction can proceed. This situation is called **Deadlock**.

So we can say that solution to inconsistency problem may leads to deadlock problem.

### 2.5.1.2 Problem of Starvation

(The below content has been taken from the website: [www.ecomputernotes.com](http://ecomputernotes.com) and link is as follows


Starvation describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress.

This happens when shared resources are made unavailable for long periods by “greedy” threads.

When a transaction requests a lock on a data item in a particular mode, and no other transaction has a lock on the same data item in a conflicting mode, the lock can be granted. However care must be taken to avoid the following scenario.

Suppose a transaction T2 has a shared-mode lock on a data item, and another transaction T1 requests an exclusive mode lock on the data item. Clearly, T1 has to wait for T2 to release the share mode lock. Meanwhile a transaction T3 may request a shared mode lock on the same data item. The lock request is compatible with the lock granted to T2 so T3 may be granted the shared mode lock. At this point T2 may release the lock, but still T1 has to wait for T3 to finish. But again there may be a new transaction T4 that request a shared mode lock on the same data item and is granted the lock before T3 releases it. In fact, it is possible that there is a sequence of transactions that each request a shared mode lock on the data item and each transaction release the lock a short while after it is granted, but T 1 never gets the exclusive mode lock on the data item. The transaction T1 may never make progress and is said to be starved.

**Solution of Starvation Problem**

We can avoid starvation problem of transactions by granting locks as follows:

1. If a shared lock is requested, the queue of requests is empty, and the object is not currently locked in exclusive mode, the lock manager grants the lock and updates the lock table entry for the object (indicating that the object is locked in shared mode and incrementing the number of transactions holding a lock by one).
2. If an exclusive lock is requested, and no transaction currently holds a lock on the object (which also implies the queue of requests is empty), the lock manager grants the lock and updates the lock table entry.

3. Otherwise, the requested lock cannot be immediately granted and the lock request is added to the queue of lock requests for this object. The transaction requesting the lock is suspended.

Note that if T1 has a shared lock on A and T2 requests an exclusive lock, T2 request is queued. Now if T3 requests a shared lock, its request enters the queue behind that of T2, even though the requested lock is compatible with the lock held by T1. This rule ensures that T2 does not starve, that is, wait indefinitely while a stream of other transactions; acquire shared locks and thereby prevent T2 from getting the exclusive lock that is waiting for.

### 2.5.1.3 Two Phase Locking Protocol (2PL)

The two-phase locking (2PL), is a protocol for concurrency control used to ensure the serializability of a schedule of transaction. This protocol is based on locks, applicable from one transaction to a given (at different levels of granularity: attribute, tuple, relationship, database), which block access from other transactions to data until the transaction is terminated. It holds.

In simple terms a transaction follows the two phase locking protocol, if all the locking operation of a transaction comes first then the first unlock operation.

This protocol has two main Phases where locks are applied and removed:

- **Growing Phase**: locks are acquired and no locks are released.
- **Shrinking Phase**: locks are released and no locks are acquired.

2PL can be represented graphically as below

![Diagram of Two Phase Locking Protocol](image)

Initially, a transaction is in the growing phase where if a transaction T wants to read an object, it needs to obtain the S (shared) lock. If T wants to modify an object, it needs to obtain X (exclusive) lock. No conflicting locks are granted to a transaction. New locks on items can be acquired but no lock can be released till all the locks required by the transaction are obtained.
Once the transaction releases a lock, it enters into the Shrinking Phase. Here, the existing locks can be released in any order but no new lock can be acquired after a lock has been released. The locks are held only till they are required.

Example:

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock-X(B)</td>
<td>Lock-S(A)</td>
</tr>
<tr>
<td>Read(B)</td>
<td>Read(A)</td>
</tr>
<tr>
<td>B=B-50</td>
<td>Lock-S(B)</td>
</tr>
<tr>
<td>Write(B)</td>
<td>Read(B)</td>
</tr>
<tr>
<td>Lock-X(A)</td>
<td>Total=A+B</td>
</tr>
<tr>
<td>Read(A)</td>
<td>Unlock(A)</td>
</tr>
<tr>
<td>A=A+20</td>
<td>Unlock(B)</td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
</tbody>
</table>

Here, in this schedule T1 and T2 are in 2PL.

Please Note here that in transaction T1 the unlock(B) is not necessarily need to appear at the end of the transaction. We can place it just after the final lock operation i.e. Lock-X(A) instruction still remain the 2PL property.

2.5.1.3.1 Problems with Two Phase Locking Protocol

There are two types of problem that may occurs with 2PL Protocol. That are

1. Deadlock
2. Cascading Rollback

We will discuss Deadlock in the later part of the unit.

Cascading Rollback:

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock-X(B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read(B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock-S(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write(B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock-X(B)</td>
<td></td>
</tr>
<tr>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td>Write(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
</tbody>
</table>

Rollback
In the above schedule there are three transactions T1, T2, and T3 working on data item A and B.

Suppose transaction T1 fails after the Write (B) operation of T3. Then as per the ACID Rule, the transaction T1 must be rolled back, which also forces T2 and T3 to roll back because, transaction T2 and T3 read the value of data item B, which is updated by T1. As T1 fails and rolls back, it means T1 will get its old original value and cancel the updated value of B but T2 and T3 process the modified value of BH and result into an inconsistent state of the database. So to obtain the consistent state of the database, transaction T2 and T3 need to be rolled back and start again. This problem is called **Cascading Rollback**.

To avoid such a situation, we use strict 2PL. The first phase of Strict-two phase Lock is the same as 2PL. After gaining all the locks in the growing phase, the transaction continues to execute normally. But in Strict-2PL, it does not release an exclusive lock after using it. It holds the exclusive locks until the commit point and releases all the locks at a time. It makes sure that if a data item is modified by one transaction, then other transactions cannot read it until the first transaction commits or abort. Strict Two-Phase Locking Protocol avoids cascaded rollbacks but it is not deadlocks free.

Most of the Real world database systems implement rigorous two phase locking protocol.

To comply with rigorous strict two-phase locking (RS2PL), the locking protocol releases both write (exclusive) and read (shared) locks applied by a transaction only after the transaction has ended. So in a rigorous two-phase locking protocol all the locks (Shared and Exclusive lock) to be held until the transaction commits.

### 2.5.2 Timestamp-Based Protocols

Timestamp is a unique identifier created by the DBMS to identify a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the transaction start time. We will refer to the timestamp of transaction T as \( \text{TS}(T) \). Concurrency control techniques based on timestamp ordering do not use locks; hence, deadlocks cannot occur.

Timestamps can be generated in two ways.

- One possibility is to use a counter that is incremented each time its value is assigned to a transaction. The transaction timestamps are numbered 1, 2, 3, ..., in this scheme. A computer counter has a finite maximum value, so the system must periodically reset the counter to zero when no transactions are executing for some short period of time.
Another way to implement timestamps is to use the current date/time value of the system clock and ensure that no two timestamp values are generated during the same tick of the clock.

