

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

A transaction is a set of instructions from user program that contains sequence of read and write operations on database. Squally, a transaction is an atomic unit of handling that should be completed wholly or not at all (Habes and Hasan, 2015). In a multiuser system with large databases and multiple users executing transactions in database, such as airline reservations, online banking, online retail purchasing, stock markets, and countless other applications. Conflicts may arise among transactions and that may create problems of database consistency, due to rw/ww synchronization. Hence there exist two styles of synchronizations desirable: “Read-Write (RW) synchronization and Write-Write (WW) synchronization”.

In rw synchronization, both operations are trying to access the same data-item whereas one of them is write operation while the other is a read operation .

Subsequently, in ww synchronization, both operations are trying to access the same data-item while been in write operations. If multiple transactions are transforming the same data-item, the integrity of the database might be compromised if there is no proper concurrency control mechanism.

We employ concurrency control techniques for managing transactions concurrently accessing a certain data-item by ensuring serializable executions or to avoid interference among transactions and thus helps in avoiding errors and maintaining database consistency. Different concurrency control techniques have been established by different researchers which are unique in their own representations and methods (Ruchi and Rachna, 2012).

Over the last few decades, transaction processing and concurrency control issues have played a key role in conventional databases and hence have been an important area of research. On reviewing relevant proposed algorithms (Arun and Ajay, 2010) found out that there are structures of only but few sub-algorithms applied by entirely practical DBMS, the concurrency control algorithms are variants of two basic techniques: “two-phase locking (2PL) and timestamp ordering (TO)”. 2PL algorithm employ the mechanism of locking data-items to prevent multiple transactions from accessing the data-items simultaneously, 2PL protocols are applied in utmost commercial DBMSs but transactions encounter problems like long waiting time and deadlock, which requires deadlock detection, prevention and avoidance mechanisms. While timestamps ordering (TO) does not use locks but timestamp, therefore deadlock cannot occur. A timestamp is a unique identifier for every transaction generated by the DBMS . Timestamps values are generated in the same order as the transaction start time base on system clock or a logical-counter

(Ramez and Shamkant, 2011), also timestamp ordering protocol guarantees that transactions with older timestamp gets higher priority in the event of conflicting operations towards a specific data-item. And for each data-item, two timestamp are given, W-timestamp (W-TS) which is the last timestamp of write operation performed successfully on data-item and R-timestamp (R-TS) which is the last timestamp of read operation performed successfully on data-item (Sonal and Morena, 2015). Read/write operations continues when the last update on that data-item was carried out by a previous transaction. Whereas, transaction demanding read/write is aborted and given a new timestamp. The rule is that when there exist a data-item (X) and some transactions T_1, T_2, \dots, T_n attempts to produce a read operation on data-item $R(X)$ or a write operation on data-item $W(X)$, the timestamp of T_1 will be equate with the R-timestamp of data-item $R-TS(X)$ and the W-timestamp of data-item $W-TS(X)$ to confirm that the timestamp order of transaction implementation is not violated. If this order is violated, at that time transaction T_1 is aborted and is rollback and start as a fresh transaction with a new timestamp, the next transaction T_2 that might use the value written by T_1 must also be abort. Similarly, T_3 that might have used the value written by T_2 may likewise be abort, and so on. This outcome is identified as cascading rollback which is one of the glitches associated with basic TO (Ruchi and Rachna, 2012). Timestamp ordering algorithm can give efficient results for concurrent control problems when provided

with relevant information of transaction on the database (Jaypalsinh and Prashant, 2016).

1.2 Statement of the Problem

Despite several algorithms have been established by researchers on concurrency control, yet problems still arise due to the increase in database size. Consider a scenarios of an online transaction processing system (OLTP) e.g. an online banking system that permits multiple users (customers and bank tellers) simultaneously tries to transfer, deposit, or withdraw funds in the same account, however, without any good concurrency control policy in place, there's the a possibility for one user's transaction to conflict with another user's, in a way that user who commits his/her transactions last will override the transactions made by the first user. While the first user transactions is lost, making the database to be inconsistent. And sometimes if the second user transaction is subject to the first user transaction, and the first user transaction is aborted due to system failure, the second user transaction will also abort and rollback, this result to high rate of abortion or cascading rollback. Which will makes the system to be lazy and courses performance degradation. Hence in this work, we attempt to develop a concurrency control algorithm that addressed the above problems.

1.3 Aim and Objectives of the Study

The aim of this research is to develop an efficient technique for optimizing Timestamp Ordering Scheduler in synchronization.

The specific objectives of the research includes:

- (i). To develop an efficient model for Optimizing Timestamp Ordering Scheduler in rw/ww synchronization in DDBMS.
- (ii). To develop a hybrid algorithm that integrate Optimistic concurrent control and Thomas write rule Timestamp algorithms that guarantee similar effect as a serial execution.
- (iii). To implement the model using C# programming language and Microsoft SQL Server Database.

1.4 Significance of the Study

This research can be of great benefit in an online transaction processing system (OLTP) such as online banking, airline reservation, stock markets etc. where several transactions are performed concurrently on database. It ensures database consistency once transactions committed i.e. producing correct result, it reduces waiting time and abortion rate of transactions i.e. improving the average response time making the system to be fast, and also improve throughput of concurrent

transactions i.e. the amount of transactions that may be completely successful will increase.

1.5 Scope of the Study

We analyze the problems of concurrent control systems and focus attention on the scope of the research which is limited to an efficient technique for optimizing timestamp ordering scheduler in RW/WW synchronization.

1.6 Limitations of the Study

This work had some constraints during the research period. Getting enough materials on the area of study that will aid the writing of the entire work was not as easy as expected. And relentless effort and expenses were spend to overcome the limitation. Time and money were spent diligently to get information on implementation of the model. This was done via the internet and also the study of this research was not carried out in a wider scope due to short time constraints.

CHAPTER TWO

LITREATURE REVIEW

2.1 Overview of Timestamps and their Origin

Generally, timestamp are digital representation of specific process recorded in time and are created at the start of executing codes running on a processor, The executed codes obtains the value based on the local clock assigned to the processor and are stored on the hard drive or it may be included in data transmitted on a network. When a timestamp is generated, it is possible to associate execution of the code immediately before and after the timestamp was created with the point in time the timestamp represents. Identification of the time when the enclosing code was executed is usually the purpose of the timestamp itself (Svein, 2008). Timestamp uniquely identifies a specific process on a computer systems. The computer systems store large numbers of timestamps on their storage and typically stored it in a manner that the process is clearly recognised.

Some common sources of timestamps are:

1. File systems: each file in a computer system is associated with a timestamp based on the user actions on the file either when the file was created, last read, last written or otherwise modified.
2. Email: SMTP sever added timestamp to transmitted email, therefore email messages are associated with timestamps and some messaging protocols, such

as social network and SMS in GSM also generates timestamps to each transmitted message (Chris and Pete, 2004).

3. Logs: System logging facilities usually log events from system processes in system logs. Each event has a timestamp e.g. Network servers such as http servers typically log each user transaction in a system log.
4. Database: Timestamp are used to organised simultaneous transactions accessing a data-item on database to avoid conflicts among each other, timestamps is assigned to every transactions at the beginning of execution by the DBMS to identify each transaction. Typically, transactions obtain timestamp values in the order in which they are submitted to the system and priority of obtaining entrance to the data-item is determine on the timestamp order. One of the potential ways timestamps are created is the use of the current date/time value of the system clock and confirm that no two transactions are assigned with the matching timestamp value. Alternative way to implement timestamps is the use a logical counter that increases each time a transaction is executed. The maximum value of the logical counter may be finite. So periodically, whenever transactions stop executing for little period of time, the system reset the counter to zero (Ramez and Shamkant, 2011).

2.2 Distributed Database Management System (DDBMS)

When database size is very large and are accessed from remote site, it requires partitioning and stored on different machines for fast data access. Distributed Database Management System comprises of different number of sites connected together by a communication network (Arun and Ajay, 2010). Each site is a centralized database that consist of:

1. Transactions (T)
2. Transaction Manager (TM)
3. Data Manager (DM)
4. Data.

Users communicate with the DBMS by executing transactions that may be embedded queries in application programs written in a high level programming language, while TM supervise and manages transactions between users and the database and DM manages the actual data (database).

In Distributed Database Management System, transactions interact with TMs and TMs interact with DMs while DMs interact and manage the Data. Meanwhile the interaction between TMs with each other and DMs with each other is not possible.

Figure 1.0 shows the components DDBMS.

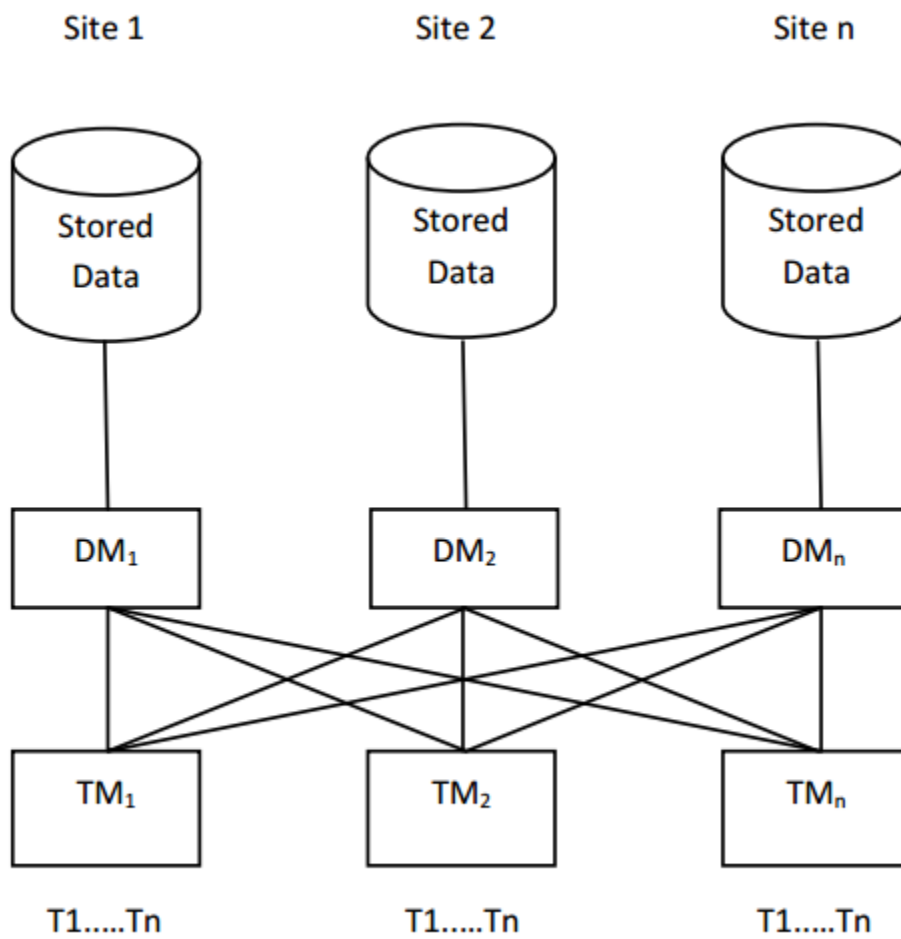


Figure 2.1 DDBMS Architecture (Rinki et al, 2011)

2.3 The Concept of Transactions and Schedules

A transaction is a set of instructions from user program that consists a sequence of read and write operations on database. In other words, a transaction is an atomic unit of processing, that should either be completed in its entirety or not done at all (Ramez and Shamkant, 2011). The read and write operations of a transaction must specify as its final operation either commit (complete) or abort (terminate and undo all performed operations). Intuitively, a user program that contains database queries (SQL query), e.g. select, insert, delete and update statement.

For recovery purposes, the database management system (DBMS) requests to keep path the following operations for every transactions which include:

1. **Begin_Transaction:** This indicates the start of transaction execution.
2. **Read Or Write:** These identify read or write operations on data-items.
3. **End_Transaction:** This shows the end of READ and WRITE operations and identifies that transaction have ended.
4. **Commit:** This shows a complete end of transaction, so any modifications or alterations made by the transaction are permanently established in the database.
5. **Abort:** This shows an unsuccessful end of transaction, so any modifications or alterations made by the transaction on database must be undone.

Figure 2.2 illustrates how a transaction moves through its execution states.

A transaction exists in an **active state** immediately after it begins execution, where the READ and WRITE operations performed its processes. Once the transaction ends, it enter the **partially committed state**. At this moment, some recovery protocols need to check if a system failure will not occur, once this is complete, the transaction now enter the **committed state**, which means the transaction is successfully ended and all its changes must be recorded permanently in the database. Else, if a system failure occurs, the transaction is aborted and go into the **failed state** or in the active state, if the transaction is aborted, it also go into the **failed state** which indicates that the transaction is unsuccessfully ended and undo the changes of its WRITE operations in the database. Thus, the transaction start over and resubmitted as a new transactions either by the user or automatically. The **terminated state** corresponds to the transaction leaving the system.

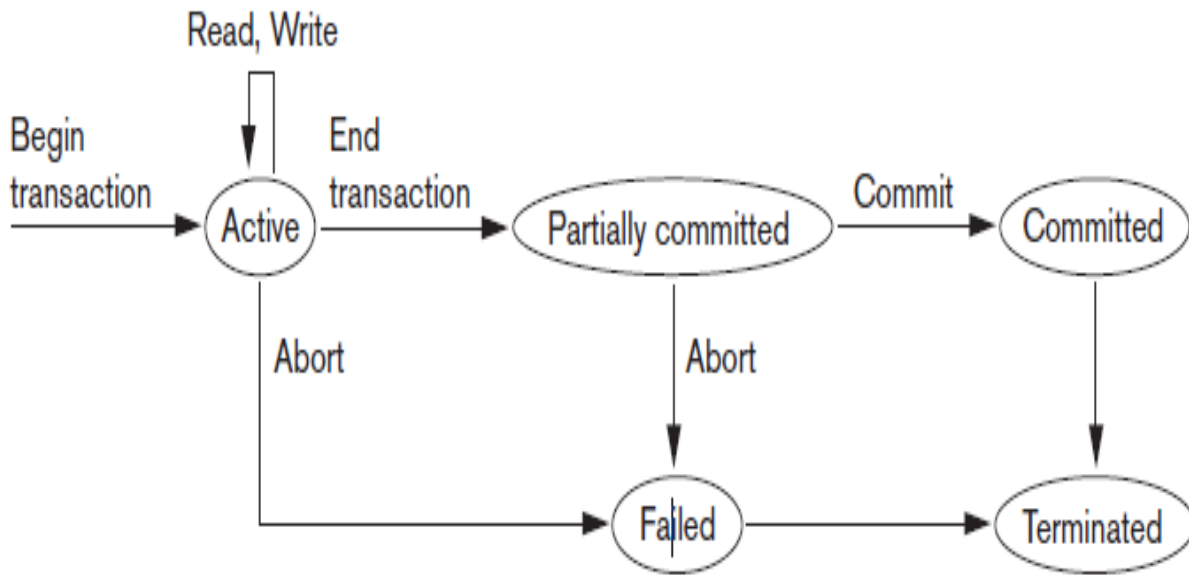


Figure 2.2: Execution state of transaction (Ramez and Shamkant, 2011)

Transactions have four properties: atomicity, consistency, isolation and durability (ACID). The properties of transaction are defined below:

1. **Atomicity:** Requires that every operations of a transaction must be complete else transaction must aborted and rollback.
2. **Consistency:** Implies that, the database must be in a correct state. If any operations in a transaction fails, the initial consistent state must be restored.
3. **Isolation:** Implies that, when a transaction accessed a data-item no other transaction will accessed that same data-item until the transaction is commit.
4. **Durability:** Indicates that committed transaction are guaranteed. That its results are permanent and cannot be changed or erased from the database.

2.3.1 Conflicting Operations

When transactions T₁ and T₂ performs operations on data-item. The operations will be in conflict, if they accessed the same data-item (A), and at least either T₁ or T₂ performs a write operation, i.e.

- (a) R-T₁(A) and W-T₂(A).
- (b) W-T₁(A) and R-T₂(A)
- (c) W-T₁(A) and W-T₂(A)

Here R- T₁(A) signifies read operation on data-item (A) by transaction T₁, W- T₁(A) signifies write operation on data-item (A) by transaction T₁ etc. The conflict operations between T₁ and T₂ shows the significant of the order of their executions,

i.e. if two or more transactions performing conflict operations, their execution may result to a state of database inconsistency. Note that the conflict relation of read operations does not matter, Table 2.1 shows the conflict relation of read and write operations.

2.3.2 Schedule

A schedule (sometimes called history) is a set of operations (read, write, abort, or commit) from a list of transactions, and the sequence of execution in which the operations are in the schedule must be the same in which they are executed in the transaction (Raghu and Johannes, 2003). Figure 2.3 shows a Schedule of Transactions T1 and T2. T1 contains two operations, read(A) and write(A), while T2 also contains two operations, read(A) and write(A). The Figure 2.4 shows the possible order of executing T1 and T2. In schedule1, the operations of T1 are executed entirely before T2 start execution. In schedule2, the operations of T2 are executed entirely before T1, which shows that schedule1 and schedule2 are serial execution. While the operations in schedule3 and schedule4 are interleaving, which means that their execution are serializable.

Table 2.1: Compatibility Matrix for T_i and T_j (Jaypalsinh and Prashant, 2016)

Compatibility Matrix for T_i and T_j		T_i	
		read(x)	write(x)
T_j	read(x)	<input type="checkbox"/>	<input type="checkbox"/>
	write(x)	<input type="checkbox"/>	<input type="checkbox"/>

T1	T2
R1(A)	
W1(A)	
	R2(A)
	W2(A)

Figure 2.3: A schedule of Transactions T1 and T2 (Raghu and Johannes, 2003).

Schedule1 = { R1(A) W1(A) R2(A) W2(A) }

Schedule2 = { R2(A) W2(A) R1(A) W1(A) }

Schedule3 = { R1(A) R2(A) W1(A) W2(A) }

Schedule4 = { R2(A) R1(A) W2(A) W1(A) }

Figure 2.4: The sequence of execution T1 and T2 (Kamal, 2014).

2.3.3 Serial Schedule

A schedule is serial if every single operations of a transaction are executed and then the operations of another transaction, etc. That is, for a schedule to be serial the sequence of transactions must be $T_1, T_2, T_3, \dots T_n$, the transaction T_1 must be executed entirely (committed) before the next transaction T_2 starts execution. Serial schedules are naturally consistent and result in correct execution and the transactions don't interleave among each other (Utku Aydonat, 2011). Following this example where two accounts X and Y have the balances amount of 500 and 1000 naira in a bank, respectively, and two active transactions T_1 and T_2 , in whereas money are transferred from one account to another. Figure 2.5 shows a schedule involving T_1 and T_2 while table 2.2 illustrates the order of execution T_1 and T_2 are serial execution.

T1	T2
R(x)	
x=x-100	
W(x)	
	R(x)
	T=x*0.2
	x=x-T
	W(x)
R(y)	
y=y+100	
W(y)	
	R(y)
	y=y+T
	W(y)

Figure 2.5: A schedule involving two Transactions (Raghu and Johannes, 2003).

Table 2.2: A serial execution of transactions T1 and T2

Time	Transactions	steps	Stored Values
1	T1	R(x)	500
2	T1	$x = x - 100$	
3	T1	W(x)	400
4	T1	R(y)	1000
5	T1	$y = y + 100$	
6	T1	W(y)	1100
7	T2	R(x)	400
8	T2	$T = x * 0.2$	
9	T2	$x = x - T$	
10	T2	W(x)	320
11	T2	R(y)	1100
12	T2	$y = y + T$	
13	T2	W(y)	1180

2.3.4 Serializable Schedule

A schedule is serializable if when executed, the output and effect on the database are equal to other serial schedules of same transactions. Since, serial schedules are naturally consistent and result in correct execution, then the serializable schedule are also consistent and result in correct execution. Consider again from the above example the schedules in Figure 2.5, in this case table 2.3 illustrates the order of execution in which operations in T1 and T2 are interleaving also the output and effect are equal to that of serial execution in table 2.2.

The schedule below shows that T1 perform the read-write operations on data-item (x) switch to T2, T2 performs its read-write operations on (x) switch again to T1, T1 performs read-write operations on data-item (y) switch to T2, T2 perform read-write on data-item (y) and commit.

Schedule = {R1(x) W1(x) - R2(x) W2(x) - R1(y) W1(y) - R2(y) W2(y)}

.

Table 2.3: A serializable execution of transactions T1 and T2

Time	Transactions	steps	Stored Values
1	T1	R(x)	500
2	T1	$x = x - 100$	
3	T1	W(x)	400
4	T2	R(x)	400
5	T2	$T = x * 0.2$	
6	T2	$x = x - T$	
7	T2	W(x)	320
8	T1	R(y)	1000
9	T1	$y = y + 100$	
10	T1	W(y)	1100
11	T2	R(y)	1100
12	T2	$y = y + T$	
13	T2	W(y)	1180

2.3.5 Problems in Concurrency Control

If transactions are executed serially, i.e., sequentially with no overlap, meaning there will be no existence of concurrent transactions. However, if concurrent transactions with interleaving operations are allowed in an unorganized manner, some unpredictable, undesirable result may occur.

Consider for example two customers Customer1 and Customer2 deposit money in the same bank account just about the same time.

As illustrated in Figure 2.6, Customer1 reads the amount in the account (200), make a local copy in a private work space and then adds 150 to the account (temp) to make it 350. Customer2 reads the amount in the account as well, which was still 200, and adds 50 to Customer2's local copy, making it 250. Then, Customer1's update of 350 was written to the original database. Subsequently, Customer2's update of 250 was also written back to the original database. At this point, an incorrect state has resulted and 150 has been lost: the correct balance should be 400.

