CHAPTER 4

TRANSACTION PROCESSING FOR THE PROPOSED
COMPREHENSIVE PRDBMS

4.1 INTRODUCTION:

The concept of transaction provides a mechanism for describing the logical units of database processing. Often, a collection of several operations on the database is considered to be a single unit from the point of view of the database user. For example, a transfer of funds from a current account to a savings account is a single operation when viewed from the account-holder’s standpoint. In fact, within the database system, it comprises several operations. It is necessary that all these operations occur, or that, in case of a failure, none should occur. It would be inappropriate if the current account is debited but savings account is not credited. Collection of operations that constitute a single logical unit of work is called a transaction. A database system must ensure the proper execution of transactions despite failures —either the transaction executes all of its operations or none is executed. Further, it must manage concurrent execution of transactions in a way that avoids the introduction of inconsistency. The chapter provides the definition of a transaction and that of a probabilistic transaction which is supported by the comprehensive probabilistic RDBMS. It also enumerates ACID and PLACID properties that a transaction and a probabilistic transaction should possess respectively. Apart from this, concurrency control techniques for probabilistic relational databases are also presented.

4.2 DEFINITION OF A TRANSACTION AND ITS PROBABILISTIC COUNTERPART:

The model of the transaction can be shown as in the fig. 4.1
A transaction can be defined as a unit of consistent, reliable and secure computing. The module of Probabilistic RDBMS that keeps track of transactions and manages them may be called as transaction manager. The transaction manager should be able to perform the following two functions:

(i) Keep the database in a consistent state in the event of concurrent accesses by two or more transactions.
(ii) Take care of the recovery in case database fails.

Typically a Transaction Manager takes the services of a scheduler for scheduling or unscheduling the requests of various transactions.

The body of the transaction may be enclosed between the two directives {Begin_Transaction and End_Transaction} as shown below:

```
Begin_Transaction
---------
---------
End_Transaction.
```

The body of the transaction consists of the statements that read or write the values of some of the data items. It may be noted here that reading or writing operations on probabilistic relations can be carried out by using the commands of PSQL (Probabilistic Structured Query Language) whose syntax has already been provided[89]. If the transaction successfully terminates carrying out all of its specified operations, then it is
said to commit itself. If it fails due to any reason, then it is said to have aborted and consequently must be rolled back. Any changes made by the unsuccessful transaction to the database must be undone.

From the above discussion, it is quite clear that the ‘input set’ to the transaction is any read or write operation performed on some database item x. Let it be denoted by IS(x) where IS ∈ {read, write}. The ‘output set’ of the transaction may be denoted by OS and this set will contain either of the two conditions i.e OS ∈ {Commit, abort}. The body of the transaction shall also be performing some operations in the form of computations and let it be denoted by AO(T). Therefore, formally a transaction can be defined as a partial order over its specified operations and the output conditions. This partial order may be denoted by P{S,≤} where S consists of operations and output set of the transaction and ≤ indicates the execution order of these operations.

The probabilistic database will be said to be in a consistent state if it obeys all the integrity constraints defined over it. Prior to and after the execution of the transaction, database should remain in a consistent state. During execution of the transaction, however, the database can be in an inconsistent state temporarily.

The definition of the transaction has its roots in contract law[75]. In making a contract, two or more parties negotiate for a while and then make a deal. The deal is made binding by the joint signatures on a document or by some other act (such as a handshake or a nod). If the parties are suspicious of each other or just want to be safe, they appoint an intermediary to coordinate the commitment of the transaction.

An important aspect of this historical perspective is that the above description encompasses some of the fundamental properties of a transaction as the term is used in the database system. It also serves to indicate the difference between a transaction and a query.

A transaction is basically a unit of consistent and reliable computation. Therefore, a transaction takes a database, performs an action on it, and generates a new version of the database, causing a state transition. This is similar to what a query does, except that if the database was consistent before the execution of the transaction, one can guarantee that it will be in a consistent state at the end of its execution regardless of the fact that

(i) the transaction may have been executed concurrently with others and
In general, a transaction is considered to be made up of a sequence of read and write operations on the database together with some computation steps. In that sense, a transaction may be thought of as a program with embedded database access queries[93]. Another definition of the transaction is that it is a single execution of a program[75]. A single query can also be thought of as a program that can be posed as a transaction.