The idea for this scheme is to order the transactions based on their timestamps. A schedule in which the transactions participate is then serializable, and the equivalent serial schedule has the transactions in order of their timestamp values. This is called timestamp ordering (TO).

In timestamp ordering, the schedule is equivalent to the particular serial order corresponding to the order of the transaction timestamps. The algorithm must ensure that, for each item accessed by conflicting operations in the schedule, the order in which the item is accessed does not violate the serializability order. To do this, the algorithm associates with each database item X two timestamp (TS) values:

1. Read_TS(X): The read timestamp of item X; this is the largest timestamp among all the timestamps of transactions that have successfully read item X that is, read_TS(X) = TS(T), where T is the youngest transaction that has read X successfully.
2. Write_TS(X): The write timestamp of item X; this is the largest of all the timestamps of transactions that have successfully written item X—that is, write_TS(X) = TS(T), where T is the youngest transaction that has written X successfully.

The Timestamp Ordering Protocol

Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with read_TS(X) and write_TS(X) to ensure that the timestamp order of transaction execution is not violated and executed in timestamp order.

If this order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp.

The protocol operates as follows:

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a Read (X) operation:
   a) If TS(Ti) < Write_TS(X) then Ti needs to read a value of X that was already overwritten. Hence the operation is rejected and Ti is rolled back.
   b) If TS(Ti) >= Write_TS(X) then the operation is executed.
      Read_TS(X) is set to the maximum of R-timestamp(X) i.e. T(Ti).
2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:
   a) If TS(Ti) < Read_TS(X) then the operation is rejected and Ti is rolled back.
   b) If TS(Ti) < Write_TS(X) then the operation is rejected and Ti is rolled back.
   c) Otherwise (TS(Ti) >= Read_TS(X) and TS(Ti) >= Write_TS(X)) the operation is executed and Write_TS(X) is set to TS(Ti).

Example:

1. Check the following condition whenever a transaction Ti issues a **Read(X)** operation:
   a) For If TS(Ti) < Write_TS(X) then the operation is rejected and Ti is rolled back.

   Suppose we have a schedule S1 as follows:

   Suppose that T1 has a time stamp 4.00 PM and T1 has a time stamp is 4.02 PM. Here when T2 perform Write(X) operation the Write_TS(X) become 4.02 PM which is equal to TS(T2).
   Then T1 wants to perform Read(X) operation.

   So TS(T1) is 4.00 PM which is less than Write_TS(X) i.e. 4.02 PM (4.00 < 4.02)
   So this operation is rejected and rolled back.

   b) For If TS(Ti) >= Write_TS(X) then the read operation is executed.

   Read_TS(X) is set to the maximum of R-timestamp(X) i.e. T(Ti).

   Suppose that T2 has a time stamp 4.02 PM and T2 has a time stamp is 4.05 PM. Here when T2 perform Write(X) operation the Write_TS(X) become 4.02 PM which is equal to TS(T2). Then T3 wants to perform Read(X) operation.

   So TS(T3) is 4.05 PM which is greater than Write_TS(X) i.e. 4.02 PM (4.05 > 4.02) So this read operation is executed and Read_TS(X) become 4.05 PM.

Example:

2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:
   a) If TS(Ti) < Read_TS(X) then the operation is rejected and Ti is rolled back.

   Suppose that T1 has a time stamp 4.00 PM and T2 has a time stamp is 4.00 PM. Here when T2 perform Read(X) operation the Read_TS(X) become 4.02 PM which is equal to TS(T2). Then T1 wants to perform Write(X) operation.
So TS(T1) is 4.00 PM which is less than Read_TS(X) i.e. 4.02 PM \((4.00<4.02)\) So this Write operation is rejected and Roll backed.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X)</td>
<td></td>
</tr>
<tr>
<td>Write(X)</td>
<td></td>
</tr>
</tbody>
</table>

b) For If TS(Ti) < Write_TS(X) then the operation is rejected and Ti is rolled back.

Suppose that T1 has a time stamp 4.00 PM and T2 has a time stamp is 4.02 PM. Here when T2 perform Write(X) operation the Write_TS(X) become updated to 4.02 PM which is equal to TS(T2). Then T1 wants to perform Write(X) operation.

So TS(T1) is 4.00 PM which is less than Write_TS(X) i.e. 4.02 PM \((4.00<4.02)\) So this Write operation is again rejected and Roll backed.

c) For Otherwise \((TS(Ti)\geq Read_TS(X) \text{ and } TS(Ti) \geq Write_TS(X))\) the operation is executed and Write_TS(X) is set to TS(Ti).

Suppose that T1 has a time stamp 4.00 PM and T2 has a time stamp is 4.02 PM. Here when T1 perform Read(X) operation the Read_TS(X) become 4.00 PM and when Write(X) operation is performed the Write_TS(X) become updated to 4.00 PM which is equal to TS(T1). Then T2 having TS(T2)=4.02 PM wants to perform Write(X) operation.

So, As per the condition TS(T2) must be greater than Read_TS(X) and Write_TS(X) to execute.

Here, \((4.02\geq4.00)\) and \((4.02>4.00)\) so So this Write operation is executed and Write_TS(X) become TS(T2) i.e. 4.02 PM.

Another Example:

Suppose that transaction T5 starts at 2.00 PM and Transaction T6 starts at 2.02PM and Value of X=500 and Y=200. The schedule is as follows:

Initially TS(T1)=2.00PM, TS(T2)=2.02PM,
- Read_TS(X)=0 and Write_TS(X)=0
- Read_TS(Y)=0 and Write_TS(Y)=0

<table>
<thead>
<tr>
<th>Steps</th>
<th>T1</th>
<th>T2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read(X)</td>
<td></td>
<td>TS(T1)&gt;Write_TS(X)/2.00&gt;0 So read operation performed and Read_TS(X) become 2.00 PM.</td>
</tr>
</tbody>
</table>
This schedule shows that both transaction T1 and T2 perform concurrently under the timestamp schedule without giving inconsistent results. This protocol also guarantee of no deadlock, since no transaction ever waits.

### 2.5.2.1 Thomas Write Rule

Thomas Write rule is the modified version of basic timestamp ordering, in which rules for write operations are slightly different from basic timestamp.

Thomas write rule basically emphasize to increase the concurrency and reduce the rollbacks of timestamp based protocols.

Let’s consider the following schedule to understand the Thomas Write rule.

Suppose we have a schedule as follows:

If we apply the Time Stamp Ordering protocol on this schedule then as per the rule 2(b), the transaction T1 will be rejected and rolled back.