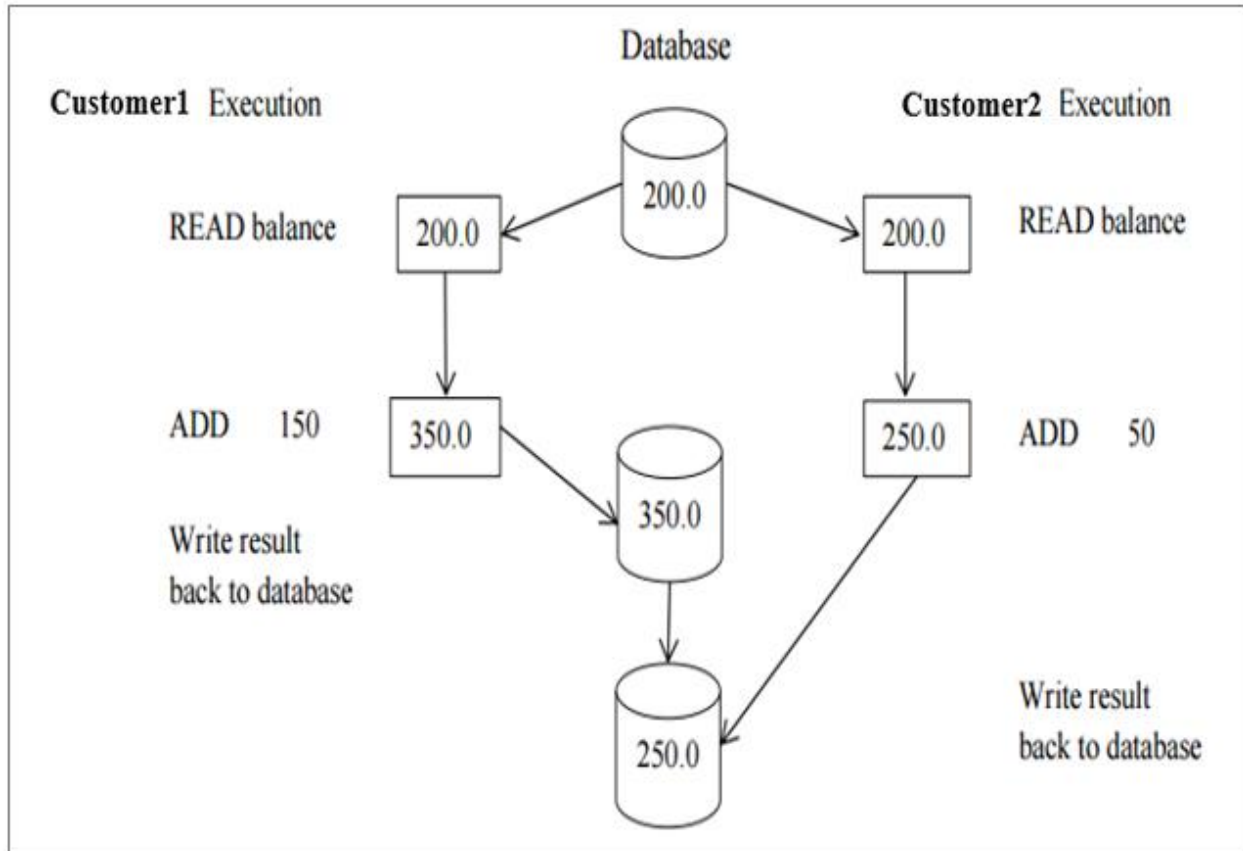


Figure 2.6: Lost update problem (Kamal, 2014)

Inconsistent Retrievals: suppose that the customers, Customer1 and Customer2 concurrently perform the transactions T1 and T2 respectively:

1. T1 transfers 1000 from a current account to the same person's saving account.
2. T2 prints the total amount of the both accounts (current and savings). As illustrated in Figure 2.7, T1 reads the amount from the current account (1200) and subtracts 1000 from it, the balance is 200, which is written back to the current account in the database. Then, at approximately the same time, T2 reads the amount of both accounts and then prints the total. T1 continues and reads the amount of the savings account and adds the 1000 to the previous balance. The new amount for the savings account will be 1500, which is be updated to the savings account in the database. This time, the final result placed in the database is correct, but the order of execution is incorrect, because the total amount printed by T2 is 700, whereas the real total balance is 1700.

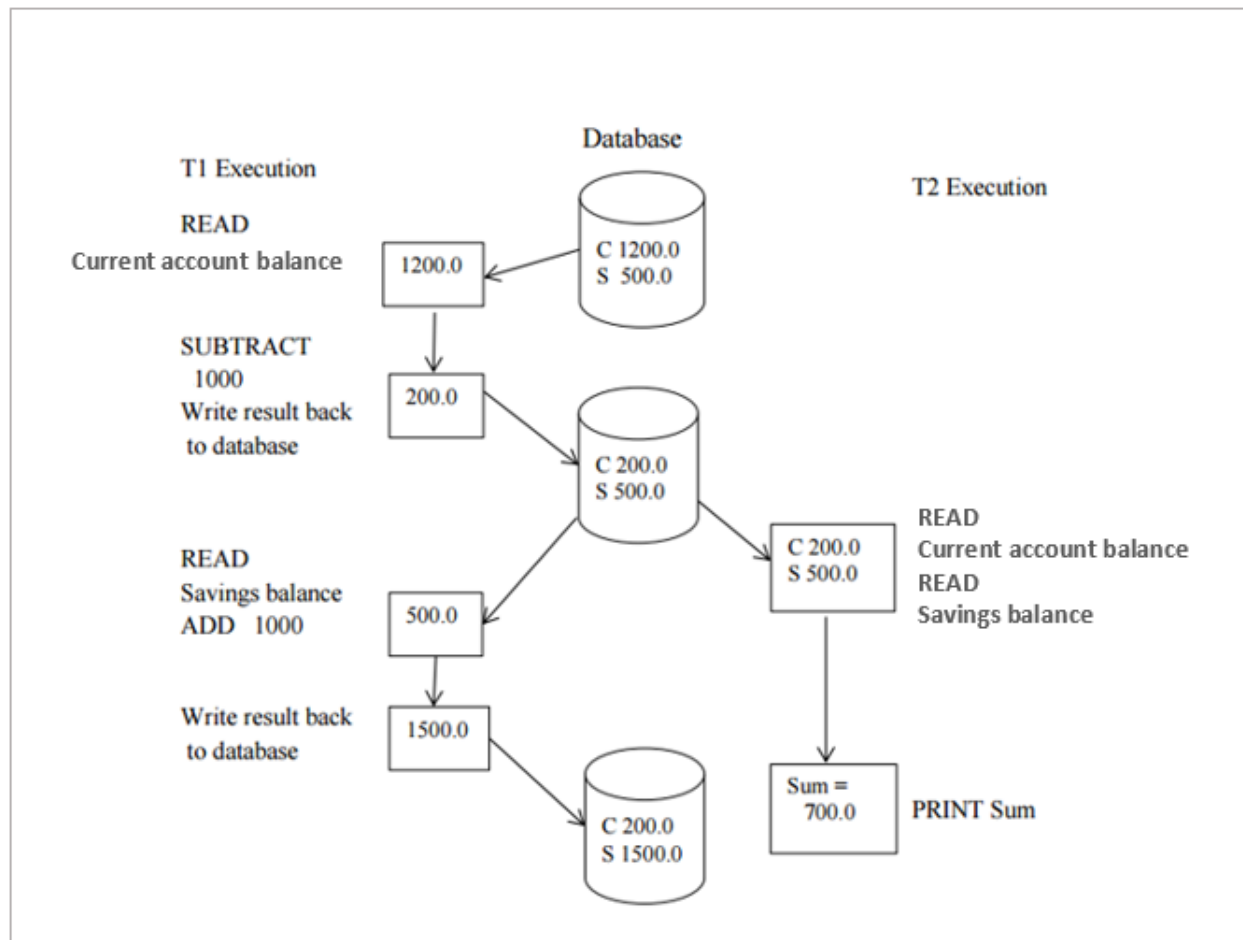


Figure 2.7: Inconsistent retrievals problem (Kamal, 2014)

2.4 Concurrency Control in DBMS

Concurrency control is a method or technique that co-ordinate simultaneous access of transactions on a shared data item such that the consistency of the database is maintained. Consistency is when any transaction start execution, the database should be in consistent state and when it ends, the database should also be in consistent state. And the output produced by the transaction should be correct, (Rinki et al, 2011).

Concurrency in a Database Management System (DBMS) is a situation in which several transactions or users from remote locations, simultaneously tries accessed the same data-item. Whereas data-item can be a database, a database table or a record. In such environs, the precedence of accessing data-item must be organised to avoid conflicts among transactions which may lead to database inconsistency.

This problem becomes difficult in distributed databases since the data is not stored at one place. The user can access the data from any site and the controlling instrument at other site may not identify it instantly. Since the goal of a concurrency control technique is to produce serializable executions and also put into effect the properties of transaction (ACID). Researchers have made valuable research in the development of concurrency control techniques in last few years based on performance and efficiency (Arun and Ajay 2010) Based on their works, they found out that there are composition of only a few sub-algorithms used by all practical

DBMS, the concurrency control algorithms are a variants of two basic techniques, such as two phase locking (2PL) and timestamp ordering (TO) and different algorithms exhibit good performance on different situations and applications, some methods prefers locking approach while others are centred on time stamp.

2.4.1 Taxonomy of Concurrency Control Algorithm (CCA)

Concurrency control and recovery issues are important area of research for several decades (Bernstein et al, 1987). Research in the region on concurrency control has led to the improvement of many CCAs both in centralized database system (single site) and distributed database system.

Concurrency control algorithms are typically classified into two broad categories, which include pessimistic and optimistic view (Choe, 2008).

2.4.2 Pessimistic View are environs with many conflicting transactions, thus verifies that executions of transactions will be correct before they are allowed to occur. This decreasing the complication of recovery but in practice is achieved only at the price of increased overhead and significantly decreased the amount of concurrency. Pessimistic view are centered on locking and non-locking protocol. Locking protocol allows a user from altering data-item in a way that affects other users. After a user executes an action that causes a lock to be applied, other users are blocked i.e., cannot execute actions that would clash with the lock while waiting for the owner to releases it, usually when the transaction is complete. The more

transactions are concurrently executed, the more the risks of transactions will be blocked, which causes deadlock and increased waiting time. On the other hand non-locking is based on timestamp ordering protocol.

2.4.3 Optimistic View are environs with not too many conflicting transactions, thus allowing transactions to execute concurrently without checking the resulting execution will be valid until transaction completion. This allows incorrect operations to occur and thus places increased importance on recovery since the effects of invalid operations of transaction must be abort after they are detected. Incorrect transactions must be “rolled back” and the work of such transactions is lost (Quazi and Hidenori, 2005). The core advantage of optimistic view over pessimistic view is conflicts are relatively rare, it is deadlock free and allows transactions to execute without waiting for other transactions. Figure 2.8 shows the classification of concurrency control algorithms.

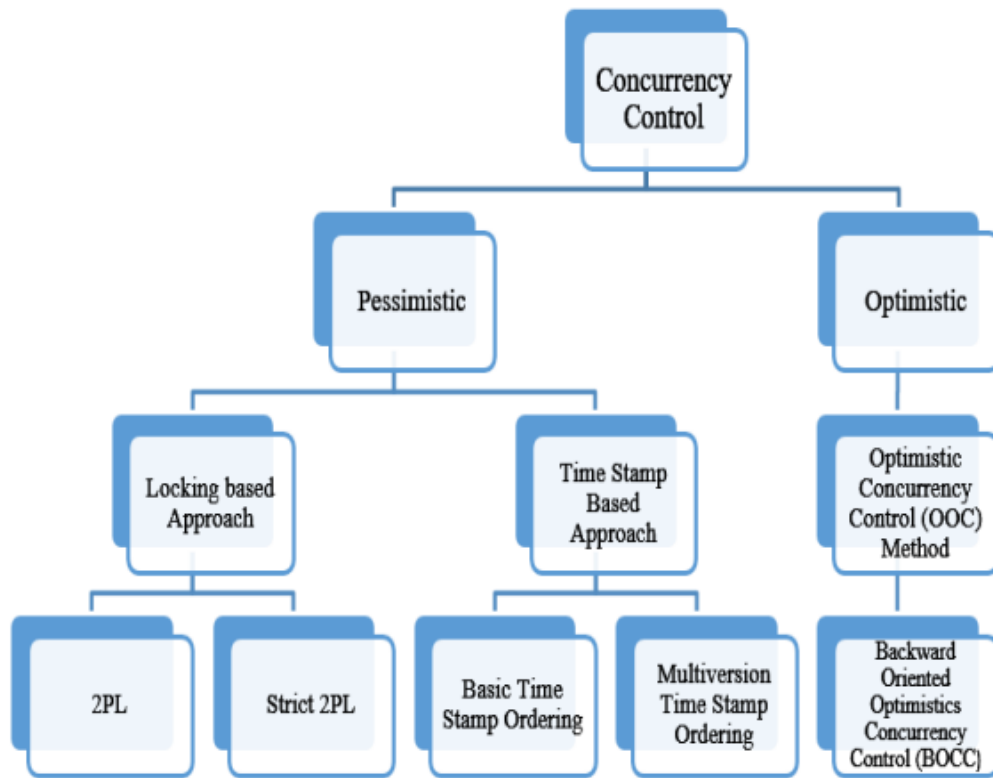


Figure 2.8: Basic taxonomy of Concurrency Control (Jaypalsinh and Prashan, 2016)

2.5 Implementation of Concurrency Control Techniques

The two common concurrency control algorithms in conventional database systems are two phase locking (2PL) and timestamp ordering. There are many variants on both algorithms.

2.5.1 Two Phase Locking Algorithm (2PL)

2PL is based on locking protocol and is the most widely used concurrency control algorithm for conventional database systems where transactions are simple and short. It ensures serializability and provides effective concurrency control. 2PL uses two types of locks on data-items which are:

1. Read locks which must be obtained before reading data-item and
2. Write locks which must be obtained before writing on data-item.

Multiple transactions are permitted to concurrently hold the same read lock, but if a transaction grips a write lock, no other transaction may hold the corresponding read or write lock. Transactions are required to get the appropriate locks for each data-item they access and are restricted so that no locks may be free until after all necessary locks have been acquired. By obtaining locks in this mode, serializability is guaranteed, but deadlock between transactions is possible. One rule in 2PL states that after a transaction has released one of its locks, should not at any point request another lock. Instead, a transaction should be certain that it will not demand another

lock before it released a lock. Transactions are executed in two phases which includes:

3. Growing phase is where transactions acquires locks and have right to use data-items,
4. Shrinking phase is during which transactions releases locks,

Whereas the lock point is the actual point when the transaction has obtained all its required locks but has not yet started to release them. Thus the lock point indicates the end and the beginning of the growing and the shrinking phase of the transaction. Figure 2.9 shows that, as soon as the execution of the accessed data-item is committed, it releases the locks for others transactions waiting to obtain the locks. This increases the degree of concurrency. Sometimes, the implementation of 2PL is difficult, since transaction has obtained its locks and need not lock at another data-item finally released the lock, if the transaction aborts after it releases a lock, other transactions that may have accessed the unlock data-item may also abort. This causes the problem of cascading aborts. Which may be overcome by strict two phase locking (S2PL). At the shrinking phase, S2PL releases all its locks at the same time when transaction commits or aborts. Thus the lock graph is as made known in Figure 2.10.

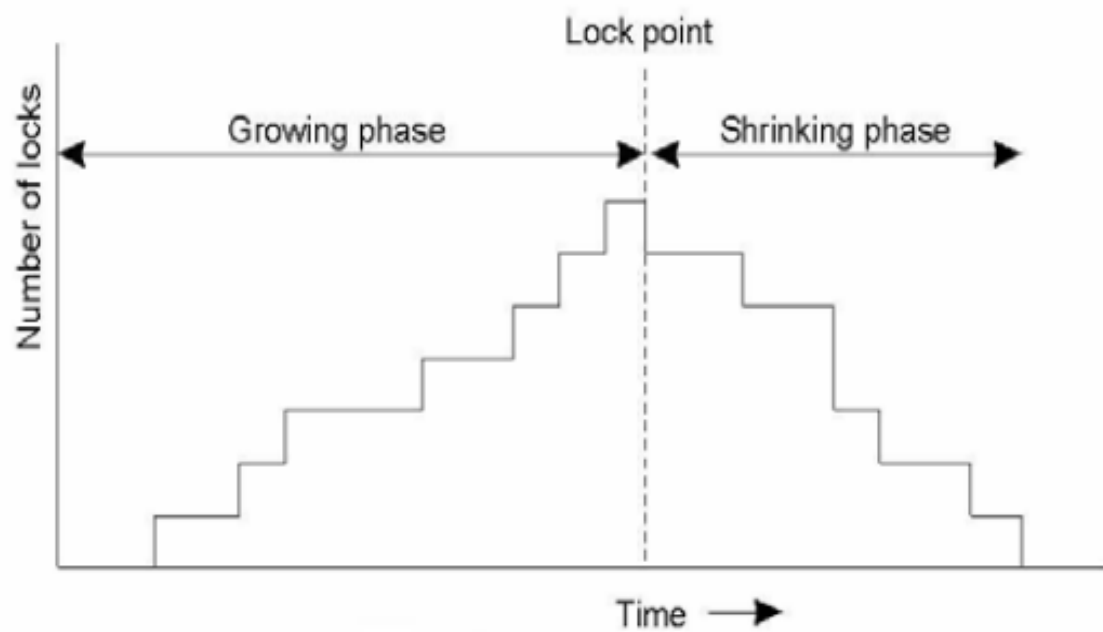


Figure 2.9: Two-phase Locking (Arun, 2010)

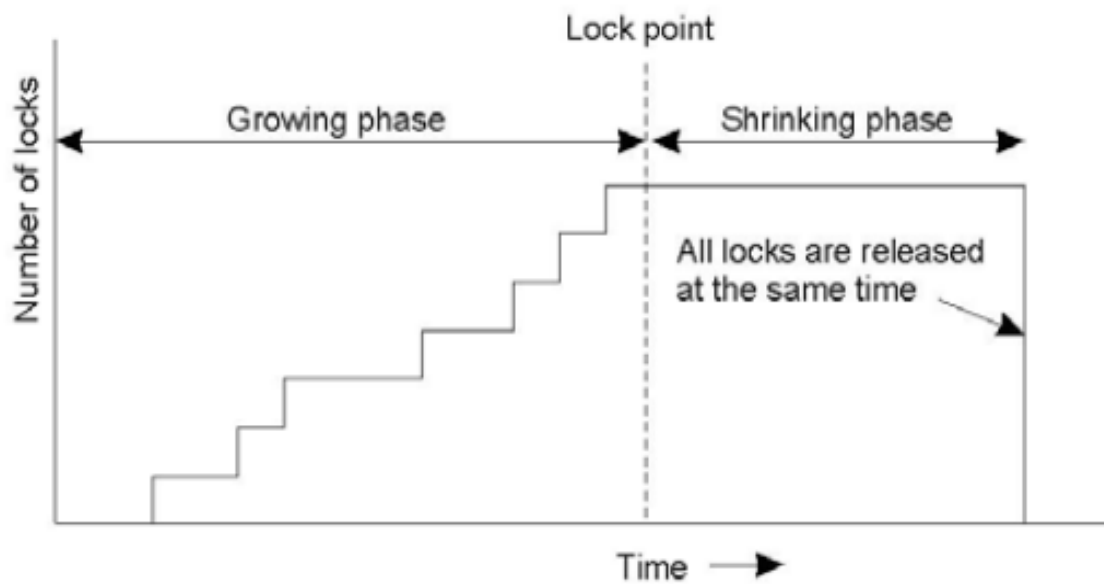


Figure 2.10: Strict two-phase Locking (Arun, 2010)

A transaction T_i issues the normal read/write instruction, without explicit locking calls.

The operation **read(X)** is processed as:

```
    if  $T_i$  has a lock-x
    then
        read(X)
    else
        begin
            if necessary wait until no other
                transaction has a lock-x
            grant  $T_i$  a lock-x ;
            read(X)
        end
```

write(X) is processed as:

```
    if  $T_i$  has a lock-x
    then
        write(X)
    else
        begin
            if necessary wait until no other transaction has any lock-x,
            if  $T_i$  has a lock-x
            then
                upgrade lock on X to lock-x
            else
                grant  $T_i$  a lock-x
            write(X)
        end;
    end;
```

All locks are released after commit or abort

2.5.2 Timestamp Ordering (TO)

Timestamp ordering provides edge over locking protocol, it eliminates the major bottleneck of deadlock for transactions and transactions do not wait for each other over a long period of time needlessly, instead it aborts the conflicting transactions. However the problem of starvation can surface for long transactions, in such situation rate of cascading rollbacks are high (Jaypalsinh and Prashant, 2016).

The Transaction Manager (TM) which is a component of the DBMS assigns timestamp to transaction T at the starts of execution denoted by $Ts(T)$.

If a transaction T_1 starts execution and assigned a timestamp $Ts(T_1)$, and another transaction T_2 starts execution and assigned a timestamp $Ts(T_2)$, then

$Ts(T_1) < Ts(T_2)$. The timestamps of the transactions must ensure the order of serializability. That is, the produced schedule is alike to a serial schedule in which transaction T_1 precedes transaction T_2 .

To implement timestamps ordering, each data-item X has two timestamp values which are: The Write Timestamp X denoted $W-Ts(X)$ and the Read Timestamp of X denoted $R-Ts(X)$

- (a) $W-Ts(X)$ is the last timestamp of any transaction that execute a write operation on (X) successfully.
- (b) $R-Ts(X)$ is the last timestamp of any transaction that execute a read operation on (X) successfully.

W-Ts(X) and R-Ts(X) are updated whenever a new read (X) or write (X) Operations is executed.

2.5.3 Basic Timestamp Ordering Algorithm (BTSO)

The timestamp ordering algorithm ensures that conflicting operations (read-write) are executed in timestamp order. This protocol operates as follows (Ramez and Shamkant, 2011):

Case1: If a transaction T1 issues read(X).

- (a) If $Ts(T1) < W-Ts(X)$, then T1 needs to read a value of X that was already overwritten by newer transaction with larger timestamp value, hence, the read operation is rejected, and T1 is rolled back and start as a new transaction.
- (b) If $Ts(T1) \geq W-Ts(X)$, then the read operation is executed, and R-Ts(X) is set to the maximum of R-Ts(X) and Ts(T1).

Case2: If a transaction T1 issues write (X).

- (a) If $Ts(T1) < R-Ts(X)$, then the value of X that T1 is producing was needed previously, and the system assumed that the value would never be produced. Then the write operation is rejected and T1 is rolled back.
- (b) If $Ts(T1) < W-Ts(X)$, then T1 is attempting to write an obsolete value of X. Hence, the write operation is rejected and T1 is rolled back.
- (c) If $Ts(T1) \geq W-Ts(X)$ and $Ts(T1) \geq R-Ts(X)$, then the write operation can be performed and sets W-Ts(X) to Ts (T1).

2.5.4 Thomas' Write Rule Timestamp Ordering Algorithm

In this algorithm, rules for read operations remain the same. But the rules for write operations are slightly different from the Basic Timestamp Ordering. Thomas' Write Rule is an upgrade version of the Basic Timestamp Ordering in which obsolete write operations can be ignored under certain conditions (Habes and Hasan, 2015).

Case1: If a transaction T_1 issues write (X).

- (a) If $Ts(T_1) < R-Ts(X)$, then the value of X that T_1 is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and T_1 rolled back.
- (b) If $Ts(T_1) < W-Ts(X)$, then T_1 is attempting to write an obsolete value of X. Hence, this write operation can be ignored.
- (c) If $Ts(T_1) \geq W-Ts(X)$ and $Ts(T_1) \geq R-Ts(X)$, then the write operation can be performed and sets $W-Ts(X)$ to $Ts(T_1)$.