The general model of the transaction management for the comprehensive PRDBMS may be shown as in Fig. 4.2 below:

![Fig. 4.2]

4.3 ACID PROPERTIES OF TRANSACTION:

To ensure consistency and reliability of the computation, a transaction should possess the following ACID properties. The acronym ACID stands for Atomicity, Consistency, Isolation, Durability.
The term Atomicity refers to the fact that a transaction is treated as a unit of operation. All the actions of the transaction are either completed to entirety or none of them is performed. This property is sometimes referred to as ‘all-or-none’ property. Atomicity requires that if the execution of the transaction is interrupted by any type of failure, database management system will be responsible for determining what to do with the transaction upon recovery from the failure. Broadly, there can be two types of failures. The transaction may fail owing to input data errors, deadlocks or certain other factors. In such a case either the transaction aborts itself or database management system aborts it while handling deadlocks. In the second type of failure related to system crashes (may be media failures, processor failures etc), the information in volatile storage may be lost or may become inaccessible.

The consistency of the transaction is simply its correctness. In other words, a transaction is a correct program that maps one consistent database state to another. There is an interesting classification of consistency. This classification groups databases into four levels of consistency[40]. The definition is taken verbatim from the original paper. In the definition of different consistency levels, dirty data refers to data values that have been updated by a transaction prior to its commitment. Then, based upon the concept of dirty data, the four levels are defined as follows:

Degree 3: Transaction T sees degree 3 consistency if:
1. T does not overwrite dirty data of other transactions.
2. T does not commit any writes until it completes all its writes [i.e. until the end of the transaction (EOT)].
3. T does not read dirty data from other transactions.
4. Other transactions do not dirty any data read by T before T completes.

Degree 2: Transaction T sees degree 2 consistency if:
1. T does not overwrite dirty data of other transactions.
2. T does not commit any writes until it completes all its writes [i.e. until the end of the transaction (EOT)].
3. T does not read dirty data from other transactions.
Degree 1: Transaction T sees degree 1 consistency if:
1. T does not overwrite dirty data of other transactions.
2. T does not commit any writes until it completes all its writes [i.e. until the end of the transaction (EOT)].

Degree 0: Transaction T sees degree 0 consistency if:
1. T does not overwrite dirty data of other transactions.

Of course, it is true that a higher degree of consistency encompasses all the lower degrees. The point in defining multiple levels of consistency is to provide application programmers the flexibility to define transactions that operate at different levels. Consequently, while some transactions operate at Degree 3 consistency level, others may operate at lower levels and may see dirty data.

Isolation is the property of transactions which requires each transaction to see a consistent database at all times. In other words, an executing transaction cannot reveal its results to other concurrent transactions before its commitment. The reason for insisting upon isolation is to maintain interconsistency of transactions. If two concurrent transactions access a data item that is being updated by one of them, it is not possible to guarantee that the second will read the correct value. Consider the following two concurrent transactions:

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(x)</td>
<td>read(x)</td>
</tr>
<tr>
<td>X:=x+1</td>
<td>x:=x+1</td>
</tr>
<tr>
<td>Write(x)</td>
<td>write(x)</td>
</tr>
<tr>
<td>Commit</td>
<td>commit</td>
</tr>
</tbody>
</table>

The transactions can be executed in a serial order i.e. T1 followed by T2 or T2 followed by T1. Since the transactions can execute concurrently, the following sequence is also possible:

T1: read(x)
T1: x=x+1
T2: read(x)
T1: write(x)
T2: x:=x+1
T2: write(x)
T1: commit
T2: commit

Initially let the value of x be 50. If the above execution sequence is followed, transaction T2 reads the value of x as 50. This is incorrect since T2 reads x while its value is being changed from 50 to 51. Furthermore, the value of x is 51 at the end of execution of T1 and T2 since T2’s write will overwrite T1’s write.

Durability refers to the property of the transactions which ensures that once a transaction commits, its results are permanent and cannot be erased from the database. Therefore, the database management system ensures that the results of a transaction will survive subsequent system failures.