Since T1 starts before T2, so TS(T1)<TS(T2). The Read(X) operation of T1 succeeds. Again Read(X) and Write(X) is also allowed but when T1 attempts...
its write(X) operation, we find that TS(T1)< Write_TS(X). Thus the Write(X) by T1 is rejected and transaction T1 must be rolled back.

But As per the Thomas write Rule, if we face such situation then instead of T1 rejected and rolled back Write operation will be simply ignored. Which is called ignore obsolete write rule. The rules for read operations remain unchanged.

The difference between the Timestamp ordering protocol and Thomas write rule lies in the second rule.

The timestamp ordering protocol requires that Ti is rolled back if Ti issues a write operation on a data item and TS(Ti)< Write_TS(data). However, in Thomas write rule where TS(Ti)< Write_TS(data) we ignore the obsolete write. This modification to the timestamp ordering protocol is called Thomas Write Rule.

2.6 Handling of Deadlock

We talk about deadlock in section 1.12.1.1 Solution of Inconsistency Problem. In order the understand deadlock in more details let us consider the following example.

As we can see the Transaction T1 is waiting for exclusive lock on data item B which is currently locked by Transaction T2 and similarly Transaction T2 is waiting for T1 to release exclusive lock on data item A so that T2 can perform exclusive lock on A. Such a cycle of transactions waiting for locks to be released is called Deadlock.

So deadlock can be defined as “A system is said to be in deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set”.

In such situation no transaction can make any progress in such situation.

There are two basic methods for dealing with the deadlock problem.

- Deadlock Prevention
- Deadlock detection

Deadlock prevention: These are the steps to ensure that the system will never enter into deadlock state.

Deadlock Detection: In Deadlock detection we allows the system to enter into the deadlock state and then we implements deadlock detection and deadlock recovery scheme to recover from it.
2.6.1 Deadlock Prevention

We can set priority to every transaction to prevent deadlock and ensure that lower priority transaction will not wait for a higher priority transaction. A timestamp can be set to every transaction to assign priority when they starts up. The transaction having lower timestamp, is considered as higher priority and higher timestamp indicates lower priority.

There are be two policies for deadlock prevention i.e. wait-die and Wound-wait.

2.6.1.1 Wait-die

As per wait-die policy when a higher priority transaction Ti requests a data item which is currently held by Tj, Ti is allowed to wait only if it has a timestamp value smaller then that of Tj. Otherwise Ti is rolled back.

If TS(T1) < TS(T2), then T1 is allowed to wait, otherwise abort T1 and restart it later with the same timestamp.

In wait-die, older transaction is allowed to wait on younger transaction, whereas a younger transaction requesting lock on record R1 held by an older transaction is aborted and restarted.

Example:

Suppose we have three transaction T1, T2 and T3 having timestamps 1, 2 and 3 respectively. If T1 requests a data item which is held by transaction T2 then T1 will wait. If T3 requests the same data item held by T2 then T3 will be rolled back.

T1 will wait T3 will rollback
T1 T2 T3
Priority 1 Priority 2 Priority 3

In wait-die, transactions wait only on younger transactions.

2.6.1.2 Wound-Wait

When transaction Ti requests a data item currently held by Tj, Ti is allowed to wait only if it has a timestamp larger than that of Tj. Otherwise Tj is rolled back.

If TS (T1) < TS (T2), then abort T1 and restart it later with same timestamp; otherwise T1 is allowed to wait.

In wound-wait approach a younger transaction is allowed to wait on an older one, whereas an older transaction requesting lock on record R1 held by an younger transaction preempts the younger transaction by aborting it.
Example:
Suppose in the previous example the transaction T1 request for a data item which is held by T2 then the data item will be preempted from T2 and t2 will be rolled back.

If T3 requests the same data item held by T2, then T3 will wait.

<table>
<thead>
<tr>
<th>T1 will get the access</th>
<th>T2 will be rollback</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T2</td>
</tr>
<tr>
<td>Priority 1</td>
<td>Priority 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data item with T2</th>
<th>wait for T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>T3</td>
</tr>
<tr>
<td>Priority 2</td>
<td>Priority 3</td>
</tr>
</tbody>
</table>

In wait-die policy, lower priority transactions never waits for higher priority transactions where as in wound-wait policy, higher priority transaction never wait for lower priority transactions. In both case no cycle form and deadlock never occurs.

2.6.2 Deadlock Detection and Recovery

Deadlock detection and recovery is more practical approach than the deadlock prevention techniques. This first checks whether deadlock state actually exist in the system before taking any actions.

An algorithm (wait for graph) that checks the system periodically to determine whether a deadlock has occurred. If yes, then the system start trying to recover from deadlock. A simple way to detect a state of deadlock is for the system to construct and maintain a “wait-for” graph.

2.6.2.1 Deadlock Detection

A system that allows concurrent operation of multiple processes and locking of resources and which does not provide mechanisms to avoid or prevent deadlock must support a mechanism to detect deadlocks and an algorithm for recovering from them. One such deadlock detection algorithm makes use of a wait-for graph.

A simple way to detect a state of deadlock is for the system to construct and maintain a “wait-for” graph.

A wait-for graph is a directed graph used for deadlock detection in database systems.

A wait-for graph is used to track which other transactions a transaction is currently blocking on. In a wait-for graph, transactions are represented as nodes, and an edge from transaction Ti to Tj (Ti → Tj) implies Tj is holding a data
item that Ti needs and thus Ti is waiting for Tj to release its lock on that data item. When Tj releases the lock on the data items that Ti was waiting for, the directed edge is dropped from the graph. Finally if there will be a cycle in wait for graph then we can conclude that there is a state of deadlock.

**Example:**

Let’s create a situation to understand the deadlock detection. Suppose there are four transaction T1, T2, T3 and T4. Where

- T1 is waiting for transaction T2 and T3
- T3 is waiting for T2 and T2 is waiting for T4.

This wait for graph has no cycle, so there is no deadlock state.

![Wait for Graph](image)

**Example:**

Suppose Now T4 requesting a data item held by T3. Then an edge from T4 \(\rightarrow\) T3 will be added as shown in figure. Which results an cycle in wait for graph?

The cycle in the wait for graph will be as follows

T2 \(\rightarrow\) T4 \(\rightarrow\) T3 \(\rightarrow\) T2.

This means that transactions T2, T3 and T4 are all deadlocked.

### 2.6.2.2 Deadlock Recovery

Once the deadlock detection algorithm concludes that deadlock exists, the next step is to recover from deadlock and the most common and simplest solution is to rollback one or more transactions to break the cycle and recover from deadlock.
Then the question comes, which transaction(s) need to be rollback so that we can overcome from deadlock? This choosing of transaction to abort is called Victim Selection.

**Victim Selection**

In the above wait for graph example of Deadlock Detection the wait for graph forms a cycle and transaction T2, T3, T4 are deadlocked. So to remove deadlock we must roll back one transaction out of these three.