Thomas' write rule ignore the write operations and executes schedules that are not conflict operations but are correct. The adjustment of transactions makes it possible to produce serializable schedules that would not be possible under basic timestamp algorithm

2.5.5 Optimistic Concurrency Control Algorithm (OCCA)

The basic aim of OCCA is the possibility to let transactions execute freely since the presumption is that most transactions do not conflict with each other. OCCA perform more efficiently than Locking based algorithm in an environment where there are fewer conflicting transactions. The phases in which OCCA executes a transaction are Read, Validate and Write phase. Figure 2.11 illustrate the phases of OCCA.

Read phase: each transaction read the value from data-item to a private workspace area, does the computations and makes local copies of its modifications.

Validation phase: if the transaction is committed, the transaction has to be validated to check if its modifications are compatible with the value of the data-item, if their compatibility is false then the transaction is aborted and its private workspace is also cleared. However, if otherwise, it goes into the next phase.

Write Phase: Data objects are copied from the local workspace into the database, once a transaction is in the write phase, it is considered to be complete. In the write phase, transaction makes all its updates permanent in the database.

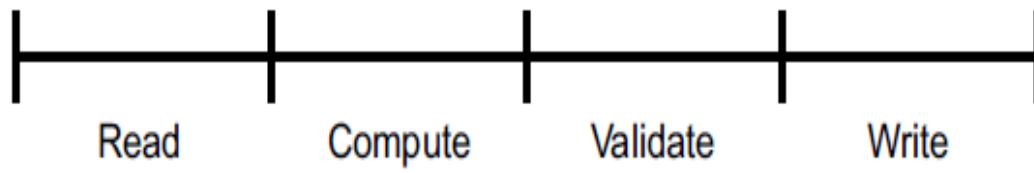


Figure 2.11: Phases of Optimistic Concurrent Control (Tamer and Patrick, 2010)

2.5.6 Optimistic Concurrency Control (Also called Validation Method)

In OCCA, transactions are assigned timestamp only at the beginning of the validation phase. The validation criteria are used to determine the ordering of the transactions.

For two transactions T_1 and T_2 , such that T_1 is before T_2 , i.e. $Ts(T_1) < Ts(T_2)$, one of the following validation criteria must satisfied these conditions (Quazi and Hidenori, 2005).

1. If T_1 complete its phases before T_2 , Then T_1 and T_2 are executed in serial order and the writes of T_1 should not be overwritten by the write of T_2 .
2. If T_2 commit write phase before T_1 , The writes of T_2 should not affect the read phase of T_1 .
3. If the read phase of T_1 finishes before T_2 and they both do not read/write any common objects. Therefore, the operation can be done simultaneously

By the validation criteria above, we have shown that T_1 and T_2 do not conflict with each other and are allowed to execute simultaneously. If the validation phase is successful the modification is allowed at the write phase then the next transaction is picked for validation. Thus it's clear that only one transaction is allowed to enter write phase and any other transactions would be put on hold.

When applying optimistic concurrency control, each time the server reads a data-item and try to update it, the server makes a copy of the "version number" of the data-item and stores that copy in a private workspace for later reference. When it's time to write the updated data back to the database, the server compares the original version number that it read with the version number that the database now contains. If the version numbers are the same, then no one else changed the data-item and we can write our updated value. Nevertheless, if the value we originally read and the current value on the database are not the same, then someone has changed the data-item since we read it, and whatever operation we did is probably out dated, so we reject our version of the data-item and give the user an error message. Certainly, each time that we update a data-item

2.6 Related Works

(Arun and Ajay 2010) proposed a distributed architecture for transactions synchronization in distributed database systems. The analysis of their approach is a decomposition of the concurrency control problem in two sub problems: read-write and write-write synchronization. They stated that the performance of their proposed system will be evaluated in their future work.

(Wette et al 2004) present a comparative study of some concurrency control algorithms for clustering based communication network. They proposed a hybrid

algorithm that integrates two high performance concurrency control mechanisms: the Pessimistic Tel-ORB Algorithm, Their proposed algorithm uses both the Two-Phase locking and Optimistic Concurrency Control methods in an exclusive mode. Two-Phase locking algorithm to verify transaction conflicts early in their execution phase and the optimistic algorithm which investigates the presence of conflict after the execution phase. Thus obviously their protocol shows better performance than the original optimistic concurrency control protocol.

(Karami and Dastjerdi, 2011) Proposed OPCT concurrent control algorithm centred on optimistic concurrency control method in mobile databases. They introduced their proposed OPCT concurrent control algorithm using serialization graph. Their results shows that OPCT algorithm reduces abortion rate and waiting time of transactions in compare to 2pl and optimistic algorithm. But the difficult of their proposed system is overhead of timestamps and their calculation.

(Ruchi and Rachna, 2012) analysed the effectiveness of concurrent control techniques in Databases. They analysed several techniques on concurrency control such as two-phase locking, timestamp ordering and optimistic-based mechanisms. Their result is optimistic based technique is free from deadlock and is efficient than two-phase lock and timestamp ordering. But its waiting time is less than locking and more than timestamp ordering. (Kamal, 2014) proposed transaction concurrent control for resource constrained application. They explore the previous techniques

relating to timely transactional systems for remote clients and centralized database. They used the first technique developed to decrease disk access time via local caching of state to tackle the problems of prevalent in real-time databases. His results gave efficient throughput to improve battery life for mobile device and minimized time complexity of the system. He suggested that it would be worth investigating the proposed approach with the new transactional phase order in non-blocking software transaction memory, order to achieve further advance in the field.

(Saeid and Mahdi, 2015) proposed modeling timestamp ordering method using coloured Petri Net. Their analysis using state space shows that timestamp ordering method in text books may face with starvation. They also analyze state space explosion of model and they observed that timestamp ordering method falls in infinite loop and therefore has starvation. They proposed concept of precedence graph produces an efficient throughput of the system. They stated that proposing new variations of TO method that inherently eliminates starvation of it is an open problem.

In (Shefali and Samrat, 2015) they revisited performance issues in concurrent transaction execution in distributed database management system. They stated that it is very difficult to find an ideal optimum solution for the distributed data with respect to the following challenges: cost of network, resources, response time, access time, memory usage, processing time, etc. Their review shows that to obtain

optimum solution these challenges should be minimized which would be done with the use of better algorithm for different principles of DDBMS.

(Jaypalsinh and Prashant, 2016) proposed a study and comparative analysis of basic Pessimistic and optimistic management system. They highlighted on the pros and cons of pessimistic and optimistic approaches for concurrency controls. They observed that Pessimistic locking based approach is suitable for update-intensive for read operation. And the optimistic method is a backward oriented concurrency control method which works on three different phases: Read, Write and Validation. They did not compare the performance of all basic methods of concurrent control to provide optimum performance of their proposed optimistic concurrency control methods.

CHAPTER THREE

MATERIALS AND METHODS

3.1 Analysis of the Existing System

The problem of co-ordinating concurrent access to a database system has been of interest by lots of researchers and several concurrent control have been introduced.

(Ruchi and Rachna, 2012) analysed the effectiveness of concurrent control techniques in Databases. They analysed several techniques on concurrency control such as two-phase locking, timestamp ordering and optimistic-based mechanisms.

Their result is optimistic based technique is free from deadlock and is efficient than two-phase lock and timestamp ordering. But its waiting time is less than locking and more than timestamp ordering. (Wette et al 2004) present a comparative study of some concurrency control algorithms for clustering based communication network.

They proposed a hybrid algorithm that integrates two high performance concurrency control mechanisms: the Pessimistic Tel-ORB Algorithm, Their proposed algorithm uses both the Two-Phase locking and Optimistic Concurrency Control methods in an exclusive mode. Two-Phase locking algorithm to verify transaction conflicts early in their execution phase and the optimistic algorithm which investigates the presence of conflict after the execution phase. Thus obviously their protocol shows better performance than the original optimistic concurrency

control protocol (Karami and Dastjerdi, 2011) Proposed OPCT concurrent control algorithm centred on optimistic concurrency control method in mobile databases. They introduced their proposed OPCT concurrent control algorithm using serialization graph. Their results show that OPCT algorithm reduces abortion rate and waiting time of transactions in compare to 2pl and optimistic algorithm. But the difficult of their proposed system is overhead of timestamps and their calculation.

3.1.1 Disadvantages of the Existing System

The disadvantages of the existing system are as follow:

1. Its waiting time is less than two-phase locking and more than timestamp ordering.
2. Traditional database system depends frequently on locking to obtain serializability among concurrent transactions.
3. If there is conflicting transactions at the read phase, it is difficult to determine the transaction to execute at the validation phase.
4. Prone to deadlock.

The Architecture of the existing system is shown in Figure 3.1.

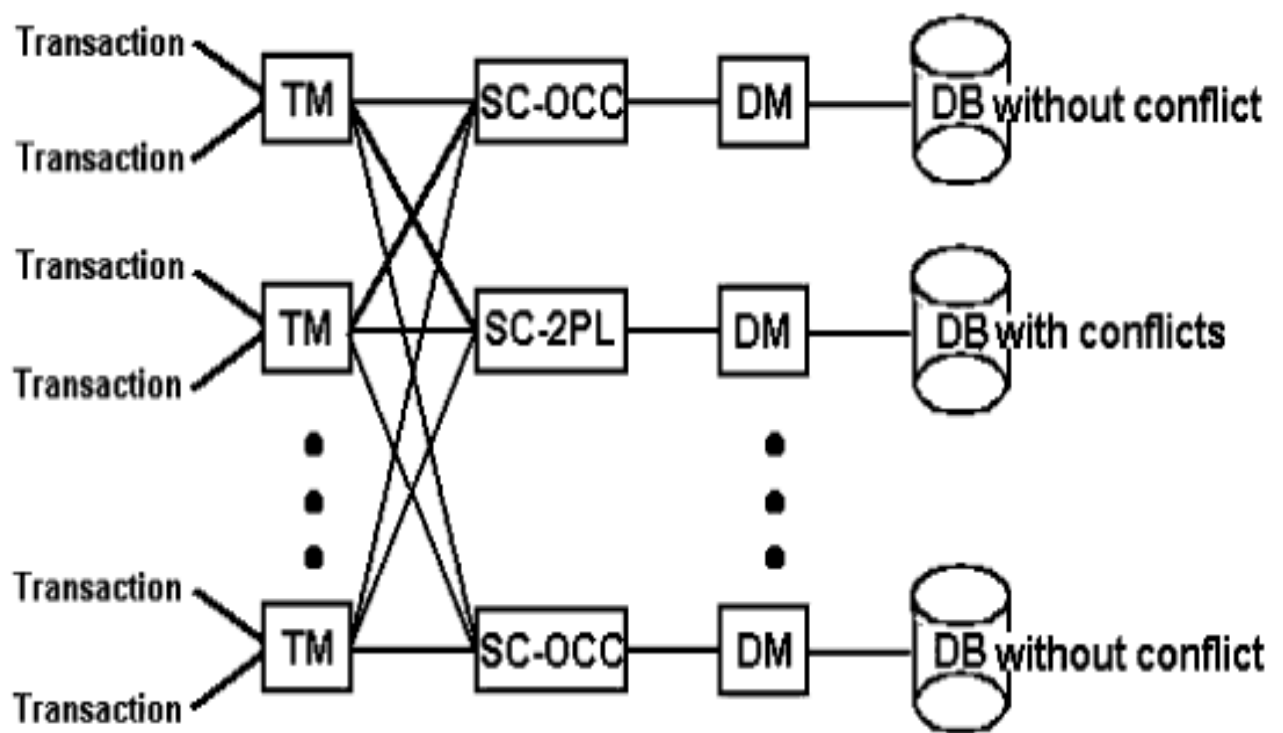


Figure 3.1: Architecture of the Existing System. Source (Wette et al 2004)

3.2 Analysis of the Proposed System

In this work, we proposed an Optimistic Thomas Algorithm (OPTH) which is a hybrid of Optimistic Concurrency Control Algorithm and Thomas' Write Rule Timestamp Algorithm for optimizing the serialization of RW/WW synchronization in DDBMS. The algorithm consists of several phases: Read phase, Validation phase and Write phase. Figure 3.2 shows transactions and the symbols allocated to each transaction at various stages. T is an active transaction reading/writing the data-item, $Ts(T)$ is the timestamp allocated to the transaction at the start of execution, $Ts(T_v)$ is validating Timestamp of the active transaction, D is the data-item to be read/write, $Ts(D)$ is the timestamp of the data-item. This algorithm is executed for all data-item.

In the read phase: When a transaction execute a Read/Write operation, the data-item timestamp $Ts(D)$ is compare against the timestamp allocated to the transaction $Ts(T)$. That is, if $Ts(T) > Ts(D)$ is true, the transaction read the values from data-item to a private workspace area, does the computations, makes local copy of its modifications in the private workspace only and not to the database.

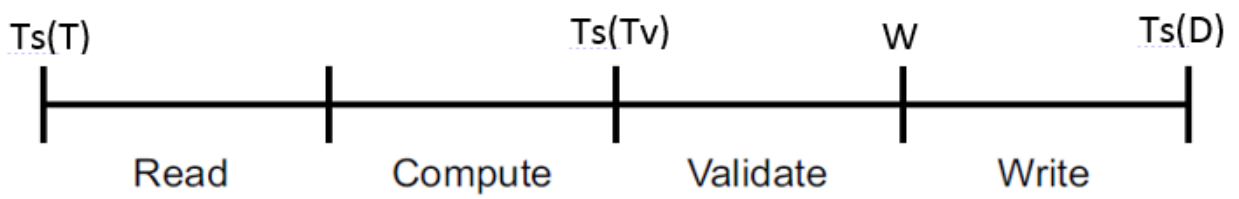


Figure 3.2: transactions at various stages of the Proposed System.

At validation phase: Timestamp $Ts(T_v)$ is allocated to the active transaction, Any transaction whose timestamp $Ts(T)$ becomes empty must be aborted, however, transaction has to be validated, to check if the copied data-item timestamp $Ts(D)$ in the private workspace is compatible with the $Ts(D)$ in the database. If their compatibility is false, then the transaction will abort and its private workspace will be cleared. However, if otherwise, the transaction will commit and enters the write phase.

In Write Phase: Data-item values are copied from the private workspace to the database, once a transaction go into the write phase, the transaction is considered to be complete. In the write phase, all the updates are permanent in the database. However, our proposed algorithm is not exactly the same with the timestamp based optimistic concurrency control protocol in (Karami and Dastjerdi, 2011) because some conflict checking is done in the read phase. The reason is to detect and restart non-serializable executions earlier. This will prevent unnecessary execution of the non-serializable transactions to its validation phase and ensures efficient serializable execution to reduce the wasting time.

3.2.1 Advantages of the Proposed System

1. It ensures that transactions that arrived too late and long-running transaction are not rejected
2. It ensures executions that is equivalent to that of serial execution (serializable execution/interleaving transactions), which reduces the rate transaction restarts and enhance system throughput.
3. It ensures isolation property of simultaneous transactions and increase transaction response time.
4. It ensures that the execution is free from deadlock and avoids cascading rollback.
5. Reads operations are completely unrestricted, since reading a data-item value can never cause loss of integrity.

The communication scenario throughout the execution of transactions between the components of the proposed system is demonstrated in Figure 3.3: Transactions communicate with TMs, TMs communicate with SC-OPTHs, SC-OPTHs communicate with DMs, and DMs manage the store data. TMs do not communicate with other TMs, neither do SC-OPTHs communicate with other SC-OPTHs.

Transactions are supervised by the TMs, which means, the transaction issues all of its database operations to that TM.

3.3 Methodology of the Proposed System

A system development methodology (SDM) refers to the step-by-step procedure using to construct, design, and control the process of developing an information system. We adopted the object Oriented Analysis and Design Methodology in the analysis of effective technique for optimizing timestamp ordering scheduler in RW/WW synchronization. It features algorithm, user case diagram, activity diagram, and architectural design.

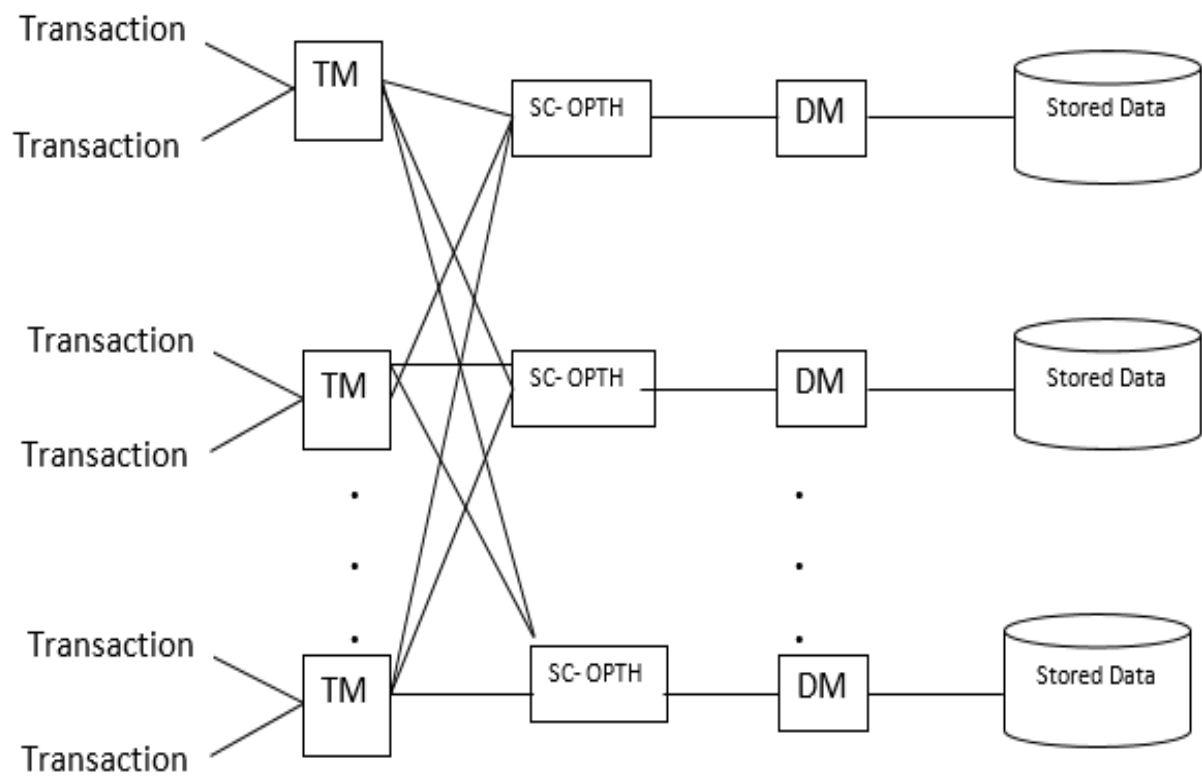


Figure 3.3: Architecture of the Proposed System.

3.3.1 The High Level Model Diagram of the Proposed System

The High Level Model Language consist of Source (Transactions generator), Transactions Manager, Scheduler (OPTH), Data Manager and Data processor. The components are represented in Figure 3.4. The source produces transactions in the system at a specified frequency, the transaction manager guarantees the coordination of transaction execution which contains the local node and, possibly, some remote nodes, the monitor displays statistics related with response time, service rate, and restart percentage, and finally, the scheduler contains the Optimistic Thomas Algorithm (OPTH) which is a hybrid of Optimistic Concurrency Control and Thomas' Write Rule Timestamp Ordering Algorithms. The scheduler controls the Relative order in which database and transaction operation are executed, (Hassan, 2008).

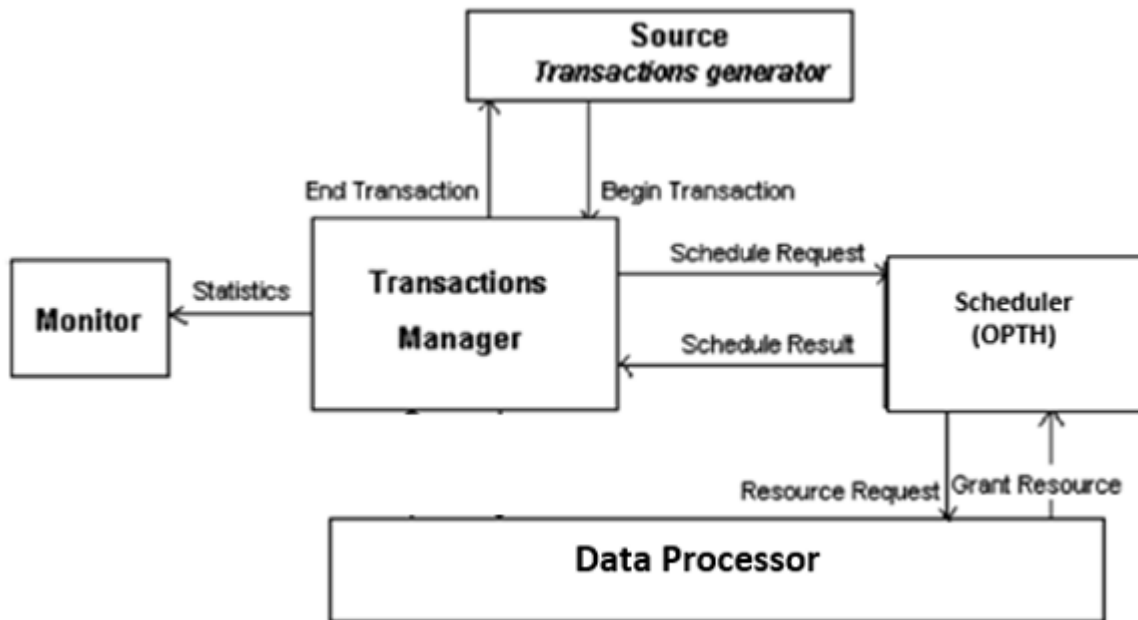


Figure 3.4: High Level Model Diagram of the Proposed System.

3.3.2 The Use Case Diagram of the Proposed System.

A Use Case diagram is a graphical representation showing the functionalities of the system. It describes the interaction between the users (customer/teller) and the system. It describes what the system does from the standpoint of an external observer. In this circumstance, the term "system" talk about to something being developed or functioned, such as an online banking system and service Website. Use case diagrams are employed in UML (Unified Modeling Language), a standard representation for modeling real world objects and systems. The Use Case diagram of the proposed system is shown in figure 3.5

A transaction: is a set of instructions from the user (customer/teller) program that consists a sequence of read/write operations on database, such as withdraw, deposit, and transfer in current account.

Transaction Manager: (TM): The TM gives tracts of transactions submitted by the users with an identifier (TrId). It then contacts the SC-OPTH that controls access to DM requested for each operation in the database system.