4.4 PLACID PROPERTIES OF PROBABILISTIC TRANSACTIONS:

The transaction to be executed by the comprehensive probabilistic RDBMS must possess the following PLACID properties[93] which are the extension of ACID properties that a transaction satisfies in a deterministic database system. In fact, the acronym ‘PLACID’ properties stands for the following:

(i) Probability preserving
(ii) Lossless
(iii) Atomicity
(iv) Consistency
(v) Isolation
(vi) Durability

Since it is assumed that the probability stamp, ps is a joint probability distribution of all its attribute values appearing for a particular tuple, the execution of the transaction should preserve the joint probability of these attributes.

This probability stamp, in fact, is a joint probability distribution of all the attribute values taken together and the probability of the individual attribute can be derived by appropriately marginalising the said distribution.
Mathematically, let $X$ and $Y$ be two discrete random variables. Probability distribution for their simultaneous occurrence can be represented by a function with values $f(x,y)$ for any pair of values $(x,y)$ within the range of the discrete random variables $X$ and $Y$. This function is referred to as joint probability distribution of $X$ and $Y$. It has the following properties:

1. $f(x,y) \geq 0$ for all $(x,y)$
2. $\sum \sum f(x,y)=1$
3. $P(X=x, Y=y) = f(x,y)$

For any region $A$ in the $xy$ plane, $P[(x,y) \in A] = \sum \sum f(x,y)$

If the joint probability distribution $f(x,y)$ of the discrete random variables $X$ and $Y$ are given, the probability distribution $g(x)$ of $X$ alone is obtained by summing $f(x,y)$ over the values of $Y$. Similarly, the probability distribution $h(y)$ of $Y$ alone is obtained by summing $f(x,y)$ over the values of $X$. $g(x)$ and $h(y)$ are said to be the marginal distributions of $X$ and $Y$ respectively.

Therefore, the marginal distribution of $x$ alone and $Y$ alone are given by

$$g(x)= \sum f(x,y)$$
and
$$h(y)= \sum f(x,y)$$

for the discrete case.

The preceding definitions of joint and marginal distributions can be generalized in the case of $n$ random variables in the similar manner.

The second property which is *lossless* property indicates that if the probabilistic relation is divided into different subrelations (each consisting of a subset of attributes of the original probabilistic relation obtained by appropriately marginalising the distribution), then the original joint probability distribution should be accurately obtainable from the marginal distribution of these subrelations. This property may be beneficial if a probabilistic relation is to be decomposed during execution of a transaction. A decomposition shall be called as lossless-join if original joint probability distribution can be accurately obtained from the marginal distributions. We should not separate two dependent attributes into smaller relations representing two marginal distributions and navigational links should be maintained in the form of foreign keys.
The Atomicity property refers to the fact that transaction is treated as a unit of operation that is to be completed in its entirety or not executed at all. It may also be called as 'all-or-none' property. Consistency refers to the correctness of the transaction i.e. a transaction maps the database from one consistent state to the other consistent state. Isolation is the property that requires each transaction to see a consistent database at all the times i.e. before a transaction commits, the results of the transaction are not available to the other concurrent transactions. Durability refers to the permanence of the changes made by the transaction i.e. once a transaction has committed, its results are permanent and cannot be erased from the database.

4.5 NEED FOR CONCURRENCY CONTROL:

If multiple transactions are submitted to the database management system for processing then DBMS interleaves the actions of different transactions to improve performance in terms of increased throughput or improved response time for short transactions. It may be noted here that not all the interleavings are allowed. Ensuring transaction isolation while permitting such concurrent execution is difficult, but is necessary for performance reasons. Firstly, while one transaction is waiting for a page to be read from the disk, CPU can process another transaction. It is because of the fact that I/O activity can be carried out in parallel with CPU activity. Overlapping CPU and I/O activities reduces the amount of times disks and processors are idle and increases the system throughput which may be defined as the average number of transactions completed per unit of time. The second reason is that interleaved execution of a short transaction with a long transaction usually allows the short transaction to complete quickly. In serial execution, a short transaction could get stuck behind a long transaction leading to unpredictable delays in response time or the average time taken to complete a transaction. The schedule of \( n \) transactions may be defined as a list of actions from the set of \( n \) transactions and the order in which two actions of a transaction \( T \) appearing in a schedule must be the same as the order in which they appear in \( T \). In other words, a schedule represents an actual or potential execution sequence.