So we should roll back that transaction that will gain the minimum cost.

This decision is made based on the following measures

- The transaction that have the fewest locks
- The transaction that has done the least work
- The transaction that is farthest from completion

Once we decide which transaction must rolled back, we have to find out how far it should be rolled back. The solution is total rolled back- abort it and restart it again but it will be more efficient if we rollback the transaction to the extent that it breaks the deadlock.

There are some cases where the same transaction will be picked as victim, so this transaction will never complete its task. This may leads to starvation. So we must ensure that transaction can be picked as a victim only a limited number of times.
2.7 Let us Sum Up

Concurrency is simultaneous occurrence. We could say something is concurrent when two processes happen at the same time.

Concurrency control in database management systems (DBMS) ensures that database transactions are performed concurrently without the concurrency violating the data integrity of a database. Executed transactions should follow the ACID rules.

Concurrent execution of transaction may improve throughput, Resource utilization, reduced waiting time and Average response time of transaction increased.

There are four types of problem with concurrent execution of transaction i.e. Lost Update Problem (Write-Write Conflict), The Temporary Update (Dirty Reads) problem (Write-Read Conflict), Unrepeatable reads (Read-Write Conflict), the incorrect summary problem or Inconsistent Analysis.

Concurrency control protocols can be roughly divided into two types i.e. Lock-Based Protocol, Timestamp Based Protocol.

Once the deadlock detection algorithm concludes that deadlock exists, the next step is to recover from deadlock and the most common and simplest solution is to rollback one or more transactions to break the cycle and recover from deadlock.

2.8 Check your Progress

1. What is concurrent Execution? How it is different from Serial Execution?
   Ans:..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................

2. What are the Advantages of concurrent Execution of Transaction?
   Ans:..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................
   ..........................................................................................................................

3. Describe different problems caused by concurrency Control.
   Ans: -

4. What do you mean by Lock? What are its types?
   Ans: -

5. What is 2PL Protocol? Explain with example.
   Ans: -

6. What is Starvation? What is the problem of Starvation?
   Ans: -
7. **What is Time-Stamp Based Protocol? How it is implemented?**
   **Ans:**

8. **What is Thomas Write Rule?**
   **Ans:**

9. **What is Deadlock? How deadlock can be handled in DBMS?**
   **Ans:**

10. **What is Deadlock Detection and Recovery?**
    **Ans:**
2.9 Multiple Choice Questions (MCQ)

1. Deadlock prevention scheme that requires each transaction to locks all its data items before it begins
   - a) Initialization
   - b) **Execution**
   - c) Evaluation
   - d) Processing

2. Deadlock prevention scheme named wound-wait is a
   - a) Non-linear preemptive technique
   - b) Linear preemptive technique
   - c) **Preemptive technique**
   - d) Non-preemptive technique

3. A transaction that is inserting a new tuple into database is given an
   - a) Shared lock
   - b) Mutual lock
   - c) **Exclusive lock**
   - d) NO lock

4. Concurrency control is a challenging task for transactions that have
   - a) Application accesses
   - b) I/O activities
   - c) **User interactions**
   - d) Application interactions

5. Cascading rollbacks can be avoided by
   - a) Strict two-phase locking protocol
   - b) Rigorous two-phase locking protocol
   - c) Deadlock two-phase locking protocol
   - d) Lock-point two-phase locking protocol

6. Two modes of locking a data item, are termed as 'shared' and
   - a) Composite
   - b) Compatible
   - c) **Exclusive**
   - d) Linear

7. Two-phase locking does not ensure freedom from
   - a) Obtain locks
   - b) Release locks
   - c) New locks
   - d) **Deadlocks**

8. A time stamp-ordering scheme ensures
   - a) **Serializability**
   - b) Cascading
   - c) Atomicity
   - d) Consistency

9. Which of the following locks the item from change but not from read?
   - a) Implicit lock
   - b) Explicit lock
   - c) **Exclusive lock**
   - d) **Shared lock**

10. Dirty read, nonrepeatable, and phantom reads are not possible in this instance.
    - a) Read committed
    - b) Read uncommitted
    - c) Repeatable read
    - d) **Serializable**

11. The situation where no transaction can proceed with normal execution is known as ________
    - a) Road block
    - b) **Deadlock**
    - c) Execution halt
    - d) Abortion

12. If a transaction Ti may never make progress, then the transaction is said to be ________
    - a) Deadlocked
    - b) **Starved**
    - c) Committed
    - d) Rolled back
2.10 References:

UNIT-03 DATABASE RECOVERY SYSTEM

Learning Objective

After going through this unit, we should be able to:

- describe the terms Data Storage and Recovery
- describe different Recovery Techniques;
- find different causes for Database Failure

Structure

3.1 Introduction
3.2 Data Storage
3.3 What are the Causes for Database Failure?
3.4 Data Recovery
3.5 Log based recovery
   3.5.1 Deferred Database Modification
   3.5.2 Immediate Database Modification
3.6 Checkpoints
   3.6.1 Maintenance of UNDO and REDO List
3.7 Shadow Paging
   3.7.1 Advantage of Shadow Paging
3.8 Let us Sum Up
3.9 Check Your Progress
3.10 Multiple Choice Questions
3.11 References
3.1 Introduction

In the earlier units of this block, we learnt about the concepts of transactions and Concurrency control. In this unit we will introduce an important issues relating to database management systems i.e. Database Recovery.

A computer system suffers from different types of failures. A DBMS controls very critical data of an organization and therefore must be reliable. However, the reliability of the database system is linked to the reliability of the computer system on which it runs. In this unit we will discuss recovery of the data contained in a database system following failure of various types and present the different approaches to database recovery. The types of failures that the computer system is likely to be subjected to include failures of components or subsystems, software failures, power outages, accidents, unexpected situations and natural or man-made disasters. Database recovery techniques are methods of making the database consistent till the last possible consistent state. The aim of recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information at an economically justifiable cost.

In order to understand the concept of database recovery it is important to understand different storage devices at which data is stored.

Let us discuss all these in more detail in this unit.

3.2 Data Storage

Based on increasing degree of reliability storage media can be categorized into four types

- Main Memory
- Magnetic disk
- Magnetic Tape and
- Optical disk

Main memory, is used by RAM chips to temporarily store the data the computer needs to calculate and track the programs it runs. As it is a volatile storage it can’t store data permanently, so it does not survive system crashes. Magnetic disk storage is used in mechanical hard drives to keep track of permanent information. Compare to main memory, disk are more reliable and cheaper but in terms of speed disk are slower.

Magnetic tape are non-volatile storage which is more reliable then main memory and magnetic disk. Magnetic tape is used for backups whereas, magnetic disk are used as secondary storage. But magnetic tape are fairly expensive and slower that provides sequential access. It is Idle for random access and Fast in data accessing.
Optical storage, the typical optical disc, stores information in a circular disc. Optical disc storage is non-volatile. The deformities may be permanent (read only media), formed once (write once media) or reversible (recordable or read/write media). Optical disk is more reliable than other media in terms of speed, storage, cost etc.