SC-OPTH: is an algorithm deployed to control database conflicts which ensure serialization of interleaving transactions and help to sustain database consistency. It makes use of this control by restricting the order in which the data manager (DM) executes the reads, writes, commits, and aborts of different transactions in the database. To satisfy this condition, the SC-OPTH pass some operations in a direct

manner to the DM and wait for result if it does not find any conflicts. But if there is a risk of conflict, it delays or terminate certain operations.

Data Manager (DM): gives access to the database and returns the value from the database. For a write operation the DM update the value and returns an acknowledgment to the SC-OPTH that will, in turn, transmit the result to the user that submitted the transaction through the TM.

Data-item: A database is a collection of entities and each entity has a single values for operation

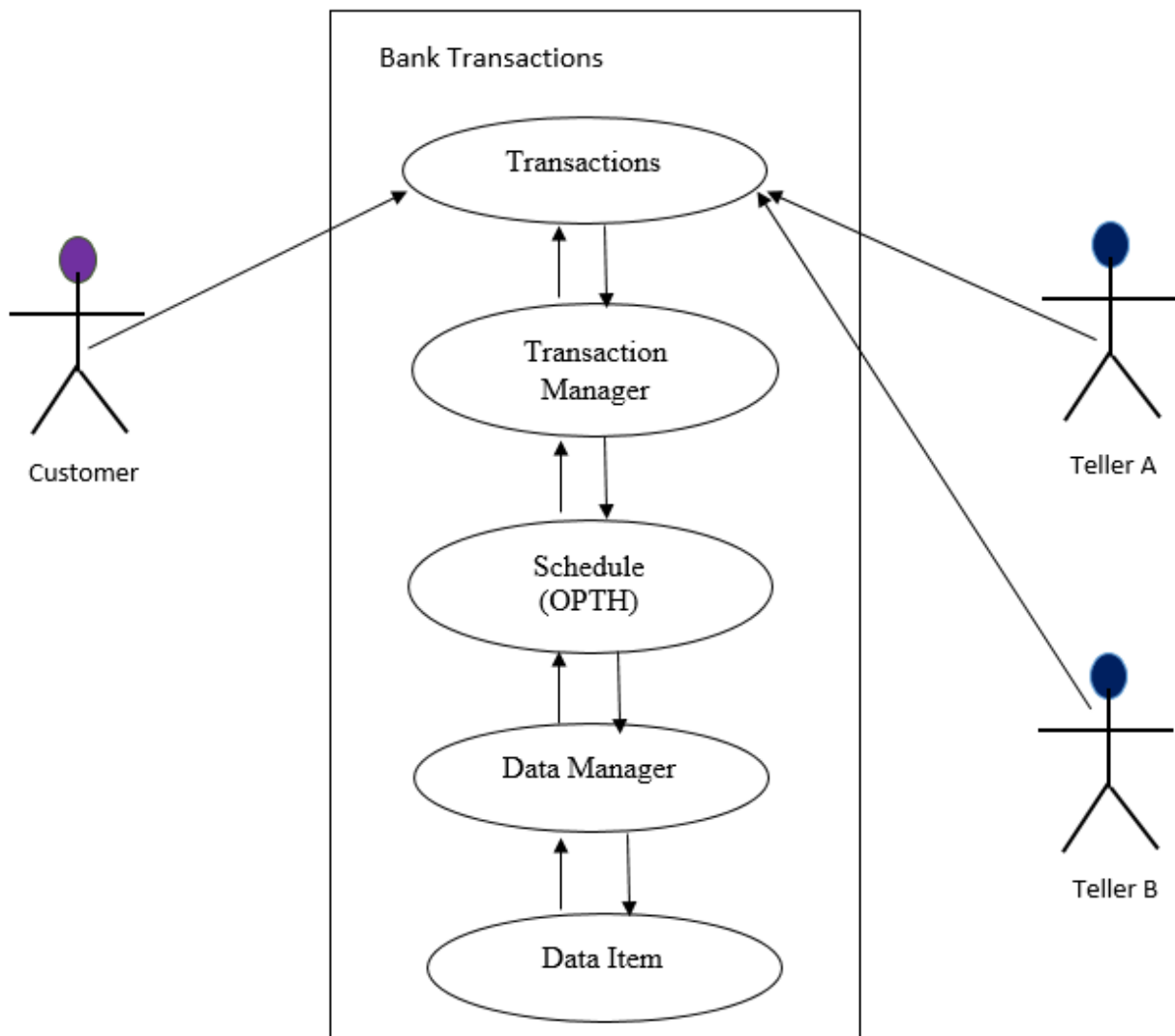


Figure 3.5: Use Case Diagram of the Proposed System.

3.3.3 The Activity Diagram of the Proposed System

The activity diagram in Figure 3.6 show the connectivity of the proposed system. To provide efficient concurrency control the proposed system integrates both Thomas' Write Rule Timestamp Algorithm and Optimistic Concurrent Control Algorithm. At the client phase, transactions are set of instructions from user programs that consists a sequence of read/write operations on database, such as check account balance, withdraw fund, deposit fund, transfer fund etc. and are positioned in remote sites in a distributed database system.

Thomas' Write Rule Timestamp Ordering Algorithm is applied in read phase, where every transaction is given a timestamp $Ts(T)$ at the start of execution and is compared against the database timestamp $Ts(D)$ which is the last timestamp a transaction is performed successfully on the database. i.e, if $Ts(T) \Rightarrow Ts(D)$ is True, the transaction read the values of database and make copy to a private workspace, do the modifications, makes local copies of its modifications in the private workspace only and not to the database.

Optimistic concurrent control algorithm is applied at the validation phase, where transaction is validated. Checks if the copy $Ts(D)$ in the private workspace is the same with the $Ts(D)$ in the database, if their compatibility is false, then the transaction will abort and its private workspace will also be cleared. However, if otherwise, new Timestamp $Ts(Tv)$ is given to the active transaction, it goes into the

write phase where the modified values in the private workspace updates the database, once a transaction is in the write phase, it is considered to be complete. In the write phase, transaction makes all its updates permanent in the database.

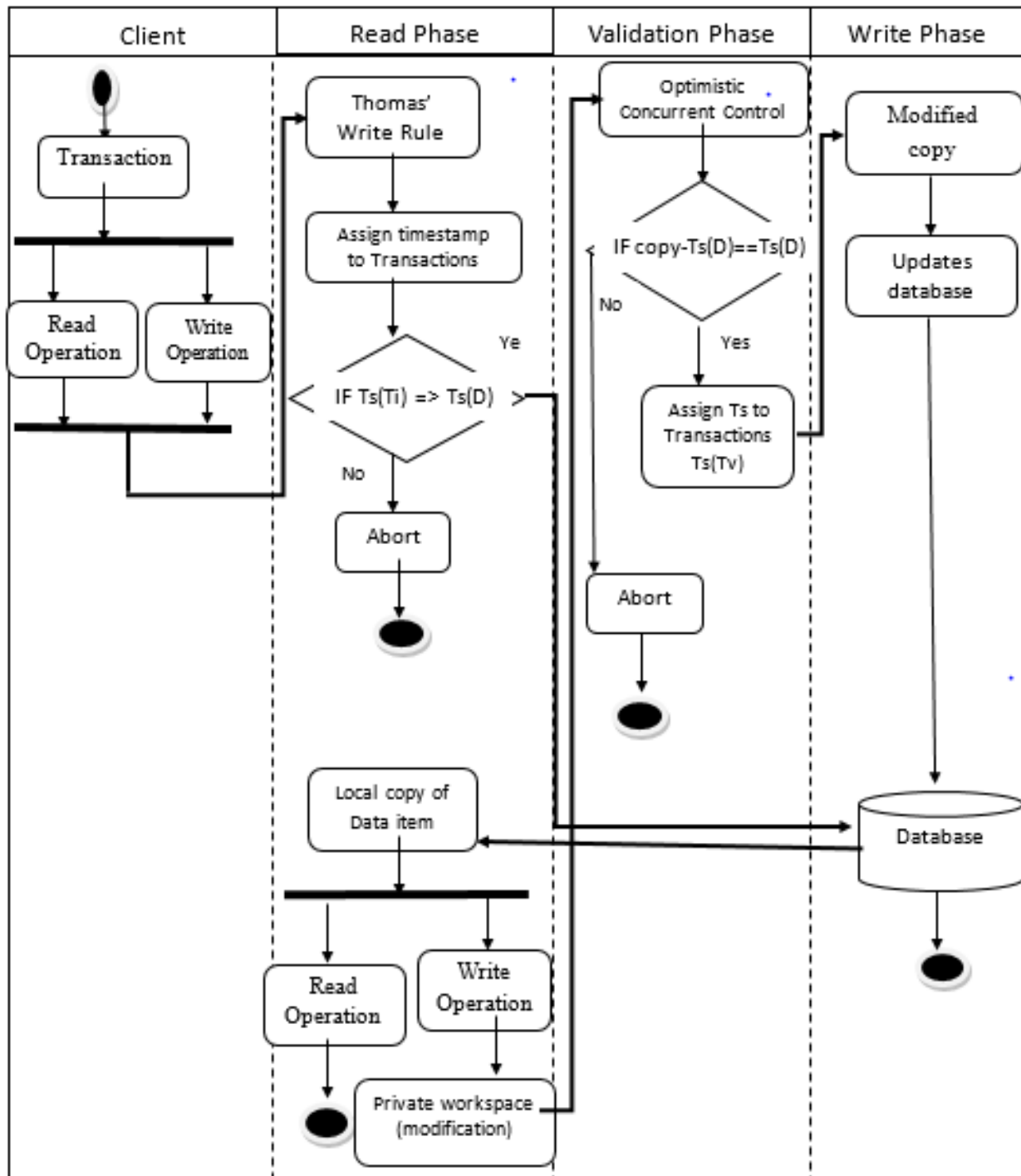


Figure 3.6: Activity Diagram of the proposed system.

3.3.4 Transaction Execution Model of Proposed Algorithm

A schedule is defined over a set of operations (read, write, abort, or commit) from a list of transactions ($T_1, T_2, T_3, \dots, T_n$), and the sequence of execution in which the operations are in the schedule must be the same in which they are executed in the transaction. Two operations $T_1(X)$ and $T_2(X)$ accessing the same data-item X , and are to be in conflict if at least one of them performed a write operation.

Note. First, read operations don't conflict with each other, our focus is on Read-Write and Write-Write operations. Second, the operations are from two different transactions. Intuitively, the two transactions are said to be conflicting. That means, the presence of conflict between the operations signified their order of execution.

In our proposed algorithm, the list of active transactions $T_1, T_2, T_3, \dots, T_n$, with timestamps are executed and sent to database for commitment. We will prove that only if scheduler involve execute and completed transactions are preserved serializability theory, then the transaction will be committed otherwise, transactions will abort.

Suppose T_1 is the last transaction with the timestamp $Ts(T_1)$ which successfully performs an operation over X data-item. This way, timestamp for X data would be $Ts(X) = Ts(T_1)$.

For transaction T_2 which is sent to database after T_1 gets access to X data-item, the resulting conditions are possible: The first situation: If T_2 reads from X data-item at

the timestamp of $R-Ts(T2)$, here the operation ought to be considered as read-write conflict. If there is

$Ts(X) \leq R-Ts(T2)$, the reading will be accepted. Therefore:

$$Ts(X) \leq R-Ts(T2) \implies W-Ts(T1) \leq R-Ts(T2).$$

That is, in serializability graph, can be an edge like

$$T1 \rightarrow T2. \dots\dots\dots (1)$$

The second situation:

If $T2$ writes to X data at the timestamp of $W-Ts(T2)$, then this operation ought to be considered as writing-writing conflict, only if the condition of

$Ts(X) \leq W-Ts(T2)$, then this kind of writing is acceptable. So,

$$Ts(X) \leq W-Ts(T2) \implies W-Ts(T1) \leq W-Ts(T2).$$

In this case, in serializability graph, can be an edge like

$$T1 \rightarrow T2 \dots\dots\dots (2)$$

Results of (1) and (2) make clear that, in all circumstances between $T1$ and $T2$ in serializability graph there is an edge, and the direction of edge would be $T1 \rightarrow T2$.

We supposed that $T2$ is committed after $T1$, so the direction of edge in the proposed algorithm is serializability graph represents the commitment ordering of transactions.

$$\text{That is: } T1 \rightarrow T2 \rightarrow \text{Commit } (T1) < \text{Commit } (T2) \dots\dots\dots (3)$$

Theorem: serializability theory of scheduler is well-preserved by OPTH algorithm.

To verify this theorem, we show that serializability graph centred on OPTH algorithm doesn't result loop. This issue will be permitted by contradiction proof.

Contradiction proof take responsibility that in serializability graph, there is a loop like the following:

$$T1 \rightarrow \dots \rightarrow T2 \rightarrow T3 \rightarrow T1$$

In this way, according to result (3):

$$\text{Commit}(T1) < \text{Commit}(T3) < \text{Commit}(T2) < \dots < \text{Commit}(T1)$$

Therefore, because $\text{Commit}(T1)$ can't be smaller than $\text{Commit}(T1)$, this condition is impossible. So the assumption is wrong and the theorem is proved.

3.3.5 The Optimistic Thomas Algorithm (OPTH)

If transaction T_i issues Read Operation $R(D)$.

- 1: if ($Ts(T_i) \Rightarrow Ts(D)$) {
- 2: Execute T_i
- 3: Read values of data-Item
- 4: }

If transaction T_i issues Write Operation $W(D)$.

- 1: if ($Ts(T_i) \Rightarrow Ts(D)$) {
- 2: Execute T_i
- 3: Copy values of data-Item to a private workspace place,
- 4: Do modifications in the private workspace and not in the database
- 5: Go to the write phase
- 6: }
- 7: if ($Copy-Ts(D) == Ts(D)$) {
- 8: Active Transaction is assigned a new Timestamp $Ts(T_v)$
- 9: Write operation is executed
- 10: Copies the modified values to database (Update)
- 11: and sets $Ts(D) = Ts(T_v)$
- 12: }
- 13: else {
- 14: transaction is abort and rollback, its private workspace is also clear
- 15: }

Figure 3.7 demonstrates how the proposed system handles transactions in an online banking system that permits multiple users, where Customer1 and Customer2 simultaneously withdrawing money, 4000 and 3000 respectively in the same Account (current Acct), and the current Acct balance is 5000 having a timestamp of 2:00Am, which is last timestamp a transaction is performed successfully on the current account.

In the Read Phase: Customer1 starts a transaction and is assigned a timestamp of 7:00Am and Customer2 also starts a transaction and is assigned a timestamp of 7:01Am. Each of their transactions both checks the balance of the current Account and make a copy in a private work place, does the modifications, and also makes local copies of its modifications in the private workspace only and not to the current account in the database.

Note: At this stage any customer that finishes its computation first enters the validation phase.

At the validation phase: It is validated by checking if the timestamp of copied account in the private workspace is the same with the timestamp of current Account in the database, if False, means an earlier transaction has already written on the database outside the active timestamp, then the transaction must abort and is

rollback. Else new timestamp is assigned to transaction $Ts(T_v)$ and transaction will verified and enter the write phase.

In the write phase: The database is updated by copying the modified Account in the private workspace to the current account in the database. At this stage, transaction makes all its updates permanent in the database.

Figure 3.8: Bank Read and Write operations describes the activities of the banks with respect to this work. The receiver's teller A writes the account of the customer, the same account may be writing and reading by "B" receivers' teller at the same time. Concurrent should occur as another payee teller is at the same time writing and debiting the account of the same user. All the transactions will be saved in the audit trail to measure the accuracy of the system.

Phases	Customer1	Customer2
Read Phase	<p>7:00Am Check Acct balance and make a copy in a private work place</p> <div>Copied Acct (2:00Am) 5000</div>	<p>7:01Am Check Acct balance and make a copy in a private work place</p> <div>Copied Acct (2:00Am) 5000</div>
Computation	<p>Withdraw 4000</p> <div> <div>Copied Acct (2:00Am) 5000</div> <div>+</div> <div>Modify Acct 5000 – 4000 => 1000</div> </div>	<p>Withdraw 2000</p> <div> <div>Copied Acct (2:00Am) 5000</div> <div>+</div> <div>Modify Acct 5000 – 2000 => 3000</div> </div>
Validation Phase	<p>7:10Am Checks if timestamp of the copied account in the private work place is the same with the timestamp of the current Acct in the database.</p> <div> <div>Copied Acct (2:00Am) 5000</div> <div>≠</div> <div> <div>Current Acct (7:05Am) 3000</div> <div>Abort</div> </div> </div>	<p>7:05Am Checks if timestamp of the copied account in the private work place is the same with the timestamp of the current Acct in the database.</p> <div> <div>Copied Acct (2:00Am) 5000</div> <div>=</div> <div> <div>Current Acct (2:00Am) 5000</div> <div>Commit</div> </div> </div>
Write Phase		<p>The database is updated by copying the modified Acct in the private work place to the current account in the database</p> <div> <div>Modify Acct 5000 – 2000 => 3000 (7:05Am)</div> <div>=></div> <div> <div>Current Acct 3000 (7:05Am)</div> </div> </div>

Figure 3.7: Illustration of Proposed System in an Online Banking System.

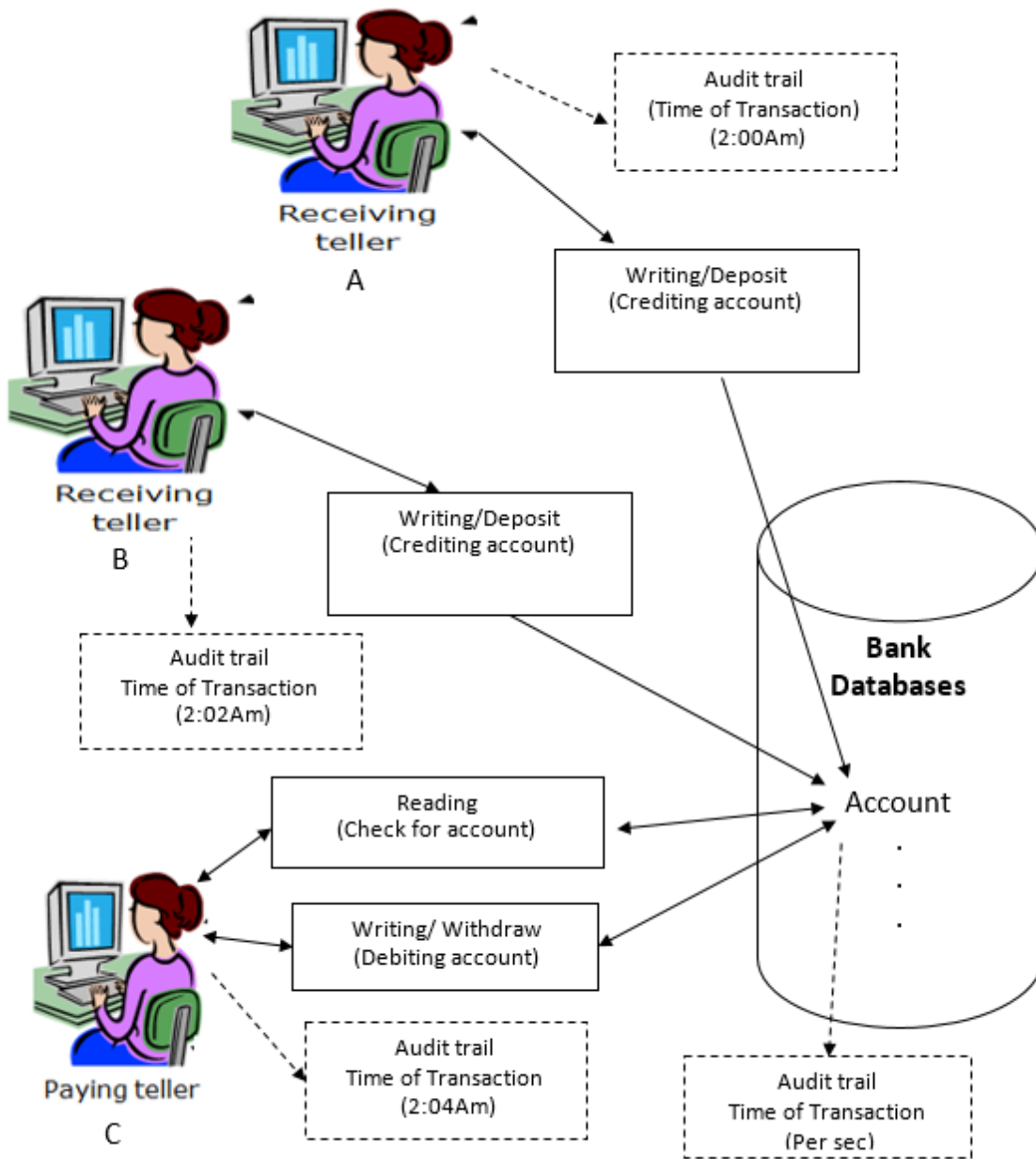


Figure 3.8: Bank Read and Write operations.

3.4 Output/Input Specification

3.4.1 Output/Input Design

Input design for this work consist of Customers account details such as Account Name, Amount, Account Types, Account Number and Branch. And the Output design summaries Write/Read Output information for online banking system. The Output/Input specification are summarized in tables below.

3.5 Database Design

Designing a distributed computing database system involves taking decision on the location of data and programs in a computer network nodes, and network design itself. In the case of distributed database, based on the assumption that the network has been designed already and there is a replica of the DBMS software on each node in the network where data are stored, it remains to focus our attention on the distribution of data. The database consists of a set of records where each of them is defined over a set of attributes such as Field Name, Data Type and Length. The data inputted into the design database is made accessible to all other station. The database tables are made known below.