A serializable schedule over a set of \( N \) transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial
schedule over \( N \), i.e., the database instance that results from executing the given schedule is identical to the database instance that results from executing the transactions in some serial order.

There could be three anomalies if the transactions are executed in an interleaved fashion as follows:

(i) Reading uncommitted data (WR Conflict)
(ii) Unrepeatable reads (RW Conflict)
(iii) Overwriting uncommitted data (WW Conflict)

In WR conflict, a transaction \( T_1 \) could read a database object that was modified by another transaction \( T_2 \), which has not yet committed. Such a read is called as a dirty read.

In RW conflict, a transaction \( T_1 \) could change the value of a database object \( A \) that has been read by a transaction \( T_2 \), while \( T_2 \) is still in progress. If \( T_2 \) tries to read the value of \( A \) again, it will get a different result, even though it has not modified \( A \) in the meantime. The situation could not arise in a serial execution of two transactions, it is called as unrepeatable read.

In WW conflict, transaction \( T_1 \) could overwrite the value of a database object \( A \), which has already been modified by a transaction \( T_2 \), while \( T_2 \) is still in progress. Even if \( T_1 \) does not read the value of \( A \) written by \( T_2 \), a potential problem exists.

Owing to the anomalies mentioned above, there is a need for concurrency control. The database management system must be able to ensure that only serializable schedules are allowed. A DBMS provides locking protocols to achieve this.

### 4.6 PROBABILISTIC CONCURRENCY CONTROL ALGORITHM BASED ON LOCKING TECHNIQUE:

This section discusses the locking technique that can be used for controlling concurrency among various simultaneous transactions submitted to the comprehensive PRDBMS for execution. For concurrency control using locks, the data items in the probabilistic database need to be locked for reading or writing operations within a transaction. The locks are required to ensure correct order of execution of the concurrent transactions. The application program hands over the transaction to the transaction manager which subsequently hands it over to the scheduler. The scheduler acts as a lock manager and is
responsible for granting or revoking of locks on some particular database items. Another important aspect related to the locking technique is the size of data items that are to be locked and it is sometimes referred to as the granularity of the locking. With reference to probabilistic databases, the granularity of locking is an important question that should be answered correctly. To maintain the database in a consistent state, the entities whose probability stamps add up to one, thereby indicating that existence of that entity is certain, should be treated as a single lock unit. On the other hand, the entities whose existence is uncertain i.e. their probability stamps do not add up to one can be clubbed together with other database items to form a single lock unit. After the transaction completes its operation, the locks can be released.

Given below are the notations that shall be helpful in formally stating the proposed algorithm subsequently.

Let operation denotes one of the database operations or one of the transaction directives. Let data-item denotes the data item that is to be fetched from the database. Let Transaction_id denotes the unique identifier for a particular transaction. The granularity of the data item to be locked is referred to as lock-item. The application program generates a message in the form {Begin_transaction, operations, end_transaction} which is submitted to the Transaction manager. Transaction manager identifies it to be a transaction and stamps it with a unique transaction identifier denoted by Transaction_id. It then passes on this transaction to the scheduler. The scheduler receives the message in the form {Transaction_id, data-item, operation} and then looks for the data item in the probabilistic database and locks the data-item by appropriately determining its granularity depending upon the joint probability distribution of the entities as described in the preceding paragraph. The transaction is executed and all of its PLACID properties are preserved when it has completed its execution. During its execution, the transaction may put the database in a temporarily inconsistent state but prior to and after its execution the database remains in a consistent state.
The formal algorithm is given below:

Algorithm Locking_for_Probabilistic_databases( )

Begin

Case : Begin_Transaction

Begin

Invoke Transaction Manager;
Stamp the Transaction with a unique Transaction_id;
Determine the data item and operation to be performed;

End

Case : read or write

Begin

Find Lock_item, \( l_g \), through joint probability distribution of
the entity in question.
If \( l_g \) is unlocked then assign lock to data-item;
    Execute the operation;
Else if \( l_g \) is locked, assign \( l_g \) to a FIFO queue and wait until
\( l_g \) is unlocked

End

End

This algorithm can be modified to execute the simultaneous transactions concurrently.

The next chapter investigates various aspects of probabilistic relations that are
distributed over a network. The issues related to distributed query processing and
distributed concurrency control are addressed.