**Stable Storage**

Stable storage is a classification of computer data storage technology that guarantees atomicity for any given write operation and allows software to be written that is robust against some hardware and power failures. In other words it is a technique where information is never lost.

This is said to be third form of memory structure but it is same as nonvolatile memory. In this case, copies of same nonvolatile memories are stored at different places. This is because, in case of any crash and data loss, data can be recovered from other copies. This is even helpful if there one of non-volatile memory is lost due to fire or flood. It can be recovered from other network location. RAID or Redundant Array of Independent Disks, is a technology to connect multiple secondary storage devices and use them as a stable storage media.

### 3.3 What are the Causes for Database Failure?

(This content has been taken from: [http://ecomputernotes.com](http://ecomputernotes.com). Link is given below: [http://ecomputernotes.com/database-system/adv-database/causes-for-database-failure](http://ecomputernotes.com/database-system/adv-database/causes-for-database-failure))

There are many different types of failure that can affect database processing, each of which has to be dealt with in a different manner. Some failures affect main memory only, while others involve non-volatile (secondary) storage. Among the causes of failure are:

- System Crashes
- User Error
- Carelessness
- Sabotage (intentional corruption of data)
- Statement Failure
- Application software errors
- Network Failure
- Media Failure
- Natural Physical Disasters

In case of system crash, the systems hang up and need to be rebooted. These failures occur due to hardware malfunction or a bug in the database software or the operating system itself. It causes the loss of the content of volatile storage and brings transaction processing to a halt. The content of nonvolatile storage
does not affected with this type of failure. The assumption that hardware errors and bugs bring the system to a halt, but do not corrupt the nonvolatile storage contents is known as the Fail-Stop Assumption.

An example of a user error is a user inadvertently deleting a row or dropping a table. Carelessness is the destruction of data by operators or users because they were not concentrating on the task at hand. Sabotage is the intentional corruption or destruction of data, hardware or software facilities.

A statement failure can be defined as the inability of the database to execute an SQL statement. While running a user program, a transaction might have multiple statements and one of the statements might fail due to various reasons. Typical examples are selecting from a table that does not exist, or trying to do an insert and having the statement fail due to lack of space. Such statement failures normally generate error codes and messages by the application software or the operating system. Recovery from such failures is automatic. Upon detection, the database usually will roll back the statement, returning control to the user or user program. The user can simply re-execute the statement after correcting the problem. Application software errors include logical errors in the program that is accessing the database, which causes one or more transactions to fail.

Network failures can occur while using a client-server configuration or a distributed database system where multiple database servers are connected by communication networks. Network failures such as communication software failures or aborted asynchronous connections will interrupt the normal operations of the database system.

Media failures are the most dangerous failures. Not only there is a potential to lose data if proper backup procedures are not followed, but it usually takes more time to recover than with other kinds of failures. In addition, the DBA's experience is very important factor in determining the kind of media recovery procedure to use to bring the database up quickly, with little or no data loss. A typical example of a media failure is a disk controller failure or disk head crash, which causes all, databases residing on that disk or disks to be lost. Every DBA needs to plan appropriate backup procedures to protect against media failures.

Natural and physical disasters are the damage caused to data, hardware and software due to natural disasters like fires, floods, earthquakes, power failures, etc.

### 3.4 Data Recovery

Recovery from transaction failures usually means that the database is restored to the most recent consistent state just before the time of failure.
To do this, the system must keep information about the changes that were applied to data items by the various transactions. This information is typically kept in the system log. A typical strategy for recovery may be summarized informally as follows:

1) If there is extensive damage to a wide portion of the database due to catastrophic failure, such as a disk crash, the recovery method restores a past copy of the database that was backed up to archival storage (typically tape or optical disk) and reconstructs a more current state by reapplying or redoing the operations of committed transactions from the backed up log, up to the time of failure.

2) When the database is not physically damaged but has become inconsistent due to no catastrophic failures of types such as transaction or system error, local errors, concurrency control enforcement, the strategy is to reverse any changes that caused the inconsistency by undoing some operations. It may also be necessary to redo some operations in order to restore a consistent state of the database. In this case we do not need a complete archival copy of the database. Rather, the entries kept in the on-line system log are consulted during recovery.

There are two ways to perform the recovery

- Log based Recovery
- Shadow Paging

### 3.5 Log based recovery

In log based recovery a log is a sequence of log records, recording all the update activities in the database. In a stable storage, logs for each transaction are maintained. Any operation which is performed on the database is recorded on the log. Prior to performing any modification to database, an update log record is created to reflect that modification.

Different types of log record are as follows:

<Ti start>: This indicates the start of the transaction

<Ti, Xj, V1, V2>: This shows the write operation

Here, Ti is the transaction

Xj is the data item

V1 is the original value

V2 is the updated value

**Example:**

<T1, A, 100,200>

Transaction T1 perform a successful write operation on data item A whose old value was 100 and after write operation A has be updated to 200.
<Ti commit>: It contains information about when a transaction Ti commits.
<Ti abort>: It contains information about when a transaction Ti aborts.

When a transaction performs a write A, it is essential that the log record for that write be created before the database is modified. Once a log record exists, we can undo a transaction.

There are two main techniques for Log based recovery:

- Deferred Database Modification
- Immediate Database Modification

### 3.5.1 Deferred Database Modification

The deferred database modification techniques do not physically update the database on disk until after a transaction reaches its commit point; then the updates are recorded in the database. Before reaching commit, all transaction updates are recorded in the local transaction workspace (or buffers). During commit, the updates are first recorded persistently in the log and then written to the database. If a transaction fails before reaching its commit point, it will not have changed the database in any way, so UNDO is not needed. It may be necessary to REDO the effect of the operations of a committed transaction from the log, because their effect may not yet have been recorded in the database. Hence, deferred update is also known as the NO-UNDO/REDO algorithm.

**Example:**

Consider the following transaction T1 and Transaction T2

Let T1 transfers the amount of 200 from Account X to Account Y and T2 transaction withdraws Rs. 500 from account Z. Suppose these transaction are executed serially. Suppose the initial value of Account X=1000, Y=2000 and Z=3000 respectively.

Using deferred update, the process is:

<table>
<thead>
<tr>
<th>Transaction Instruction(T1)</th>
<th>Database</th>
<th>Log file</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 Read(X) X=X-200</td>
<td>Buffer</td>
<td>&lt;T1,start&gt;</td>
</tr>
<tr>
<td>T1 Write(X)</td>
<td></td>
<td>&lt;T1,X,1000,800&gt;</td>
</tr>
<tr>
<td>T1 Read(Y) Y=Y+200</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The value of X and Y is changed in the database only after the record <T1, commit> has placed in the log file.