Table 3.1: Admin Table


	Name	Data Type	Allow Nulls
	Admin_id	int	<input type="checkbox"/>
	names	varchar(30)	<input checked="" type="checkbox"/>
	username	varchar(15)	<input checked="" type="checkbox"/>
	password	varchar(15)	<input checked="" type="checkbox"/>

Table 3.2: Transaction Table



	Name	Data Type	Allow Nulls
	trans_id	int	<input type="checkbox"/>
	acctNo	varchar(15)	<input checked="" type="checkbox"/>
	date	varchar(15)	<input checked="" type="checkbox"/>
	transacType	varchar(15)	<input checked="" type="checkbox"/>
	names	varchar(30)	<input checked="" type="checkbox"/>
	tranf_amt	varchar(15)	<input checked="" type="checkbox"/>
	balance	float	<input checked="" type="checkbox"/>

Table 3.3: Customer Table

	Name	Data Type	Allow Nulls
	cust_id	int	<input type="checkbox"/>
	names	varchar(30)	<input checked="" type="checkbox"/>
	phone	varchar(15)	<input checked="" type="checkbox"/>
	address	varchar(50)	<input checked="" type="checkbox"/>
	gender	varchar(6)	<input checked="" type="checkbox"/>
	age	varchar(3)	<input checked="" type="checkbox"/>
	state	varchar(15)	<input checked="" type="checkbox"/>
	marital	varchar(15)	<input checked="" type="checkbox"/>
	acctNo	varchar(15)	<input checked="" type="checkbox"/>
	password	varchar(10)	<input checked="" type="checkbox"/>
	balance	float	<input type="checkbox"/>
	acct_time	varchar(15)	<input checked="" type="checkbox"/>
	imagename	varchar(15)	<input checked="" type="checkbox"/>

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 System Requirements

The objective of preparing the system requirements is to provide better information for the users of this system in such a manner that ultimately leads to successful software implementation and better results. The basic requirements for the designed model (online banking system, using optimistic concurrency control and Thomas write rule techniques to synchronization RW/WW) needed these Hardware and Software requirements to enable it work effectively in a computer system

4.1.1 Hardware Requirements

During testing the designed model, the computer system needs physical resources for it to effectively handle the model, the basic required resources includes the following

1. **Processor:** Pentium-IV
2. **Random Access Memory (RAM):** 1 Gigabytes
3. **Hard disk:** 10 Gigabytes

4.1.2 Software Requirements

The computer system needs software resources to be installed to enable the model function effectively, the basic required resources includes the following:

1. **Operating System:** Windows 7 (32 bit or 64 bit)
2. **Integrated Development Environment (IDE):** Visual Studio 2010
3. **Database:** MS SQL Server 2008
4. **Web Server:** Internet Information Services (IIS),
5. **Web Browsers:** Internet Explorer7, Firefox2, Opera9, and Google Chrome.

4.2 System Testing

Software testing is a process of running with intent of finding errors in software. Software testing confirm the excellence of software and signifies final review of other phases of software like specification, design, code generation etc.

4.2.1 Unit Testing

Unit testing give emphasis to the verification strength on the tiniest unit of software design i.e. a software module or component. Unit testing is a dynamic technique for verification, where programs are correctly compiled and executed. Unit testing is

done in parallel with the coding phase. Unit testing checks each modules or units not the entire software.

We have tested each module/component of the application independently. As the modules were built-up, testing was carried out instantaneously, tracing out each and every single kind of input and testing the corresponding output until module is working properly. In each module all the functionalities were tested in isolation.

The unit testing is done using Visual Studio IDE without the need of any external application. Visual Studio IDE has in-built support for testing the application. Various techniques have been made for the purpose of unit testing. Test cases are automatically generated for these methods. The tests run under the ASP.net using C# and Ms SQL server context which means settings from Web Configured files are automatically picked up once the test case starts running. Methods were written to reclaim all the manufacturers from the database, strings that is equal to a certain search term, products that is equal to certain filter criteria, all images that belong to a particular product etc. Unit test cases were automatically generated for these methods and it can be seen that the tests have passed.

4.3 Choice of Programming Language Used

Program is a set of instruction written by a programmer in an actual programming language, which enables particular processes to be executed by a computer.

The choice of programming language in this dissertation was C# (pronounced C Sharp). C# is a programming language from Microsoft designed specifically to target the .NET Framework. Microsoft's .NET Framework is a runtime environment and class library that dramatically simplifies the development and deployment of modern component-based applications. This language is chosen because of its object oriented nature, and object oriented programming has been a great success. The fundamental principles behind it, inheritance, encapsulation, and polymorphism, have allowed us to cleanly encapsulate related application data and logic, making code maintenance easier. Microsoft Visual Studio is a development tool from Microsoft, it is used to console and graphical applications along with Windows form application, Web Sites, Web applications and Web Services in both native code together with managed codes for every platforms supported by Microsoft windows. Microsoft Visual Studio was chosen as development environment, mainly because web developers can develop very compelling applications using ASP.NET, with the simplicity of drag and drop server controls.

4.4 Presentation of Data

The system was designed using C#, ASP.net, Css3, JavaScript and Ms SQL server. The proposed system is experimented using online banking system. Which consist of Admin/teller and Customer Panel. Checking customers account details represents the Read operation while Withdrawal and Deposit customers account represent the writing operations. The interactive interfaces for RW/WW synchronization is shown below. The figures 4.1(a) and 4.1(b) are login pages for customers and teller respectively. The figures 4.2(a) and 4.2(b) allows tellers and customers to initiate transaction. The teller input account number in the textbox to get access to the customers' account. This represents the executions of read Operations.



Figure 4.1(a): Customer Login Page.

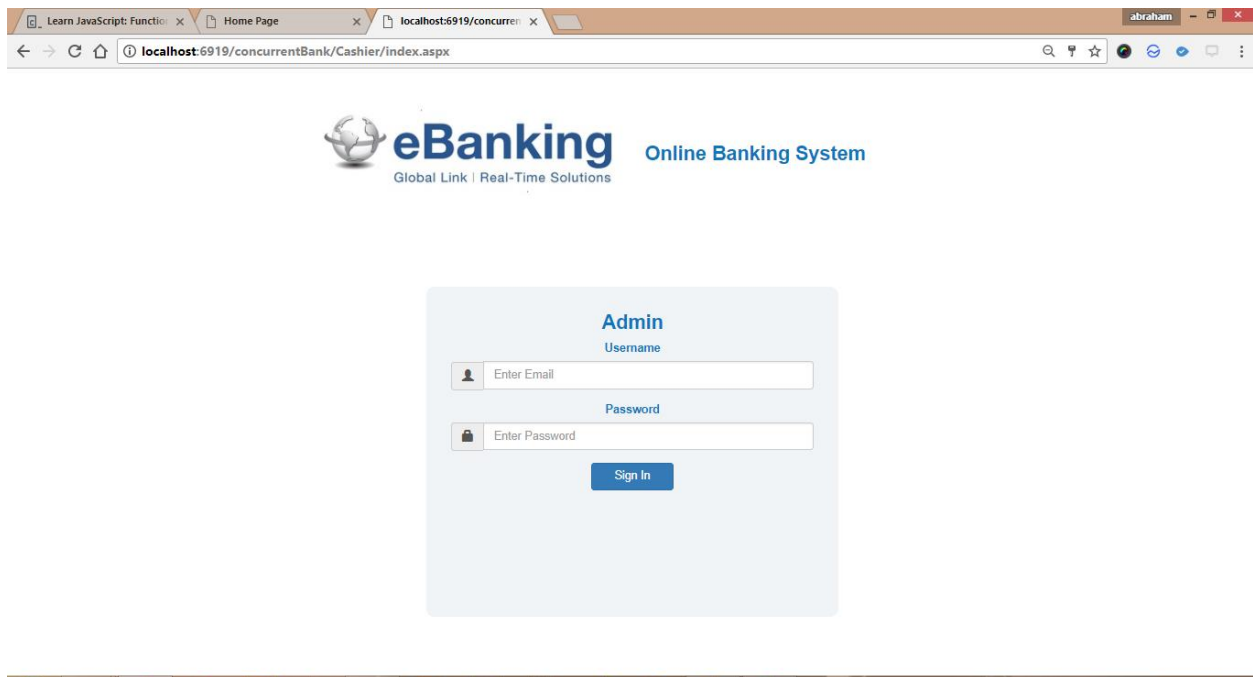


Figure 4.1(b): Teller Login Page.

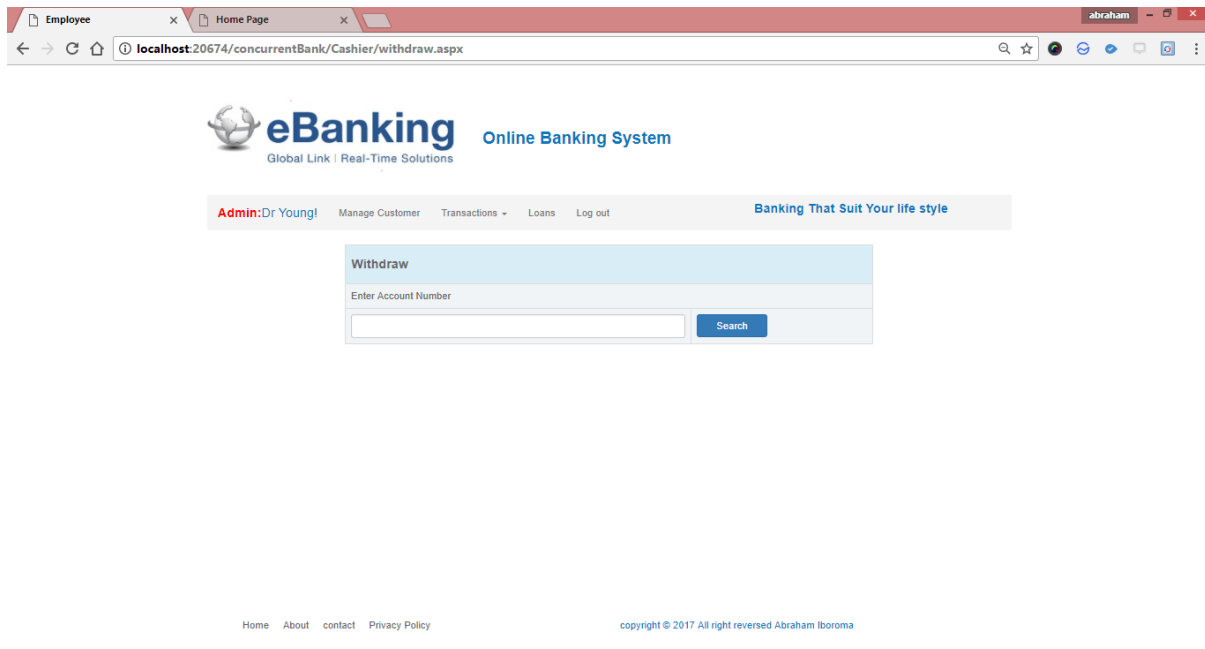


Figure 4.2(a): Teller initiate transaction

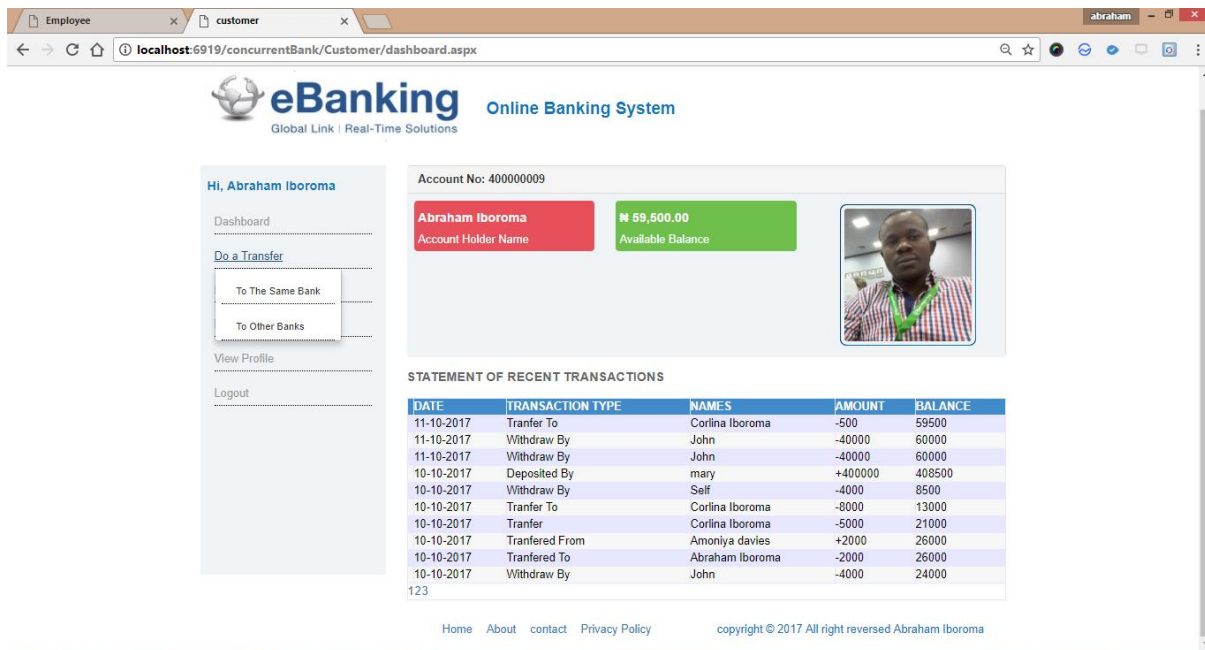


Figure 4.2(b): Customer initiate transaction

When two users (teller and customer) simultaneously withdraws money in the same account, without any concurrency strategy in place, the user who commits her withdrawal last, override the withdrawal made by the first user.

The proposed system ensures that the co-ordination of the users transactions are serializable execution and maintain the consistency of the database. Write-write operations were performed simultaneously on customer's savings account (400000009) by teller and customer respectively. The customer initiate a transaction with the timestamp "04/10/2017 10:25:14 AM" by transferring #5000 from his own savings account to a different savings account. While the teller initiate its transaction with the timestamp "04/10/2017 10:25:19 AM" by Withdrawing #8000.00 from the same savings account in remote location. The analysis to verify the productivity of the proposed system are made known from Figures 4.3 to Figure 4.6.