Recovery Procedure

In differed database modification the recovery procedure is based on the redo operation.

Once failure occurred the system check the log file and starts it recovery and identify the transaction that need to be redo. There are two operation that can be performed i.e. redo and re-execute.

If the log file contain both <Ti, start> and <Ti, commit>, this transaction may or may not be stored to the disk/database physically so use redo operation to get the modified values from the log record.

There will be three case possible of transaction failure as follows

<table>
<thead>
<tr>
<th>Case-I</th>
<th>Case-II</th>
<th>Case-III</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;T1,start&gt; &lt;T1,X,1000,800&gt; &lt;T1,Y,2000,2200&gt;</td>
<td>&lt;T1,start&gt; &lt;T1,X,1000,800&gt; &lt;T1,Y,2000,2200&gt; &lt;T1, commit&gt; &lt;T2,start&gt; &lt;T2,Z,3000,2500&gt;</td>
<td>&lt;T1,start&gt; &lt;T1,X,1000,800&gt; &lt;T1,Y,2000,2200&gt; &lt;T1, commit&gt; &lt;T2,start&gt; &lt;T2,Z,3000,2500&gt;</td>
</tr>
</tbody>
</table>

Case-I

In case-I system crash just after the write(Y). In such situation, as there is no <T1, commit> in the log file, it means no update has been made in the database. So database value will remain same i.e. X=1000 and Y=2000. And log file records for Transaction T1 will be deleted.
Case-II

In case-II system crash just after the write(Z). In such situation, as there is only <T1, commit> is present in the log file but no commit record for transaction T2, it means Transaction T1 has updated its database value in disk but T2 have-not. So When the system comes back then redo operation will be perform on transaction T1 and log value for T1 will be updated in database but as there is no <T2, commit> for T2 so there is no need of redo(T2) operation and T2 will be re-executed and log record for T2 will be deleted.

Recovery: As <T1, commit> is present in log file, redo(T1) will performed and will results X=800 and Y=2200.

Case-III

In case-III system crash just after the <T2, commit>. In such situation, as both <T1, commit> and <T2, commit> is present in the log file, redo(T1) and redo(T2) will be performed. After these operations are executed the value of X, Y and Z will be 800, 2200 and 4500 respectively.

Recovery: As <T1, commit> and <T2, commit> is present in log file then both redo(T1) and redo(T2) will performed and will results X=800, Y=2200 and Z=4500 respectively.

3.5.2 Immediate Database Modification

In the immediate update techniques, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are typically recorded in the log on disk by force writing before they are applied to the database, making recovery still possible.

If a transaction fails after recording some changes in the database but before reaching its commit point, the effect of its operations on the database must be undone; that is, the transaction must be rolled back.

In the general case of immediate update, both undo and redo may be required during recovery. This technique, known as the UNDO/REDO algorithm, requires both operations, and is used most often in practice.

Example:

Consider the following transaction T1 and Transaction T2

Let T1 transfers the amount of 200 from Account X to Account Y and T2 transaction withdraws Rs. 500 from account Z. Suppose these transaction are executed serially. Suppose the initial value of Account X=1000, Y=2000 and Z=3000 respectively.
Using immediate database Modification, the process is:

<table>
<thead>
<tr>
<th>Transaction Instruction(T1)</th>
<th>Database</th>
<th>Log file</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 Read(X) X=X-200</td>
<td>(Buffer)</td>
<td>&lt;T1,start&gt;</td>
</tr>
<tr>
<td>T1 Write(X)</td>
<td></td>
<td>&lt;T1,X,1000,800&gt;</td>
</tr>
<tr>
<td>T1 Read(Y) Y=Y+200</td>
<td></td>
<td>X=800</td>
</tr>
<tr>
<td>T1 Write(Y)</td>
<td></td>
<td>&lt;T1,Y,2000,2200&gt;</td>
</tr>
<tr>
<td>T1</td>
<td></td>
<td>Y=2200</td>
</tr>
<tr>
<td>T1</td>
<td></td>
<td>&lt;T1, commit&gt;</td>
</tr>
<tr>
<td>T2 Read(Z) X=X-500</td>
<td></td>
<td>&lt;T2,start&gt;</td>
</tr>
<tr>
<td>T2 Write(Z)</td>
<td></td>
<td>Z=2500</td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td>&lt;T2, commit&gt;</td>
</tr>
</tbody>
</table>

The value of X and Y is changed in the database only after the record <T1, X, 1000, 800> and <T1, Y, 2000, 2200> has placed in the log file but before commit record.

**Recovery Procedure**

In immediate database Modification the recovery procedure is based on the undo and redo operation.

**Undo(Ti):** it will restore the value of all data items updated by Transaction Ti to the old value present in the log file.

**Redo(Ti):** Set the value of all data items updated by transaction Ti to the new values.
There will be three cases possible of transaction failure as follows:

<table>
<thead>
<tr>
<th>Case-I</th>
<th>Case-II</th>
<th>Case-III</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;T1,start&gt;</td>
<td>&lt;T1,start&gt;</td>
<td>&lt;T1,start&gt;</td>
</tr>
<tr>
<td>&lt;T1,X,1000,800&gt;</td>
<td>&lt;T1,X,1000,800&gt;</td>
<td>&lt;T1,X,1000,800&gt;</td>
</tr>
<tr>
<td>&lt;T1,Y,2000,2200&gt;</td>
<td>&lt;T1,Y,2000,2200&gt;</td>
<td>&lt;T1,Y,2000,2200&gt;</td>
</tr>
<tr>
<td>&lt;T1,commit&gt;</td>
<td>&lt;T1,commit&gt;</td>
<td>&lt;T1,commit&gt;</td>
</tr>
<tr>
<td>&lt;T2,start&gt;</td>
<td>&lt;T2,start&gt;</td>
<td>&lt;T2,start&gt;</td>
</tr>
<tr>
<td>&lt;T2,Z,3000,2500&gt;</td>
<td>&lt;T2,Z,3000,2500&gt;</td>
<td>&lt;T2,commit&gt;</td>
</tr>
</tbody>
</table>

**Case-I**

In case-I system crash just after the write(Y). In such situation, as there is no <T1, commit> in the log file but <t1, start> is there. So undo(T1) operation will be performed and old value are copied from log file to database.

Recovery: The old value of X =1000 and Y=2000 are taken from the log record and T1 and T2 must be re-executed.

**Case-II**

In case-II system crash just after the write(Z). In such situation, as there is only <T1, commit> is present in the log file but no commit record for transaction T2. When the system comes back then redo operation will be perform on transaction T1 and log value for T1 will be updated in database but as there is no <T2, commit> for T2 so undo(T2) will be performed.

**Recovery: redo(T1) and undo(T2)**

So the results will be X=800, Y=2200 and Z=3000 and transaction will be re-executed.