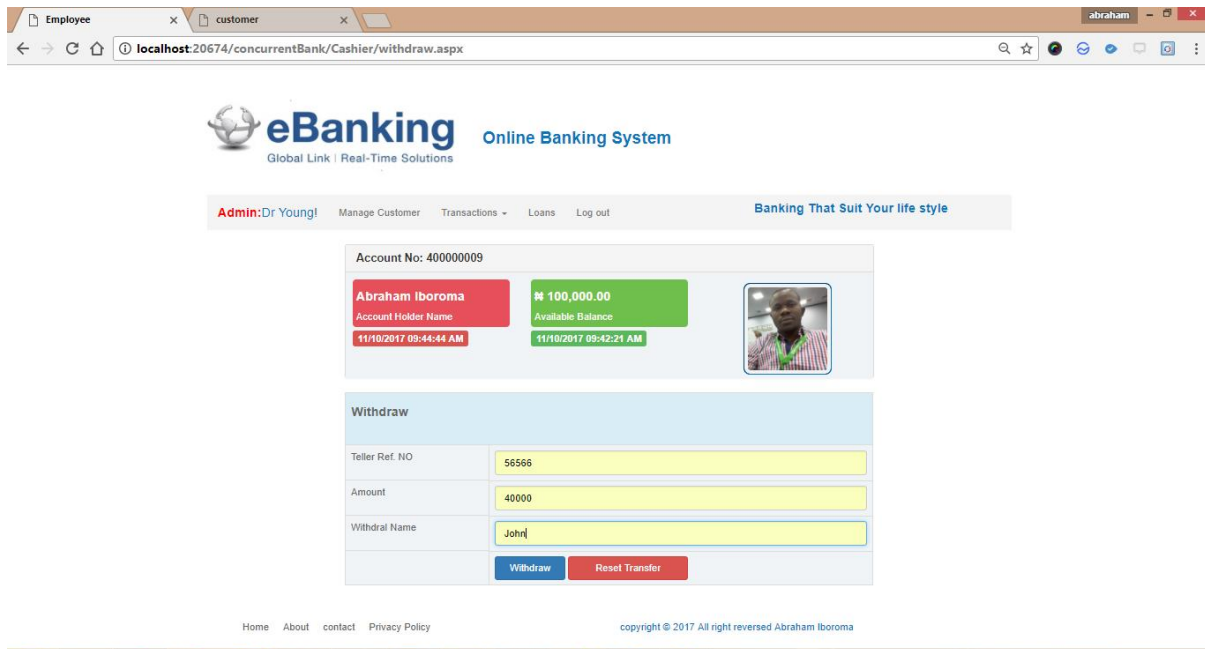


Figure 4.3(a): Teller Withdrawing from Customers Savings Account

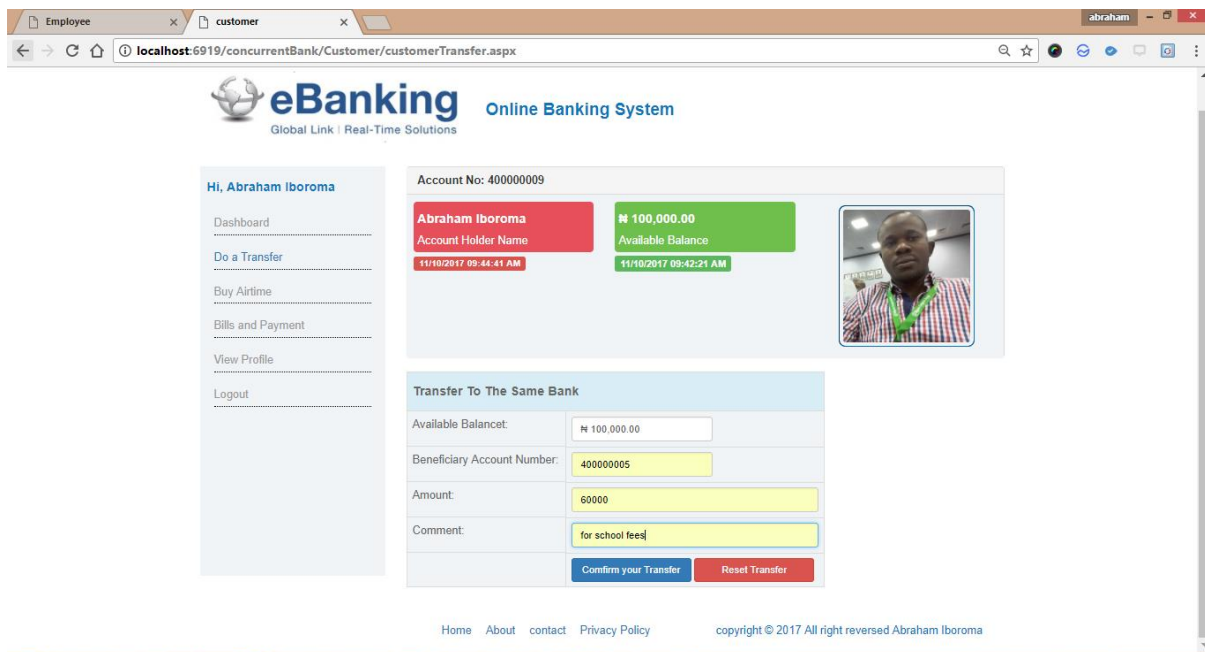


Figure 4.3(b): Customer Transferring From Savings Account

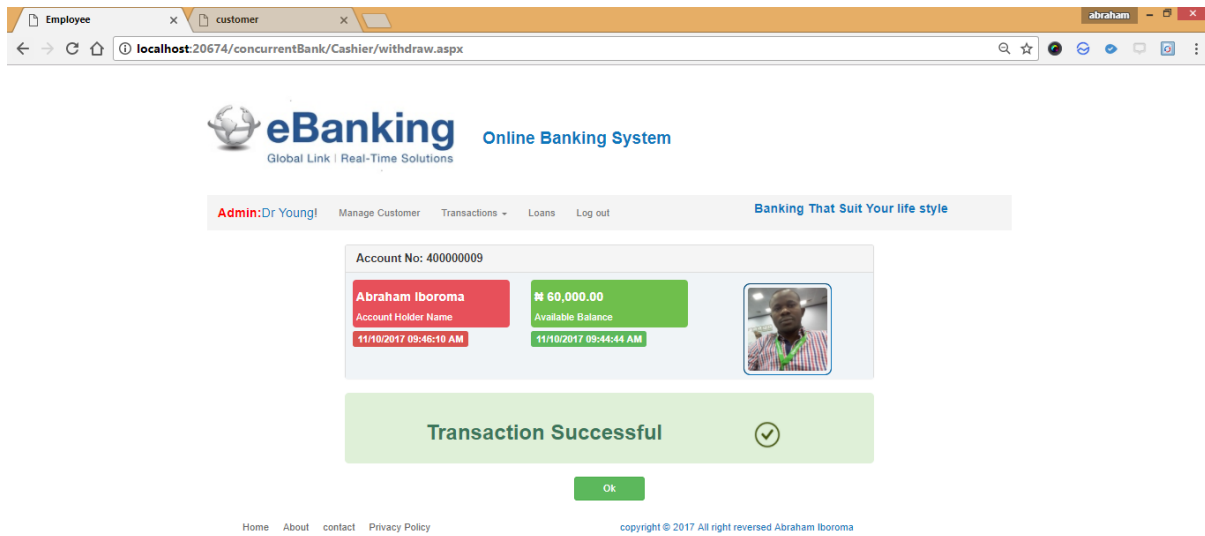


Figure 4.4: Teller Complete Transaction

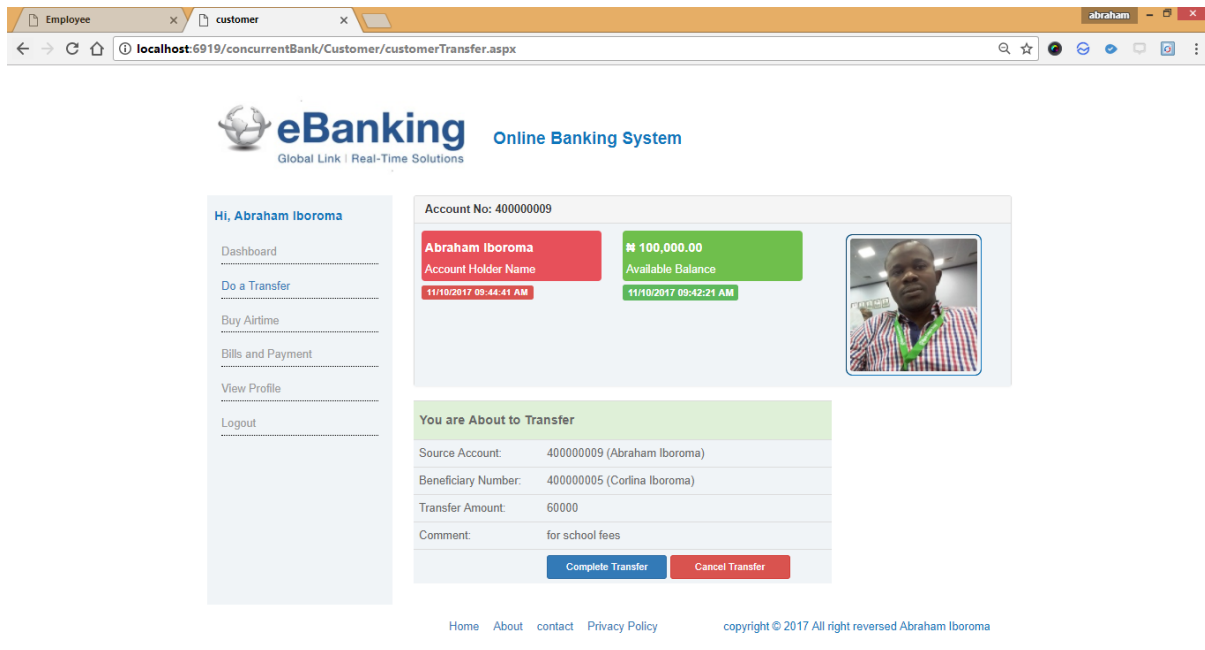


Figure 4.5: Validating Transaction

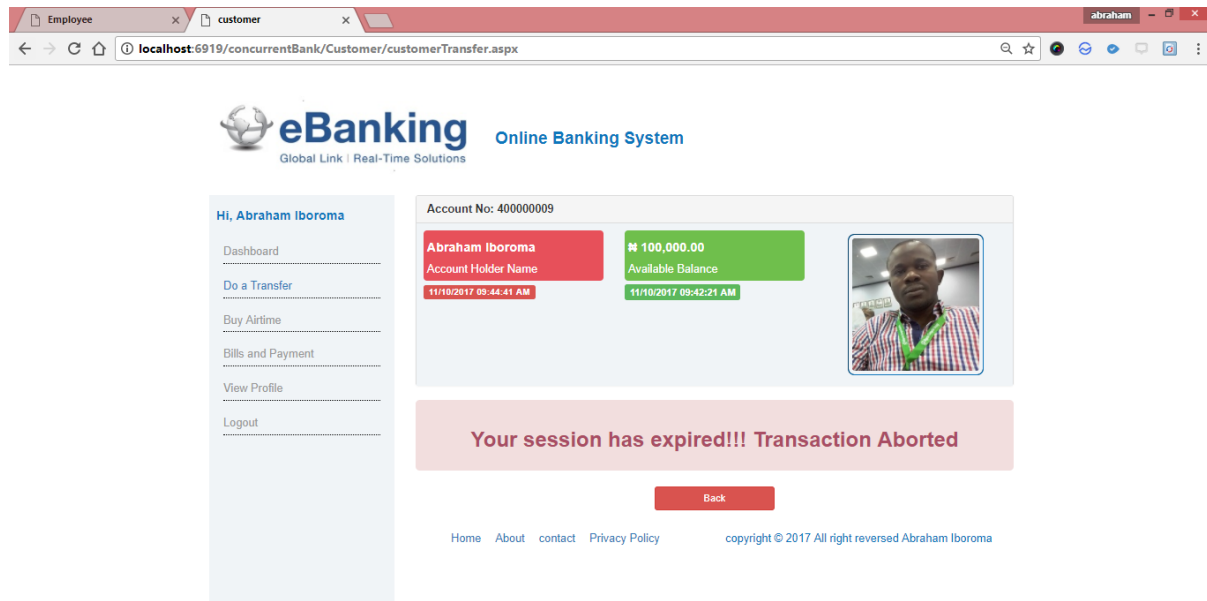


Figure 4.6: Conflict is Detected Transaction Terminate

4.6 Result and Discussion

Two transactions that contain Read and Write operations, were performed, the operations from these transactions in Figures 4.3(a) and 4.3(b) start execution almost at the same time in seconds. The timestamp for both transactions are 04/10/2017 10:25:14 AM and 04/10/2017 10:25:19 AM. The system enables the transactions to read the customer's savings account and make copy to a private workspace place where it does the modifications and makes local copies of its modifications in the private workspace only and not to the customer account in the database. This is done in the Read phase of the proposed system as shown in Figure 3.7. The conflict check is done in the validation phase, new Timestamp $T_s(T_v)$ is assigned to the active transaction. If the timestamp of the copied savings account in the private workspace is incompatible with the timestamp of the savings account in the database, it means an earlier transaction outside the active transaction has written on the database, transaction must then be aborted and rollback. Else, if compatibility is true, then the active transaction go into the write phase where the modified account in the private workspace updates the original account in the database. Once transaction is in the write phase, it is considered to be complete. In the write phase, transaction makes all its updates permanent in the database. This is also valid to the rest of the simulations. The experimental results shown the significant performance of the

proposed system with reference to system throughput, and response time as the Read-Write operations Validate approach is deployed in the database system.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Summary

The proposed approach for this dissertation is transaction processing model for concurrency control in distributed database systems. We present two main concept, first, a model that make available common terms and ideas used in a several concurrency control algorithms, second, a problem disintegration that break down concurrency control algorithms into read-write/write-write synchronization sub-algorithms. Generally, most concurrency control techniques are variations of two basic algorithms, which are two-phase locking (2PL) and timestamp ordering algorithms. We have explained these two algorithms and also several concurrency control algorithms earlier in the literature review. We also made known, how to integrate our proposed algorithms to form complete concurrency control algorithm. The Optimistic Concurrency Control Algorithm and Thomas' Write Rule Timestamp Algorithm is integrated for form Optimistic Thomas Algorithm (OPTH). Our focus in this project is basically the structure and accuracy of synchronization and concurrency control algorithms. The performance of concurrency control algorithms influence the system throughput and transaction response time. The influence on system throughput and response time varies from algorithm to algorithm, system to system, and application to application. The performance

analysis of the algorithm remains and will be resolve in our future work. Our expectations and recommendation that future work on concurrency control algorithms in distributed-database system will concentrate on the performance.

5.2 Conclusions

The research work aimed to develop an efficient Timestamp Ordering algorithm and model in conflicting operations like rw/ww synchronization. It proposed an Optimistic Thomas Algorithm (OPTH) which is a hybrid algorithm integrating Optimistic Concurrency Control Algorithm and Thomas' Write Rule Timestamp Algorithm for optimizing the serialization of RW/WW synchronization in DDBMS. The simulation results shows that the system is efficient and provide an execution that has the same effect as a serial execution. It reduces the rate transactions restarts and improve transaction throughput in a distributed database system.

5.3 Recommendation

It is recommended that the hybridization of other pessimistic concurrent control algorithms and optimistic concurrent control algorithms should be used to address the problems of conflicting transactions in a centralized and distributed database system.

5.4 Contribution to Knowledge

1. The project work developed an efficient algorithm to optimize the serialization of RW/WW synchronization in DDBMS using Optimistic Concurrency Control Algorithm and Thomas' Write Rule Timestamp Algorithm
2. The integration of both approaches is beneficial and fixed the problem of long executing transactions and transactions that arrived too late. It reduces the risk of conflict and improves system throughput.

REFERENCES

- Arun, K.Y., and Ajay, A. (2010): A Distributed Architecture for Transactions Synchronization in Distributed Database Systems, International Journal on Computer Science and Engineering, Vol. 02, No. 06, Pp. 1984-1987.
- Bernstein, P., Hadzilacos, V., and Goodman, N (1987). Concurrency Control and Recovery in Database Systems. Addison Wesley, Reading, MA. Pp. 1-17.
- Chris, B., and Pete, F. (2004). Time and date issues in forensic computing a case study, Digital Investigation, vol. 1, pp. 18-23.
- Wette, C., Pierre, S., and Conan, J. (2004). A Comparative Study of Some Concurrency Control Algorithms for Cluster-Based Communication Networks, Computers and Electrical Engineering, Vol. 30, Issue 8, Pp. 616-618.
- Hassan, M. N. (2008). A New Approach for Recoverable Timestamp Ordering Schedule, World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering Vol. 2, No. 1, Pp.
- Habes, A., and Hasan, R. (2015). Multiversion Thomas' Write Rule Timestamp based Concurrency Control, International Journal of Computer Science Issues, Vol. 12, Issue 3, Pp. 202-204.

- Jaypalsinh, A.G., and Prashant, M.D. (2016). Study and Comparative Analysis of Basic Pessimistic and Optimistic Concurrency Control Methods for Database Management System, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 5, Issue 1, Pp. 178-185.
- Kamal, S. (2014). Transactional Concurrency Control for Resource Constrained Applications, School of Computing Science, Newcastle University. <http://hdl.handle.net/10443/2553>, Pp. 9-13.
- Karami, A., and Dastjerdi, B.A. (2011). A Concurrent Control Method Based on Commitment Ordering in Mobile Databases, International Journal of Database Management System, Vol. 3, No. 4, Pp. 39-42.
- Quazi, E.K., Mamun., and Hidenori, N. (2005), Timestamp Based Optimistic Concurrency Control. Conference Paper December. Institute of Electrical and Electronics Engineers.
- Raghu, R., and Johannes, G. (2003). Database Management Systems, Third Edition, Pp. 523-529.
- Ramez, E., and Shamkant, B.N. (2011). Fundamentals of Database Systems, sixth edition, Pp. 743-791.
- Rinki, C., Suman, M., Rathy, R.K., and Preeti, B. (2011). Recoverable Timestamping Approach for Concurrency Control in Distributed Database, International Journal on Computer Science and Engineering, Vol. 3, NO. 7, 2707-2708.

- Ruchi, D., and Rachna, B. (2012). Analysis of Effectiveness of Concurrency Control Techniques in Databases, International Journal of Engineering Research & Technology, Vol. 1, Issue 5, Pp. 1-4.
- Saeid, P., and Mahdi, R. (2015). Modeling Timestamp Ordering Method Using Coloured Petri Net, Indian Journal of Science and Technology, Vol. 8, Issue 35, Pp 1-6.
- Shefali, N., and Samrat, K. (2015). Revisited Performance Issues in Concurrent Transaction Execution in Distributed Database Management System, International Journal of Current Engineering and Scientific Research, Vol. 2, Issue 4, Pp. 23-25.
- Sonal, K., and Morena, R. D (2015). Analysis and Comparison of Concurrency Control Techniques, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 3, Pp 245-247.
- Svein, Y.W. (2008). Methods for Enhancement of Timestamp Evidence in Digital Investigations, Norwegian University of Science and Technology, Faculty of Information Technology, Mathematics and Electrical Engineering Department of Telematics, Pp. 1-3.
- Choe, T.Y. (2008). Optimistic Concurrency Control based on Cache Coherency in Distributed Database Systems, International Journal of Computer Science and Network Security, Vol. 8 No. 11, Pp. 148-149.

Tamer, M.O., and Patrick, V. (2010). Principles of Distributed Database Systems, Third Edition, Pp. 384-385.

Utku, A. (2011), Relaxing Concurrency Control in Transactional Memory, Department of Electrical and Computer Engineering University of Toronto, Pp. 19-21.

APPENDIX A

SOURCE CODE

index.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;

public partial class Cashier_index : System.Web.UI.Page
{
    dbcon dbconnect = new dbcon();
    DataTable dt = new DataTable();
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        string userNam = txtuname.Text.Replace("'", "\'");
        string password = txtpwd.Text.Replace("'", "\'");

        bool ValidRecord = true;
        if (ValidRecord == true)
        {
            string query = "SELECT * FROM admin WHERE username='" + userNam + "' and password='" + password + "'";
            dt = dbconnect.databaseReturnDt(query);
            if (dt.Rows.Count > 0)
            {
                Session["Admin_ID"] = dt.Rows[0]["Admin_id"].ToString();
                Response.Redirect("manageCutomer.aspx");
            }
            else
            {
                //Panell1.Visible = true;
                msg.Text = "Wrong Username or Password";
            }
        }
    }
}
```

index.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="index.aspx.cs"
Inherits="Cashier_index" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>

    <link href="css/bootstrap.css" rel="stylesheet" />
    <link href="css/style.css" rel="stylesheet" />
    <link href="css/cashier.css" type="text/css" rel="Stylesheet" />

    <style type="text/css">

        body
        {
            background-color:#fff;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">

        <header>
            <div class="container">
                <div class="row">
                    <div class="col-md-2">
                    </div>
                    <div class="col-md-4">
                        
                    </div>
                    <div class="col-md-6">
                        <h3><span class="em-text">eBanking
System</span></h3>
                    </div>
                </div>
            </div>
        </header>

        <div class="block">
            <h3>Admin</h3>
            <h4><asp:Label ID="msg" runat="server" class="label label-danger"
Text=""></asp:Label></h4>

            <form>
                <div class="form-group">
                    <label >Username</label>
                    <div class="input-group">
                        <span class="glyphicon glyphicon-user input-group-addon"></span>
```

```

        <asp:TextBox ID="txtuname" runat="server" class="form-control"
placeholder="Enter Email" required autofocus></asp:TextBox>
        </div>
    </div>

    <div class="form-group">
        <label>Password</label>
        <div class="input-group">
            <span class="glyphicon glyphicon-lock input-group-addon"></span>
            <asp:TextBox ID="txtpwd" runat="server"
class="form-control" placeholder="Enter Password" TextMode="Password" required
autofocus></asp:TextBox>
        </div>
    </div>

    <asp:Button ID="Button1" class="btn btn-primary"
runat="server" Text="Sign In" onclick="Button1_Click"/>

    </form>
</div>
</form>
</body>
</html>

```

Default.aspx.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;

public partial class Customer_Default : System.Web.UI.Page
{
    dbcon dbconnect = new dbcon();
    DataTable dt = new DataTable();
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void signIn_Click(object sender, EventArgs e)
    {
        string acctNum = txtAcctNo.Text;
        string password = txtpwd.Text;

        bool ValidRecord = true;
        if (ValidRecord == true)
        {
            string query = "SELECT * FROM customer WHERE acctNo='" + acctNum.Replace("'",
"\'") + "' and password='" + password.Replace("'", "\'") + "'";

```

```

dt = dbconnect.databaseReturnDt(query);
if (dt.Rows.Count > 0)
{
    Session["sessionAcctNum"] = int.Parse(dt.Rows[0]["acctNo"].ToString());
    Response.Redirect("dashboard.aspx");
}
else if (txtAcctNo.Text == "")
{
    Panel1.Visible = true;
    msg.Text = "Enter Username";
}
else if (txtpwd.Text == "")
{
    Panel1.Visible = true;
    msg.Text = "Enter Password";
}
else
{
    Panel1.Visible = true;
    msg.Text = "Wrong Username or Password";
}
}
}

```

Default.aspx

```

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="Customer_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Home Page</title>

    <link href="css/bootstrap.css" rel="stylesheet" />
    <link href="css/style.css" rel="stylesheet" />
    <link href="css/font-awesome.css" rel="stylesheet" />
</head>
<body>
    <form id="form1" runat="server">
    <section id="contact1" class="container">
        <header>
            <div class="container container-main">
                <div class="row">
                    <div class="col-md-4">
                        
                    </div>
                    <div class="col-md-8">

```



```

System</span></h3>
                                <h3><span class="em-text">eBanking
                                </div>
                                </div>
                                </div>
                                </div>
                                </header>

    <div class="container">
        <nav class="navbar navbar-default">
            <div class="container-fluid">
                <section id="contact">
                    <div class="row">
                        <div class="col-md-8">

                            <div class="navbar-header">
                                <button type="button" class="navbar-toggle
collapsed" data-toggle="collapse" data-target="#navbar"
                                aria-expanded="false" aria-controls="navbar">
                                    <span class="sr-only">Toggle navigation </span>
                                    <span class="icon-bar"> </span>
                                    <span class="icon-bar"> </span>
                                    <span class="icon-bar"> </span>
                                </button>
                            </div>
                            <div id="navbar" class="collapse navbar-collapse">
                                <ul class="nav nav-pills">
                                    <li class="active"><a
href="Default.aspx">Home</a></li>

                                    <li><a href="about.aspx">About</a></li>
                                    <li><a
href="contact.aspx">contact</a></li>

                                </ul>
                            </div>
                        </div>
                        <div class="col-md-4">
                            <h4><span class="em-text">Banking That Suit Your life
style</span></h4>
                        </div>
                    </div>
                </section>
            </div>
        </nav>

        <div class="row">
            <div class="col-md-8">

                Next
                <div id="myCarousel" class="carousel slide" data-
ride="carousel">

                    <!-- Indicators -->
                    <ol class="carousel-indicators">
                        <li data-target="#carousel-example-generic" data-
slide-to="0" class="active"></li>

```

```

slide-to="1"></li>
slide-to="2"></li>
</ol>

<!-- Wrapper for slides -->
<div class="carousel-inner" role="listbox">

    <div class="item active">
        
        <div class="container">
            <div class="carousel-caption">
                <div class="rows">
                    <div class="col-md-7">
                        <h1>Image
one</h1>
                        <p></p>
                        <p><a
class="btn btn-lg btn-primary" href="#" role="button">Learn more</a></p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <div class="item">
        
        <div class="container">
            <div class="carousel-caption">
                <div class="rows">
                    <div class="col-md-7">
                        <h1>Image
Two</h1>
                        <p></p>
                        <p><a
class="btn btn-lg btn-primary" href="#" role="button">Learn more</a></p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div><!-- Wrapper for slides ends -->

    <div class="item">
        
        <div class="container">
            <div class="carousel-caption">
                <div class="rows">
                    <div class="col-md-7">
                        <h1>Image
Three</h1>
                        <p></p>
                        <p><a
class="btn btn-lg btn-primary" href="#" role="button">Learn more</a></p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

```

```

        </div>
    </div>
</div>
</div>

<!-- Controls -->
<a class="left carousel-control" href="#myCarousel"
role="button" data-slide="prev">
    <span class="glyphicon glyphicon-chevron-left" aria-
hidden="true"></span>
    <span class="sr-only">Previous</span>
</a>
<a class="right carousel-control" href="#myCarousel"
role="button" data-slide="next">
    <span class="glyphicon glyphicon-chevron-right" aria-
hidden="true"></span>
    <span class="sr-only">Next</span>
</a>
</div>

</div>
<div class="col-md-4">
    <h4><span class="em-text">LOGIN</span></h4>

    <asp:Panel ID="Panel1" runat="server" Visible="False">
        <asp:Label ID="msg" class="label label-danger"
runat="server"></asp:Label>
    </asp:Panel>

    <asp:Panel ID="Panel2" runat="server">
        <div class="form-group">
            <asp:Label ID="Label1" class="control-label" runat="server"
Text="Label">Account Number</asp:Label>
            <div class="input-group">

                <span class="glyphicon glyphicon-user input-group-
addon"></span>
                <asp:TextBox ID="txtAcctNo" runat="server" class="form-
control" placeholder="Account Number" required autofocus></asp:TextBox>
            </div>

            <asp:Label ID="Label2" class="control-label"
runat="server" Text="Label">Password</asp:Label>
            <div class="input-group">

                <span class="glyphicon glyphicon-lock input-group-
addon"></span>
                <asp:TextBox ID="txtpwd" runat="server" class="form-
control"
                    placeholder="Password" required
TextMode="Password"></asp:TextBox>
            </div>
            <asp:Button ID="Button1" runat="server"
                class="btn btn-lg btn-primary btn-block" Text="Submit"
onclick="signIn_Click"></asp:Button>

```

```

        <br />
        <p><span class="em-text">Forgot your password?</span></p>
    </asp:Panel>

    </div>

</div>
</section>

<footer>
    <div class="container">
        <div class="row">
            <div class="col-md-6">
                <ul>
                    <li><a href="Default.aspx">Home</a></li>
                    <li><a href="about.aspx">About</a></li>
                    <li><a href="#">contact</a></li>
                    <li><a href="#Privacy">Privacy
Policy</a></li>
                </ul>
            </div>
            <div class="col-md-6">
                copyright &copy; 2017 All right reversed Abraham
Iboroma
            </div>
        </div>
    </div>
</footer>
<script src="jquery/jquery-3.1.1.min.js"></script>
<script src="js/bootstrap.js"></script>
</form>
</body>
</html>

```

dashboard.aspx.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;

public partial class Customer_dashboard : System.Web.UI.Page
{
    dbcon dbconnect = new dbcon();
    DataTable dt = new DataTable();
    protected void Page_Load(object sender, EventArgs e)
    {
        if (Session["sessionAcctNum"] == null || Session["sessionAcctNum"].ToString() ==
        "")
        {
            Response.Redirect("Default.aspx");
        }
    }
}

```

```

    }

    if (!IsPostBack)
    {
        loadgrid();
    }
}

public void loadgrid()
{
    string query = "select * from transactions where acctNo =" +
Session["sessionAcctNum"] + "' ORDER BY trans_id DESC";
    dt = dbconnect.databaseReturnDt(query);
    GridView2.DataSource = dt;
    GridView2.DataBind();

    string query5 = "select * from customer where acctNo =" +
Session["sessionAcctNum"] + "'";
    dt = dbconnect.databaseReturnDt(query5);

    acctName.Text = dt.Rows[0]["names"].ToString();
    msgAcctNo.Text = dt.Rows[0]["acctNo"].ToString();

    double acctBal22 = Convert.ToDouble(dt.Rows[0]["balance"].ToString());
    acctBal.Text = acctBal22.ToString("C",
System.Globalization.CultureInfo.GetCultureInfo("yo-NG"));

    string image = dt.Rows[0]["imagename"].ToString();

    if (image == "NoImage")
    {
        Image9.ImageUrl = "image/nill.jpg";
    }
    else
    {
        Image9.ImageUrl = "~/UserImage/" + image;
    }
}

protected void RowDataBound(object sender, GridViewRowEventArgs e)
{
    e.Row.Attributes.Add("onmouseover", "MouseEvents(this, event)");
    e.Row.Attributes.Add("onmouseout", "MouseEvents(this, event)");
}

protected void GridView2_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    GridView2.PageIndex = e.NewPageIndex;
    loadgrid();
}

protected void GridView2_SelectedIndexChanged(object sender, EventArgs e)
{
}

```

dashboard.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Customer/CustomerMasterPage.master"
AutoEventWireup="true" CodeFile="dashboard.aspx.cs" Inherits="Customer_dashboard" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">

    <link rel="stylesheet" href="css/style1111/core.css" />
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <div>

        <asp:Panel ID="Panel6" runat="server">

            <!--Panel Header Start -->
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h4 class="panel-title">
                        Account No:
                        <asp:Label ID="msgAcctNo" runat="server"
Text="Label"></asp:Label>
                    </h4>
                </div>

                <div class="panel_Header">
                    <div class="row">
                        <div class="col-md-4">
                            <div class="heaaad reed">
                                <h4 class=""><asp:Label ID="acctName"
runat="server"></asp:Label></h4>
                                <span>Account Holder Name</span>
                            </div>
                        </div>
                        <div class="col-md-4">
                            <div class="heaaad gren">
                                <h4 class=""><asp:Label ID="acctBal"
runat="server"></asp:Label></h4>
                                <span> Available Balance</span>
                            </div>
                        </div>
                        <div class="col-md-4">
                            <div class="heaaad">
                                <asp:Image ID="Image9" runat="server"
BorderStyle="Ridge" class="profile-img"/>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

        <!--Panel Headerr end -->

    </asp:Panel>

    <h4>STATEMENT OF RECENT TRANSACTIONS</h4>
    <asp:GridView ID="GridView2" runat="server" CssClass="Grid" CellPadding="4"
        AutoGenerateColumns="False" onrowdatabound="RowDataBound"
        AllowPaging="True"
        onpageindexchanging="GridView2_PageIndexChanging"
        onselectedindexchanged="GridView2_SelectedIndexChanged">
        <Columns>
            <asp:TemplateField>
            </asp:TemplateField>
            <asp:BoundField DataField="date" HeaderText="DATE"/>
            <asp:BoundField DataField="transacType" HeaderText="TRANSACTION
TYPE"/>