**Case-III**

Crash occurs just after <T2, commit>

When system comes back both T1 and T2 need to be redo. As both start and commit record are present in log file for T1 and T2.

**Recovery: redo(T1) and redo(T2)**

X=800, Y=2200 and Z=4500

**3.6 Checkpoints**

When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery
system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

A checkpoint record is written into the log periodically at that point when the system writes out to the database on disk all DBMS buffers that have been modified. As a consequence of this, all transactions that have their [commit, T] entries in the log before a checkpoint entry do not need to have their WRITE operations redone in case of a system crash, since all their updates will be recorded in the database on disk during check pointing.

The recovery manager of a DBMS must decide at what intervals to take a checkpoint. The interval may be measured in time say, every m minutes or in the number t of committed transactions since the last checkpoint, where the values of m or t are system parameters.

Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Taking a checkpoint consists of the following actions:

1) Suspend execution of transactions temporarily.
2) Force-write all main memory buffers that have been modified to disk.
3) Write a checkpoint record to the log, and force-write the log to disk.
4) Resume executing transactions

**Example:** (This example has been taken from the book “Simplified Approach to DBMS by Parteek Bhatia and Gurvinder Singh, Page no. 375, 8th Edition”)

Let’s consider the below figure

- A system failure has occurred at time t$_f$.
- The most recent checkpoint prior to time t$_f$ was taken at time t$_c$.
- Transaction T1 completed successfully prior to time t$_c$.
Transaction T2 started prior to time $t_c$ and completed successfully after time $t_c$ and before time $t_f$

Transaction T3 also started prior to time $t_c$ but did not complete by time $t_f$

Transaction T4 started after time $t_c$ and completed successfully before time $t_f$

Finally, transaction T5 also started after time $t_c$ but did not complete by time $t_f$

When the system is restarted in case of immediate database modification, transactions of types T3 and T5 must be undone and transaction T2 and T4 must be redone. However, transactions of type T1 do not enter in the restart process at all, because their updates were forced to the database at time $t_c$ as part of the checkpoint access.

### 3.6.1 Maintenance of UNDO and REDO List

When the system restart after crash, first it goes through the subsequent procedure to identify all transactions i.e. T2 – T5.

- It starts by creating two transaction list, one UNDO list and another REDO list. All the transactions that are executing at the time if recent checkpoint will be listed in the UNDO List. And keep the REDO list empty.

<table>
<thead>
<tr>
<th>UNDO LIST</th>
<th>REDO LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>Empty</td>
</tr>
<tr>
<td>T3</td>
<td></td>
</tr>
</tbody>
</table>

- Start searching in the log, starting from the checkpoint record.
- If the log contains a BEGIN TRANSACTION log entry for Transaction Ti then add Ti to the UNDO list

<table>
<thead>
<tr>
<th>UNDO LIST</th>
<th>REDO LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>Empty</td>
</tr>
<tr>
<td>T3</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td></td>
</tr>
</tbody>
</table>

As per above diagram in 3.6 T4 and T5 begins after checkpoint $t_c$.

- If a COMMIT entry is found for Transaction Ti, then transfer Ti from UNDO list to REDO List.

<table>
<thead>
<tr>
<th>UNDO LIST</th>
<th>REDO LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>T3</td>
<td>T2</td>
</tr>
<tr>
<td>T5</td>
<td>T4</td>
</tr>
</tbody>
</table>

T2 and T4 are committed before failure ($t_f$) as the above diagram.
Once we reached to the end of log, both the lists identify. As per the above example of 3.6 UNDO list contains T3 and T5 Whereas REDO list contains T2 and T4.

### 3.7 Shadow Paging

Shadow Paging is a simple and common database recovery Technique. It is an alternative to log-based recovery techniques, which has both advantages and disadvantages. Under certain circumstances, shadow paging may require fewer disk accesses than do the log-based methods. Shadow paging provides the capability to manipulate pages in a database.

In Shadow paging technique, updated data item is stores at a different disk location. Thus there are multiple copies of a data item can be maintained on disk whereas in log based recovery updated data item in the same location, so that the new data item will overwriting the old value of the data item on disk. So only one copy of data item maintained in disk.

In this Technique, database is subdivided into different fixed-length blocks, which are referred to as pages.

Suppose there is a database which is divided into different fixed-length page of n number, numbered 1 through n. These pages do not need to be stored in any particular order on disk. However, there must be a way to find the ith page of the database for any given i. We use a page table for this purpose. The page table has n entries—one for each database page. Each entry contains a pointer to a page on disk. The first entry contains a pointer to the first page of the database, the second entry points to the second page, and so on.

The main purpose of this technique is to keep two-page tables during the life of a transaction: the current page table and the shadow page table. When the transaction starts, both page tables are alike. During the life of a transaction shadow page table is never changed but when a transaction performs a write operation, the current page table may be changed. The current page table is used for all I/O operations to locate database pages on disk.

Suppose a transaction Ti performs a write(A) operation, and that A exist in the ith page. The system executes the write operation as follows:

1. If the ith page is not already in main memory, then the system issues input(A).
2. If this is the first write Operation on the ith page by the same transaction, then the system modifies the current page table as follows:
   a. First find an unused page on disk. Generally, the database system has access to a list of unused (free) pages.
b. It deletes the page found in step 2(a) from the list of unused free page; it copies the contents of the ith page to the page found in step 2(a).

c. Then modifies the current page table so that the ith entry points to the page found in step 2(a).

3. Finally allocate the value of Aj to A in the buffer page.

Intuitively, the shadow-page approach to recovery is to store the shadow page table in nonvolatile storage, so that the state of the database prior to the execution of the transaction can be recovered in the event of a crash, or transaction abort. When the transaction commits, the system writes the current page table to non-volatile storage. The current page table then becomes the new shadow page table, and the next transaction is allowed to begin execution. It is important that the shadow page table be stored in nonvolatile storage, since it provides the only means of locating database pages. The current page table may be kept in main memory (volatile storage). We do not care whether the current page table is lost in a crash, since the system recovers by using the shadow page table.

Successful recovery requires that we find the shadow page table on disk after a crash. A simple way of finding it is to choose one fixed location in stable storage that contains the disk address of the shadow page table. When the system comes back up after a crash, it copies the shadow page table into main memory and
uses it for subsequent transaction processing. Because of our definition of the write operation, we are guaranteed that the shadow page table will point to the database pages corresponding to the state of the database prior to any transaction that was active at the time of the crash. Thus, aborts are automatic. Unlike our log-based schemes, shadow paging needs to invoke no undo operations.