            <asp:BoundField DataField="names" HeaderText="NAMES" />
            <asp:BoundField DataField="tranf_amt" HeaderText="AMOUNT" />
            <asp:BoundField DataField="balance" HeaderText="BALANCE" />

        </Columns>
        <HeaderStyle CssClass="GridHeader" />
        <RowStyle CssClass="GridRow" />
        <AlternatingRowStyle CssClass="GridAlternateRow" />
        <SelectedRowStyle CssClass="GridSelectedRow"/>
    </asp:GridView>

</div>
</asp:Content>

```

customerTransfer.aspx.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;

public partial class customerTransfer : System.Web.UI.Page
{
    dbcon dbconnect = new dbcon();
    DataTable dt = new DataTable();
    double cust_acctNo;
    double ben_acctNo;
    double tranAmt;
    double fin_bal;
    double ben_fin_bal;
    double acct1;

```

```

double acct2;
string receiveName;
string tranfType = "Transfer";

protected void Page_Load(object sender, EventArgs e)
{
    if (Session["sessionAcctNum"] == null || Session["sessionAcctNum"].ToString() ==
    "")
    {
        Response.Redirect("Default.aspx");
    }

    if (!IsPostBack)
    {
        AcctInfo();
    }
}

public void AcctInfo()
{
    string query5 = "select * from customer where acctNo = '" +
    Session["sessionAcctNum"] + "'";
    dt = dbconnect.databaseReturnDt(query5);

    lbl_Acct_Time.Text = dt.Rows[0]["acct_time"].ToString();
    lbl_Tran_Time.Text = DateTime.Now.ToString();

    acctName.Text = dt.Rows[0]["names"].ToString();
    msgAcctNo.Text = dt.Rows[0]["acctNo"].ToString();

    double acctBal22 = Convert.ToDouble(dt.Rows[0]["balance"].ToString());
    acctBal.Text = acctBal22.ToString("C",
    System.Globalization.CultureInfo.GetCultureInfo("yo-NG"));

    txtBBal.Text = acctBal22.ToString("C",
    System.Globalization.CultureInfo.GetCultureInfo("yo-NG"));

    string image = dt.Rows[0]["imagename"].ToString();

    if (image == "NoImage")
    {
        Image9.ImageUrl = "image/nill.jpg";
    }
    else
    {
        Image9.ImageUrl = "~/UserImage/" + image;
    }
}

```



```

}

protected void ComfirmTranf_Click(object sender, EventArgs e)
{
    if (txtbenAcctNo.Text == "")
    {
        msg22.Text = "Enter Beneficiary Acct No";
    }
    else if (txtTranAmt.Text == "")
    {
        msg22.Text = "Enter Ammount";
    }
    else
    {
        //retrieve customer account from database
        string query1 = "SELECT * FROM customer " + " where acctNo = '" +
Session["sessionAcctNum"] + "'";
        dt = dbconnect.databaseReturnDt(query1);
        lblSourceAcct.Text = dt.Rows[0]["acctNo"].ToString() + " (" +
dt.Rows[0]["names"].ToString() + ")";

        //retrieve beneficiary account from database
        bool Validrecord = true;
        string query2 = "SELECT * FROM customer " + " where acctNo = '" +
txtbenAcctNo.Text + "'";
        dt = dbconnect.databaseReturnDt(query2);

        if (dt.Rows.Count > 0)
        {
            lblBenAcct.Text = txtbenAcctNo.Text + " (" +
dt.Rows[0]["names"].ToString() + ")";
            lblAmt.Text = txtTranAmt.Text;
            lblcomment.Text = txtcomment.Text;

            Panel2.Visible = true;
            Panel1.Visible = false;
            Panel3.Visible = false;
            Panel4.Visible = false;
            Panel6.Visible = true;
        }
        else {
            msg22.Text = "Beneficiary Account Number Does not Exist";
        }
    }
}

protected void CancelTranf_Click(object sender, EventArgs e)
{
    AcctInfo();
    Panel1.Visible = true;
    Panel2.Visible = false;
    Panel3.Visible = false;
    Panel4.Visible = false;
}

```

```

protected void CompleteTransfer_Click(object sender, EventArgs e)
{
    //retrieve beneficiary customer from database
    string query3 = "select * from customer where acctNo = '" +
Session["sessionAcctNum"] + "'";
    dt = dbconnect.databaseReturnDt(query3);

    string custNames = dt.Rows[0]["names"].ToString();

    if (lbl_Acct_Time.Text == dt.Rows[0]["acct_time"].ToString())
    {
        acct1 = double.Parse(dt.Rows[0]["Balance"].ToString());
        if (!string.IsNullOrEmpty(txtTranAmt.Text))
        {
            tranAmt = Convert.ToDouble(txtTranAmt.Text);
        }

        if (acct1 <= tranAmt)
        {
            msgError.Text = " <h2>Transaction failed!!! Insufficient Balance</h2>";
            AcctInfo();
            Panel1.Visible = false;
            Panel2.Visible = false;
            Panel3.Visible = false;
            Panel4.Visible = true;
        }
        else
        {
            //compute and update customer details
            fin_bal = acct1 - tranAmt;
            string query4 = "update customer set balance = '" + fin_bal.ToString() +
"'where acctNo='" + Session["sessionAcctNum"] + "'";
            dbconnect.databaseInsert(query4);

            //retrieve and compute beneficiary account from database
            string query5 = "select * from customer where acctNo = '" +
txtbenAcctNo.Text + "'";
            dt = dbconnect.databaseReturnDt(query5);
            string benfnames = dt.Rows[0]["names"].ToString();
            acct2 = double.Parse(dt.Rows[0]["Balance"].ToString());
            ben_fin_bal = acct2 + tranAmt;

            //update beneficiary details
            string query6 = "update customer set balance = '" + ben_fin_bal + "'where
acctNo='" + txtbenAcctNo.Text + "'";
            dbconnect.databaseInsert(query6);

            string update = "Update customer set acct_time = '" + lbl_Tran_Time.Text +
"' where acctNo = '" + Session["sessionAcctNum"] + "'";
            dbconnect.databaseInsert(update);

            //Insert to recent Transactions to customer
            string transactType = "Transfer";

```

```

        string query = "insert into
transactions(acctNo,date,transacType,names,tranf_amt,balance) values ("
        //+ "'" + txtbenAcctNo.Text.Replace("'", "\'") + "',"
        + "'" + Session["sessionAcctNum"] + "',"
        + "'" + DateTime.Now.ToString("dd-MM-yyyy") + "',"
        + "'" + transactType + "',"
        + "'" + benfnames + "',"
        + "'" + "-" + txtTranAmt.Text.Replace("'", "\'") + "',"
        + "'" + fin_bal.ToString() + "')";
        dbconnect.databaseInsert(query);

        //Insert to recent Transactions to beneficiary
        string query2 = "insert into
transactions(acctNo,date,transacType,names,tranf_amt,balance) values ("
        + "'" + txtbenAcctNo.Text.Replace("'", "\'") + "',"
        + "'" + DateTime.Now.ToString("dd-MM-yyyy") + "',"
        + "'" + transactType + "',"
        + "'" + custNames + "',"
        + "'" + "+" + txtTranAmt.Text.Replace("'", "\'") + "',"
        + "'" + ben_fin_bal.ToString() + "')";
        dbconnect.databaseInsert(query2);

        msgSucc.Text = "<h2>Transfer Successful </h2>";
        Panel1.Visible = false;
        Panel2.Visible = false;
        Panel3.Visible = true;
        Panel4.Visible = false;
    }
}
else
{
    /*Response.Write("<script>alert('Your session has expired!!! Transaction
Aborted#')</script>");*/

    msgError.Text = " <h2>Your session has expired!!! Transaction Aborted</h2>";
    Panel1.Visible = false;
    Panel2.Visible = false;
    Panel3.Visible = false;
    Panel4.Visible = true;
}

}

protected void succ_Click(object sender, EventArgs e)
{
    AcctInfo();
    Panel1.Visible = true;
    Panel2.Visible = false;
    Panel3.Visible = false;
    Panel4.Visible = false;

    txtbenAcctNo.Text = "";
    txtcomment.Text = "";
    txtTranAmt.Text = "";
    msg22.Text = "";

```

```

    }
    protected void Reset_Click(object sender, EventArgs e)
    {
        txtbenAcctNo.Text = "";
        txtcomment.Text = "";
        txtTranAmt.Text = "";
        msg22.Text = "";
    }
}

```

customerTransfer.aspx

```

<%@ Page Title="" Language="C#" MasterPageFile="~/Customer/CustomMasterPage.master"
AutoEventWireup="true" CodeFile="customerTransfer.aspx.cs" Inherits="customerTransfer" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">

    <link rel="stylesheet" href="css/style1111/core.css" />

</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <div class="row-item">

        <asp:Panel ID="Panel6" runat="server">

            <!--Panel Header Start -->
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h4 class="panel-title">
                            Account No:
                            <asp:Label ID="msgAcctNo" runat="server"
Text="Label"></asp:Label>
                        </h4>
                    </div>

                    <div class="panel_Header">
                        <div class="row">
                            <div class="col-md-4">
                                <div class="heaaad reed">
                                    <h4 class=""><asp:Label ID="acctName"
runat="server"></asp:Label></h4>
                                    <span>Account Holder Name</span>
                                </div>
                                <h4><asp:Label ID="lbl_Tran_Time" class="label label-
danger" runat="server"></asp:Label></h4>
                            </div>
                            <div class="col-md-4">
                                <div class="heaaad gren">
                                    <h4 class=""><asp:Label ID="acctBal"
runat="server"></asp:Label></h4>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </asp:Panel>
    </div>
</asp:Content>

```

```

        <span> Available Balance</span>
    </div>
    <h4><asp:Label ID="lbl_Acct_Time" class="label label-
success" runat="server"></asp:Label></h4>
    </div>
    <div class="col-md-4">
    <div class="heaaad">
        <asp:Image ID="Image9" runat="server"
BorderStyle="Ridge" class="profile-img"/>
    </div>
    </div>
    </div>
    </div>
    </div>
    <!--Panel Headerr end -->

</asp:Panel>

<asp:Panel ID="Panel1" runat="server">
    <table class="table table-bordered">
        <tr class="info">
            <td colspan="2">
                <h4> Transfer To The Same Bank <asp:Label ID="msg22" class="label
label-danger" runat="server"></asp:Label>
                </h4>
            </td>
        </tr>
        <tr>
            <td>
                <asp:Label ID="Label1" runat="server" Text="Available
Balancet:"></asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtBBal" runat="server" class="form-control"
Width="200px"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>
                <asp:Label ID="Label2" runat="server" Text="Beneficiary Account
Number:"></asp:Label>
            </td>
            <td class="style1">
                <asp:TextBox ID="txtbenAcctNo" runat="server" class="form-control"
Width="200px"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>
                <asp:Label ID="Label3" runat="server" Text="Amount:"></asp:Label>
            </td>

```

```

        <td>
            <asp:TextBox ID="txtTranAmt" runat="server" class="form-
control"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td>
            <asp:Label ID="Label4" runat="server" Text="Comment:"></asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtcomment" runat="server" class="form-
control"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td>
        </td>
        <td>
            <asp:Button ID="Button3" runat="server" Text="Comfirm your Transfer"
                class="btn btn-primary" onclick="ComfirmTranf_Click"/>
            <asp:Button ID="Button1" runat="server" Text="Reset Transfer"
                class="btn btn-danger" onclick="Reset_Click"/>
        </td>
    </tr>
</table>
</asp:Panel>

```

```

<asp:Panel ID="Panel2" runat="server" Visible="False">
    <table class="table">
        <tr class="success">
            <td colspan="2">
                <h4> You are About to Transfer</h4>
            </td>
        </tr>
        <tr>
            <td>
                <asp:Label ID="Label5" runat="server" Text="Source
Account:"></asp:Label>
            </td>
            <td>
                <asp:Label ID="lblSourceAcct" runat="server" Text=""></asp:Label>
            </td>
        </tr>
        <tr>
            <td>
                <asp:Label ID="Label6" runat="server" Text="Beneficiary
Number:"></asp:Label>
            </td>
            <td>
                <asp:Label ID="lblBenAcct" runat="server" Text=""></asp:Label>
            </td>
        </tr>
        <tr>
            <td>

```

```

        <asp:Label ID="Label7" runat="server" Text="Transfer
Amount:"></asp:Label>
    </td>
    <td>
        <asp:Label ID="lblAmt" runat="server" Text=""></asp:Label>
    </td>
</tr>
<tr>
    <td>
        <asp:Label ID="Label8" runat="server" Text="Comment:"></asp:Label>
    </td>
    <td>
        <asp:Label ID="lblcomment" runat="server" Text=""></asp:Label>
    </td>
</tr>
<tr>
    <td>
        <asp:Button ID="Button2" runat="server" Text="Complete Transfer"
            class="btn btn-primary" onclick="CompleteTransfer_Click"/>

        <asp:Button ID="Button4" runat="server" Text="Cancel Transfer"
            class="btn btn-danger" onclick="CancelTranf_Click"/>
    </td>
</tr>
</table>
</asp:Panel>

```

```

<asp:Panel ID="Panel3" runat="server" class="text-center" Visible="False">
    <div class="disp Succ">
        <div class="row">
            <div class="col-md-9">
                <asp:Label ID="msgSucc" runat="server" style="color: #3C765A; font-
size: medium; " ></asp:Label>
            </div>
            <div class="col-md-1">
                
            </div>
            <div class="col-md-2">
            </div>
        </div>
    </div><br />
    <asp:Button ID="Button5" runat="server" class="btn btn-success" Text="Ok"
onclick="succ_Click"/>
</asp:Panel>

```

```

<asp:Panel ID="Panel4" runat="server" class="text-center" Visible="False">
    <div class="disp Error">
        <asp:Label ID="msgError" runat="server" style="color: #A94D64; font-size:
medium; " ></asp:Label>
    </div><br />
    <asp:Button ID="Button6" runat="server" class="btn btn-danger" Text="Back"
onclick="CancelTranf_Click"/>
</asp:Panel>

```

```

    </div>
</asp:Content>

```

editCustomer.aspx.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
using System.Globalization;

public partial class Cashier_editCustomer : System.Web.UI.Page
{
    dbcon dbconnect = new dbcon();
    DataTable dt = new DataTable();
    string ID;
    string sImageFileExtension;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (Request.QueryString["id"] == null || Request.QueryString["id"].ToString() ==
"" )
        {
            Response.Redirect("index.aspx");
        }

        ID = Request.QueryString["id"]; //get the id from grid in manageCutomer.aspx
        if (!IsPostBack)
        {
            Load_Form();
        }
    }
    public void Load_Form()
    {
        string str = "SELECT * FROM customer" + " where cust_id = '" + ID + "'";
        dt = dbconnect.databaseReturnDt(str);

        lbl_Acct_Time.Text = dt.Rows[0]["acct_time"].ToString();
        lbl_Tran_Time.Text = DateTime.Now.ToString();

        msgAcctNo.Text = dt.Rows[0]["acctNo"].ToString();
        acctName.Text = dt.Rows[0]["names"].ToString();

        double acctBal22 = Convert.ToDouble(dt.Rows[0]["balance"].ToString());

        acctBal.Text = acctBal22.ToString("C",
System.Globalization.CultureInfo.GetCultureInfo("yo-NG")); //format number to
money(Nigeria naira)
    }
}

```



```

lblNames.Text = dt.Rows[0]["names"].ToString();
lblPhone.Text = dt.Rows[0]["phone"].ToString();
lblAddr.Text = dt.Rows[0]["address"].ToString();
lblGender.Text = dt.Rows[0]["gender"].ToString();
lblAge.Text = dt.Rows[0]["age"].ToString();
lblState.Text = dt.Rows[0]["state"].ToString();
lblMarital.Text = dt.Rows[0]["marital"].ToString();
lblAcctNo.Text = dt.Rows[0]["acctNo"].ToString();
lblPwd.Text = dt.Rows[0]["password"].ToString();
lblAcctBal.Text = acctBal.Text;

string image = dt.Rows[0]["imagename"].ToString();

if (image == "NoImage")
{
    Image9.ImageUrl = "image/nill.jpg";
}
else
{
    Image9.ImageUrl = "~/UserImage/" + image;
}
}
protected void save_Click(object sender, EventArgs e)
{
    if (txtnames.Text == "")
    {
        msg22.Text = "Enter Customer Names";
    }
    else if (txtphone.Text == "")
    {
        msg22.Text = "Enter Phone Number";
    }

    else if (txtpassword.Text == "")
    {
        msg22.Text = "Enter Password";
    }
    else if (txtacctNo.Text == "")
    {
        msg22.Text = "Enter Account Number";
    }
    else if (txtaddr.Text == "")
    {
        msg22.Text = "Enter Address";
    }
    else if (txtAcctBal.Text == "")
    {
        msg22.Text = "Opening Amount";
    }
    else
    {
        string str = "SELECT * FROM customer" + " where cust_id = '" + ID + "'";
        dt = dbconnect.databaseReturnDt(str);

        if (txtpic.HasFile == false)

```

```

    {
        string str1 = "UPDATE customer SET " + "names='" +
txtnames.Text.Replace("'", "\'") + "', " +
        "phone='" + txtphone.Text.Replace("'", "\'") + "', " +
        "address='" + txtaddr.Text.Replace("'", "\'") + "', " +
        "gender='" + txtgender.Text.Replace("'", "\'") + "', " +
        "age='" + txtage.Text.Replace("'", "\'") + "', " +
        "state='" + txtstate.Text.Replace("'", "\'") + "', " +
        "marital='" + txtmarital.Text.Replace("'", "\'") + "', " +
        "acctNo='" + txtacctNo.Text.Replace("'", "\'") + "', " +
        "password='" + txtpassword.Text.Replace("'", "\'") + "', " +
        "balance='" + txtAcctBal.Text.Replace("'", "\'") + "'" +
        " where cust_id='" + ID + "'";
        dt = dbconnect.databaseReturnDt(str1);

        Response.Redirect("manageCutoomer.aspx");
    }
    else if (txtpic.HasFile == true)
    {
        string myMap = MapPath("~/").ToLower();
        Random r = new Random();
        int next = r.Next();
        string ImageName = txtpic.PostedFile.FileName;

        sImageFileExtension =
ImageName.Substring(ImageName.LastIndexOf(".")).ToLower();
        if (sImageFileExtension == ".gif" || sImageFileExtension == ".png" ||
sImageFileExtension == ".jpg" || sImageFileExtension == ".jpeg" || sImageFileExtension ==
".bmp")
        {
            string ImageSaveURL = myMap + "UserImage/" + next +
sImageFileExtension;

            txtpic.PostedFile.SaveAs(ImageSaveURL);

            string str2 = "UPDATE customer SET " + "names='" +
txtnames.Text.Replace("'", "\'") + "', " +
            "phone='" + txtphone.Text.Replace("'", "\'") + "', " +
            "address='" + txtaddr.Text.Replace("'", "\'") + "', " +
            "gender='" + txtgender.Text.Replace("'", "\'") + "', " +
            "age='" + txtage.Text.Replace("'", "\'") + "', " +
            "state='" + txtstate.Text.Replace("'", "\'") + "', " +
            "marital='" + txtmarital.Text.Replace("'", "\'") + "', " +
            "acctNo='" + txtacctNo.Text.Replace("'", "\'") + "', " +
            "password='" + txtpassword.Text.Replace("'", "\'") + "', " +
            "imagename='" + next + sImageFileExtension + "', " +
            "balance='" + txtAcctBal.Text.Replace("'", "\'") + "'" +
            " where cust_id='" + ID + "'";
            dt = dbconnect.databaseReturnDt(str2);

            Response.Redirect("manageCutoomer.aspx");
        }
    }
    else
    {
        //Response.Write("<script> alert('Your session has expired!!! Transaction
Aborted#</script>");
    }
}

```

```

        }
    }
}
protected void Edit_Click(object sender, EventArgs e)
{
    string str3 = "SELECT * FROM customer" + " where cust_id = '" + ID + "'";
    dt = dbconnect.databaseReturnDt(str3);

    //double acctBal22 = Convert.ToDouble(dt.Rows[0]["balance"].ToString());

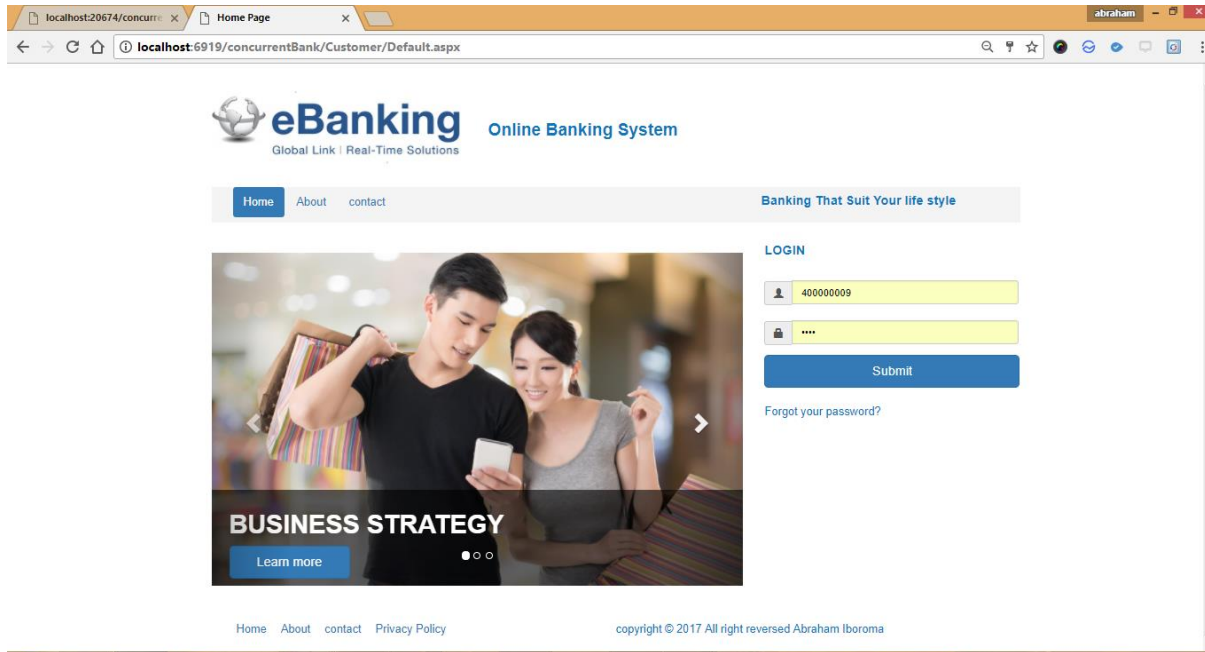
    txtnames.Text = dt.Rows[0]["names"].ToString();
    txtphone.Text = dt.Rows[0]["phone"].ToString();
    txtaddr.Text = dt.Rows[0]["address"].ToString();
    txtgender.Text = dt.Rows[0]["gender"].ToString();
    txtage.Text = dt.Rows[0]["age"].ToString();
    txtstate.Text = dt.Rows[0]["state"].ToString();
    txtmarital.Text = dt.Rows[0]["marital"].ToString();
    txtacctNo.Text = dt.Rows[0]["acctNo"].ToString();
    txtAcctBal.Text = dt.Rows[0]["balance"].ToString(); // acctBal.Text;
    txtpassword.Text = dt.Rows[0]["password"].ToString();

    Panel1.Visible = false;
    Panel2.Visible = false;
    Panel3.Visible = true;
}
protected void Cancel_Click(object sender, EventArgs e)
{
    Response.Redirect("manageCutomer.aspx");
}
}

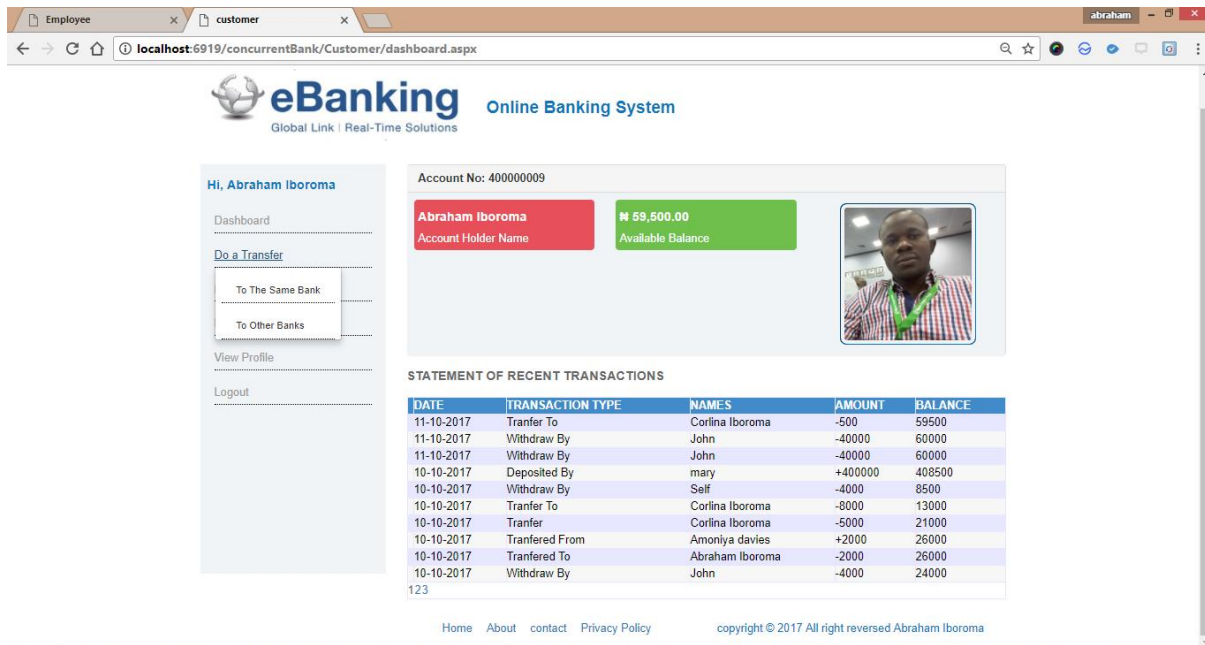
```

APPENDIX B

OUTPUTS SNAP SHOT



Customer Login Page.



Customer initiate transaction

The screenshot shows a web browser window with the URL `localhost:6919/concurrentBank/Customer/customerTransfer.aspx`. The page header includes the eBanking logo and the text "Online Banking System" and "Global Link | Real-Time Solutions". The user is logged in as "Abraham Iboroma" with account number "400000009". The available balance is "₦ 100,000.00". The left sidebar contains links: Dashboard, Do a Transfer, Buy Airtime, Bills and Payment, View Profile, and Logout. The main content area shows the "Transfer To The Same Bank" form with the following fields: Available Balance (₦ 100,000.00), Beneficiary Account Number (400000005), Amount (60000), and Comment (for school fees). There are "Confirm your Transfer" and "Reset Transfer" buttons at the bottom of the form. The footer contains links: Home, About, contact, Privacy Policy, and copyright © 2017 All right reserved Abraham Iboroma.

Employee customer abraham

localhost:6919/concurrentBank/Customer/customerTransfer.aspx

eBanking Online Banking System
Global Link | Real-Time Solutions

Hi, Abraham Iboroma

Account No: 400000009

Abraham Iboroma
Account Holder Name
11/10/2017 09:44:41 AM

₦ 100,000.00
Available Balance
11/10/2017 09:42:21 AM

Transfer To The Same Bank

Available Balance: ₦ 100,000.00

Beneficiary Account Number: 400000005

Amount: 60000

Comment: for school fees

Confirm your Transfer Reset Transfer

Home About contact Privacy Policy copyright © 2017 All right reserved Abraham Iboroma

Customer Transferring From His/hers Account

The screenshot shows the same web browser window as the previous one, but the form is now the "You are About to Transfer" confirmation screen. The fields are: Source Account (400000009 (Abraham Iboroma)), Beneficiary Number (400000005 (Corina Iboroma)), Transfer Amount (60000), and Comment (for school fees). There are "Complete Transfer" and "Cancel Transfer" buttons at the bottom of the form. The footer is the same as the previous screenshot.

Employee customer abraham

localhost:6919/concurrentBank/Customer/customerTransfer.aspx

eBanking Online Banking System
Global Link | Real-Time Solutions

Hi, Abraham Iboroma

Account No: 400000009

Abraham Iboroma
Account Holder Name
11/10/2017 09:44:41 AM

₦ 100,000.00
Available Balance
11/10/2017 09:42:21 AM

You are About to Transfer

Source Account: 400000009 (Abraham Iboroma)

Beneficiary Number: 400000005 (Corina Iboroma)

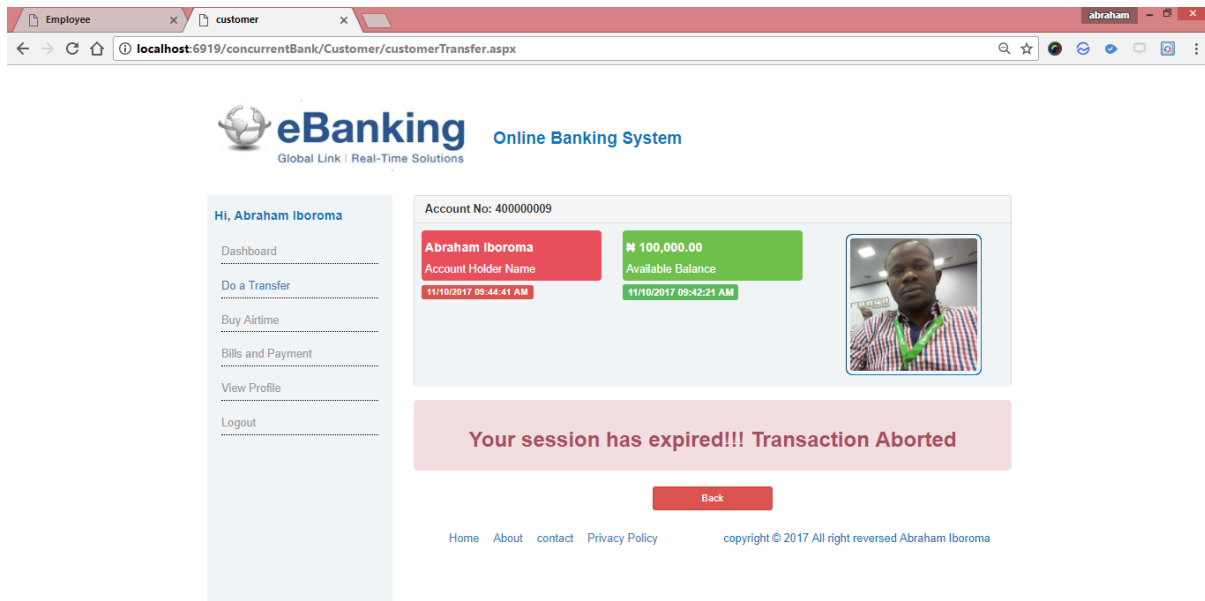
Transfer Amount: 60000

Comment: for school fees

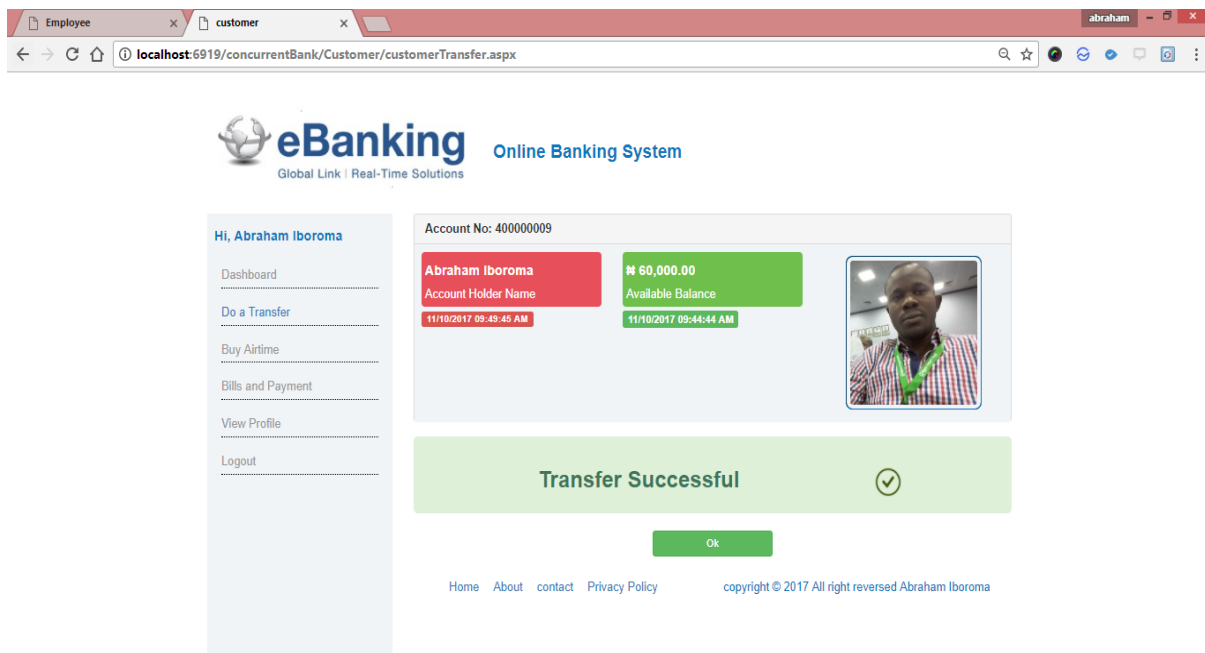
Complete Transfer Cancel Transfer

Home About contact Privacy Policy copyright © 2017 All right reserved Abraham Iboroma

Validating Customer's Transaction




Concurrency Violation is Detected (abort)



Customer Successful Transaction (Committed)

Employee
customer
abraham
localhost:6919/concurrentBank/Customer/customerTransfer.aspx


eBanking
Online Banking System
Global Link | Real-Time Solutions


Hi, Abraham Iboroma

Dashboard
Do a Transfer
Buy Airtime
Bills and Payment
View Profile
Logout

Account No: 400000009

Abraham Iboroma
Account Holder Name
11/10/2017 09:47:41 AM

N 60,000.00
Available Balance
11/10/2017 09:44:44 AM




Transaction failed!!! Insufficient Balance

Back

Home About contact Privacy Policy
copyright © 2017 All right reversed Abraham Iboroma

Employee
customer
abraham
localhost:6919/concurrentBank/Customer/editCustomer.aspx


eBanking
Online Banking System
Global Link | Real-Time Solutions


Hi, Abraham Iboroma

Dashboard
Do a Transfer
Buy Airtime
Bills and Payment
View Profile
Logout

Account No: 400000009

Abraham Iboroma
Account Holder Name
11/10/2017 09:48:17 AM

N 60,000.00
Available Balance
11/10/2017 09:44:44 AM



Edit Customer Profile

Edit Cancel

Names : Abraham Iboroma
Age : 28

Gender : Male
Phone : 08032550577

State : Rivers
Address : Elelenwo

Marital Status : Single
Account No : 400000009


Password : 0000
Acct Balance: N 60,000.00

Home About contact Privacy Policy
copyright © 2017 All right reversed Abraham Iboroma

View Customer Profile

Employee customer abraham

localhost:6919/concurrentBank/Customer/editCustomer.aspx

 **eBanking** Online Banking System
Global Link | Real-Time Solutions

Hi, Abraham Iboroma

Dashboard
Do a Transfer
Buy Airtime
Bills and Payment
View Profile
Logout

Edit Customer Profile

Save Cancel


Names : Abraham Iboroma Age : 28
Gender : Male Phone : 08032550577
State : Rivers Address : Elenwo
Marital Status : Single Account No : 400000009
Password : 0000 Acct Balance : 60000
Upload Picture : Choose File No file chosen

Home About contact Privacy Policy copyright © 2017 All right reversed Abraham Iboroma

Edit Customer Profile

localhost:20674/concurr Home Page abraham

localhost:20674/concurrentBank/Cashier/Index.aspx

 **eBanking** Online Banking System
Global Link | Real-Time Solutions

Admin
Username

admin

Password

....

Sign In

Teller Login Page

Employee x Home Page x abraham

localhost:20674/concurrentBank/Cashier/manageCutoomer.aspx

eBanking Online Banking System
Global Link | Real-Time Solutions

Admin:Dr Young! Manage Customer Transactions Loans Log out Banking That Suit Your life style

Manage Customer Register Delete

Search by Name Search

ID	Name	Phone	Address	Gender	State	Account Number	Account bal
1	Abraham Iboroma	08032550577	Etelenwo	Male	Rivers	400000009	408500
4	Amoniya davis	07026598569	onne	Male	Rivers	400000008	26000
2	Chikaodi Jombo	07023658965	Oyigbo	Female	Imo	400000010	100000
6	Corlina Iboroma	08033666655	Ada George	Female	Rivers	400000005	233500
3	Dabo Etela	07026598569	Niger Str	Male	Rivers	400000007	100000
5	Earnest Karibi	09032658854	Degema	Male	Rivers	400000006	100000
11	Faithful Braide	07069358988	01 Onne	Male	Rivers	400000001	100000
16	Gogo Abe	08033699985	bori	Female	Rivers	400000015	1000
13	John Law	08063259888	01 Onne	Male	Rivers	400000012	1000
15	John Lawrence	07069358988	65 Wosu Close	Male	Kaduna	400000014	410000
8	Luky Maduagu	07069358988	Oyigbo	Male	Imo	400000004	100000
12	Mary Iyalla	08063259888	65 Wosu Close	Male	Rivers	400000000	10000
14	Nimi Allison	07069358988	65 Wosu Close	Female	Select	400000013	10000
20	Nyingi Allison	08033412479	Ivufe	Female	Rivers	400000020	10000

Home About contact Privacy Policy copyright © 2017 All right reversed Abraham Iboroma

Teller View All Customers

RUDIGO Urgent Notificat x Learn JavaScript: Intro x String.prototype - JavaS x discuss.codecademy.com x Employee x abraham

localhost:6919/concurrentBank/Cashier/manageCutoomer.aspx

eBanking Online Banking System
Global Link | Real-Time Solutions

Admin:Dr Young! Manage Customer Transactions Loans Log out Banking That Suit Your life style

Manage Customer Register Delete

Search by Name Search


ID	Name	Phone	Address	Gender	State	Account Number	Account bal
1	Abraham Iboroma	08032550577	Etelenwo	Male	Rivers	400000009	59500

Home About contact Privacy Policy copyright © 2017 All right reversed Abraham Iboroma

Teller Search a Customer

RUDIGO Urgent Notifica... Learn JavaScript: Intro... String.prototype - JavaS... discuss.codecademy.com Employee

localhost:6919/concurrentBank/Cashier/regCustomer2.aspx

 **eBanking** Online Banking System
Global Link | Real-Time Solutions

Admin:Dr Youngl Manage Customer Transactions Loans Log out Banking That Suit Your life style

Add New Customer


Names:
Gender: Male
State: Select
Age:
Phone:
Address:
Marital Status: Select
Password:
Account No: admin
Open Amount:
Upload Picture: No file chosen

Home About contact Privacy Policy copyright © 2017 All right reversed Abraham Iboroma

Teller Add New Customer

Employee Home Page

localhost:20674/concurrentBank/Cashier/withdraw.aspx

 **eBanking** Online Banking System
Global Link | Real-Time Solutions

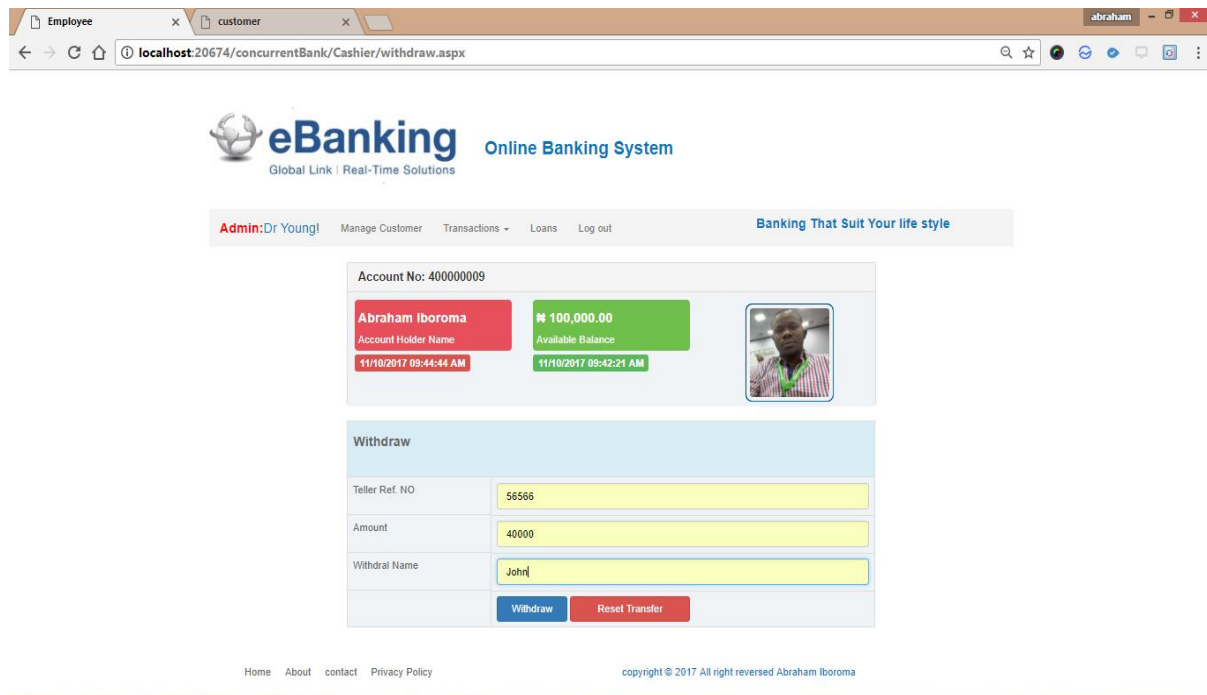
Admin:Dr Youngl Manage Customer Transactions Loans Log out Banking That Suit Your life style

Withdraw

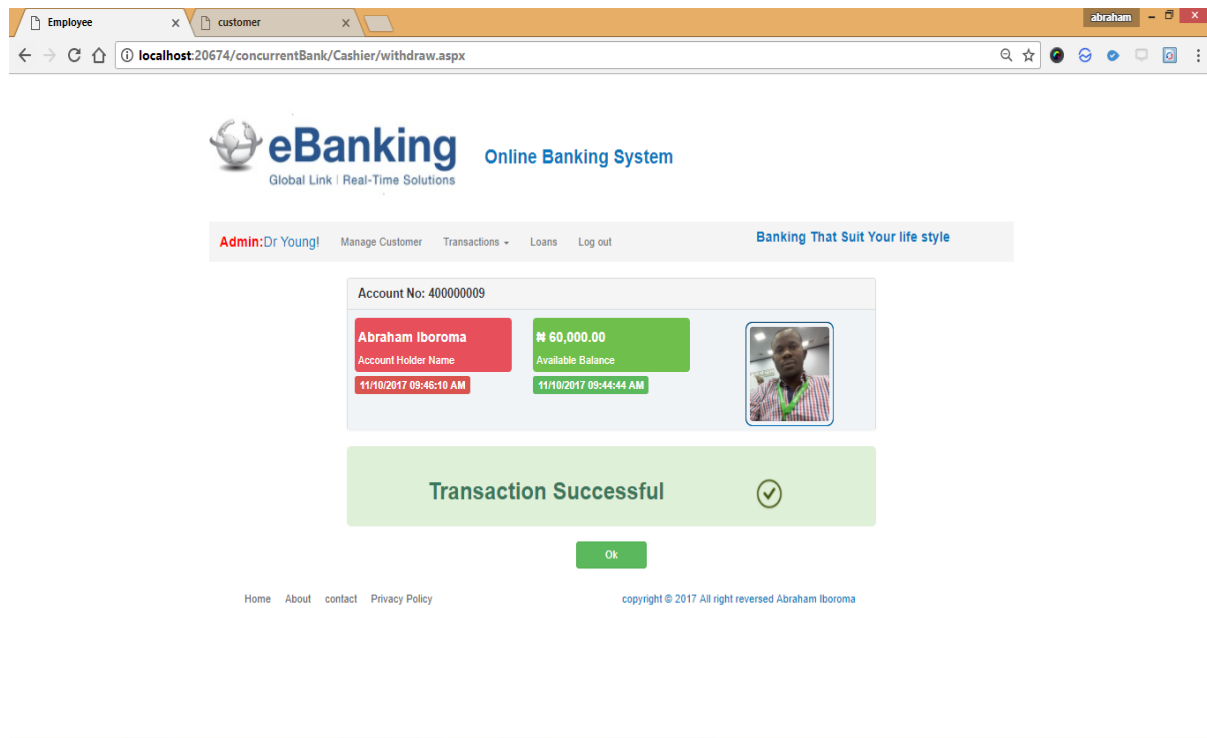
Enter Account Number

Home About contact Privacy Policy copyright © 2017 All right reversed Abraham Iboroma

Teller Initiate transaction



Teller Withdrawing From Customers Account



Teller Successful Transaction (Committed)