To commit a transaction, we must do the following:

1. Ensure that all buffer pages in main memory that have been changed by the transaction are output to disk. (Note that these output operations will not change database pages pointed to by some entry in the shadow page table.)
2. Output the current page table to disk. Note that we must not overwrite the shadow page table, since we may need it for recovery from a crash.
3. Output the disk address of the current page table to the fixed location in stable storage containing the address of the shadow page table. This action over-writes the address of the old shadow page table. Therefore, the current page table has become the shadow page table, and the transaction is committed.

If a crash occurs prior to the completion of step 3, we revert to the state just prior to the execution of the transaction. If the crash occurs after the completion of step 3, the effects of the transaction will be preserved; no redo operations need to be invoked.

**Example:**

Suppose there are two transaction T1 and T2. Let T1 be a transaction that transfer 500 rupees from Account X to Y whereas in Transaction T2 withdraws 200 rupees from Account Z.

Suppose the initial values of X, Y & Z are 1000, 2000 and 3000 respectively.

The Operations of the Transactions are as follows:

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X, x)</td>
<td>Read(Z, z)</td>
</tr>
<tr>
<td>x=x-500</td>
<td>z=z-200</td>
</tr>
<tr>
<td>Write(X, x)</td>
<td>Write(Z, z)</td>
</tr>
<tr>
<td>Read(Y, y)</td>
<td></td>
</tr>
<tr>
<td>Y=y+500</td>
<td></td>
</tr>
<tr>
<td>Write(Y, y)</td>
<td></td>
</tr>
</tbody>
</table>

When the above transaction starts executing the shadow page table stored in the disk is replicate a copy onto the main memory and it becomes current page table as shown in the below diagram.
T1: Read (X, x)

When the read (X) operation performed, the block that containing the data X is moved from database (Hard disk) to main memory (RAM) by input (X) operation and copied the value of X to x i.e. 1000.

T1: x=x-500

When the above operation performed it will Subtract 500 from the local variable x, so now x contains 500.

Write (X, x)

1. If block contain data X is not available in main Memory, then input(X) operation is performed.
2. 
   a. Find the unused space on the disk i.e. 7th page as per the above diagram.
   b. Delete the page found in step 2 (a) from the list of free page frame.
   c. Then change the current page table so that CPT entry for X will point to page 7th instead of page 2.
3. Assign the value of 500 to data item X in RAM.

T1: Read (Y, y)

When the read (Y) operation performed, the block that containing the data Y is moved from database (Hard disk) to main memory (RAM) and copied the value of Y to y i.e. 2000.

T1: y=y+500

When the above operation performed it will Subtract 500 from the local variable x, so now x contains 2500.
Write \((Y, y)\)
The block that contains \(B\) moved in main memory and a unused space is choose. The value of \(Y\) i.e. 2500 is copied in page 11. After that 11\(^{th}\) page is discarded from unused free page list.
Finally, change the CPT so that \(B\) will points to page 11.

**T1: Commit**
1. All the pages in main memory that has been changed by Transaction T1 are updated in disk. Means \(X\)’s value 500 stored in 7\(^{th}\) page and \(Y\)’s value 2500 stored at page 11.
2. Output the CPT to disk without overwriting the SPT. Note that we must not overwrite the shadow page table, since we may need it for recovery from a crash.
Similarly Transaction T2 will execute.

**Failure Cases:**

**Case-I** (If system crash before commit of T1)
In this case Current Page Table (CPT) is also crashed as it presents in main memory. When system restarts it get the address of Shadow Paging Table (SPT) from the disk let say 20000 (address). This table point to the old value of \(X\), \(Y\) & \(Z\), it means it is same as undo operation of Log-base Recovery. In short if the system crashed before commit point, we will get the previous state of data from shadow Page.

**Case-II** (If system crash after commit of T1)
In this case, when system restarts it get the address of Shadow Paging Table (SPT) from the disk now this table point to modified value of \(X\) and \(Y\). Which is similar to redo operation in case of Log based recovery. It means no need to re-execute the transaction and it will recover the data successfully.

**3.7.1 Advantage of Shadow Paging**

Advantage of Shadow Paging techniques are as follows:

1. The overhead of log-based output is eliminated.
2. Recovery from crashes is faster.
3. It does not require the use of log, thus no undo and redo operation is needed. This makes the recovery process simpler.

**3.8 Let us Sum Up**

Data recovery is an important aspect of Database management system. In this Unit we learnt about the terms Data Storage and Recovery, why it is important for database management system, different Recovery Techniques i.e. Log based recovery, Checkpoints and Shadow Paging, its advantage. Finally we also learnt about different causes for Database Failure.
3.9 Check Your Progress

1. What is the importance of database recovery? How it can be achieved?
   Ans:----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------

2. What are different types of database failure?
   Ans:----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------

3. Comparison between deferred and immediate database modification recovery Technique.
   Ans:----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------

4. What is Log based database recovery? How it is different from Shadow Paging?
   Ans:----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------

5. What is Checkpoint? How it can be implemented in database recovery?
   Ans:----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   ----------------------------------------------------------------------------------------------------------------------------------
   **Ans:**

7. What are the Causes for Database Failure?
   **Ans:**

8. Explain Redo and Undo operation in database management system with example.
   **Ans:**

9. Explain about different storage media in database.
   **Ans:**
3.10 Multiple Choice Questions (MCQ)

1. Log records contain
   a) Old value
   b) New value
   c) Both A and B
   d) Error value

2. Error that causes loss of data of volatile storage, and halts transaction processing is known as
   a) System error
   b) Application error
   c) System crash
   d) Transaction error

3. An integral part of database that can restore database to consistent state of before failure is called
   a) Recovery scheme
   b) Backup scheme
   c) Restoring scheme
   d) Transaction scheme

4. Database is partitioned into storage units of fixed length, known to be
   a) Blocks
   b) Sectors
   c) Units
   d) Tracks

5. Log record type that describes a single database write is known as
   a) Transaction log record
   b) Write log record
   c) Read log record
   d) Update log record

6. In log-based schemes, all updates are recorded on
   a) Blocks
   b) Logs
   c) Disks
   d) Main memory

7. Number of fields in an update log record are
   a) 3
   b) 4
   c) 5
   d) 6

8. Which of the following is not a recovery technique?
   a) Deferred update
   b) Immediate update
   c) Two-phase commit
   d) Recovery management

9. Checkpoints are a part of
   a) Recovery measures
   b) Security measures
   c) Concurrency measures
   d) Authorization measures

10. Failure recovery and media recovery fall under ……..
    a) transaction recovery
    b) database recovery
    c) system recovery
    d) value recovery

11. ………. is an alternative of log based recovery.
    a) Disk recovery
    b) Shadow paging
    c) Dish shadowing
    d) Crash recovery

12. If a transaction does not modify the database until it has committed, it is said to use the ___________ technique.
    a) Deferred-modification
    b) Late-modification
    c) Immediate-modification
    d) Undo
3.11 References

1. Fundamentals of Database Systems by Elmasri Navathe
2. https://test.classle.in/book/recovery-concepts