# ATTRIBUTE LEVEL LOCKING TO IMPROVE DATA AVAILABILITY IN A DISTRIBUTED SYSTEM

**Swe Zin Aung**

**M.C.Sc.** **JANUARY, 2020**

# ATTRIBUTE LEVEL LOCKING TO IMPROVE DATA AVAILABILITY IN A DISTRIBUTED SYSTEM

**By**

**Swe Zin Aung**

**B.C.Sc.**

**A Dissertation Submitted in Partial Fulfilment of the Requirements for the Degree of**

**Master of Computer Science**

**(M.C.Sc.)**

**University of Computer Studies, Yangon**

**JANUARY, 2020**

## ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere thanks to those who helped me with various aspects of conducting research and writing this thesis. To complete this thesis, many things are needed like my hard work as well as the supporting of many people.

First and foremost, I would like to express my deepest gratitude and my thanks to **Dr. Mie Mie Thet Thwin**, Rector, University of Computer Studies, Yangon, for her kind permission to submit this thesis.

I would like to express my appreciation to **Dr. Thi Thi Soe Nyunt**, Professor, Head of Faculty of Computer Science, University of Computer Studies, Yangon, for her superior suggestion, administrative supports and encouragement during my academic study towards completion of this thesis.

My thanks and regards go to my supervisor, **Daw Khaing**, Associate Professor, Faculty of Information Science Department, University of Computer Studies, Yangon, for her support, guidance, supervision, patience and encouragement during the period of study towards completion of this thesis.

I also wish to express my deepest gratitude to **Daw Aye Aye Khine**, Lecturer, the Department of Language, University of Computer Studies, Yangon, for her editing this thesis from the language point of view.

Moreover, I would like to extend my thanks to all my teachers who taught me throughout the master's degree course and my friends for their cooperation.

I especially thank to my parents, all of my colleagues, and friends for their encouragement and help during my thesis.

## STATEMENT OF ORIGINALITY

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

---------- ----------

Date Swe Zin Aung

## ABSTRACT

Distributed systems consist of a number of computers connected over a network in which numerous users are using such systems locally or via the Internet to meet their requirements. As the system is multiple users' environment, the lock control is an essential part to control the concurrent accessing on the same data item. This system presents a lockable unit on the attributes of a record. So, the database records increased in accessibility because the attributes are locked instead of the entire row. This technique may allow several transactions to access the database row simultaneously by utilizing some attributes and keeping others available for other transactions. In this thesis, concurrency control is mainly implemented by using attribute level locking with the combination of three phases locking (which contain read lock, write intent lock and write lock) in order to prevent dead lock and to control concurrent processing for employee management system. This system is implemented using C# programming language with Microsoft Visual Studio 2013 and database engine is implemented with Microsoft SQL Server 2008 R2.

## TABLE OF CONTENTS


| | | Page |
|---|---|---|
| **ACKNOWLEDGEMENTS** | | **i** |
| **STATEMENT OF ORGINALITY** | | **ii** |
| **ABSTRACT** | | **iii** |
| **TABLE OF CONTENTS** | | **iv** |
| **LIST OF FIGURES** | | **vii** |
| **LIST OF TABLES** | | **viii** |
| **CHAPTER 1** | **INTRODUCTION** | 1 |
| | 1.1 Objective of the Thesis | 1 |
| | 1.2 Related Works | 1 |
| | 1.3 Overview of the Thesis | 2 |
| | 1.4 Organization of the Thesis | 3 |
| **CHAPTER 2** | **THEORETICAL BACKGROUND** | 4 |
| | 2.1 Concurrency Control | 4 |
| | 2.2 Typical Problems between the Concurrent Executions of Transaction | 5 |
| | 2.2.1 Lost or Buried Updates | 5 |
| | 2.2.2 Inconsistent Analysis (Non Repeatable Read) | 6 |
| | 2.2.3 Uncommitted Dependency (Dirty Read) | 7 |
| | 2.2.4 Phantom Reads | 8 |
| | 2.3 Distributed Database | 9 |
| | 2.4 The Structure of Distributed Transaction | 9 |
| | 2.5 Modeling A Distributed Database Management System (DBMS) | 11 |
| | 2.6 Distributed Concurrency Control Algorithm | 11 |

2.6.1 Distributed Two-Phase Locking (2PL) 12

2.6.2 Wound-Wait (WW) 12

2.6.3 Basic Timestamp Ordering (BTO) 13

2.6.4 Distributed Certification (OPT) 13

2.6.5 Three-Phase Locking (3PL) 14

**CHAPTER 3 OPTIMISTIC CONCURRENCY CONTROL** 15

3.1 Types of Concurrency Control Techniques 15

3.1.1 Pessimistic Concurrency Control Method 15

3.1.2 Optimistic Concurrency Control Method 16

3.2 Three Phases of Optimistic Concurrency Control 17

3.2.1 Read Phase 17

3.2.2 Validation Phase 17

3.2.3 Write Phase 18

3.3 Basic Timestamp Ordering (BTO) Algorithm 18

3.4 Local and Global set of Optimistic Approach 19

3.5 Advantages of Optimistic Approach 20

3.6 Lockable Unit in the Traditional Locking Mechanisms 20

3.7 Attribute Level Locking in the proposed System 21

3.8 Three-Phase Locking (3PL) Mechanism 21

**CHAPTER 4 SYSTEM DESIGN AND IMPLEMENTATION** 23

4.1 Authorities of HR Department 24

4.2 Authorities of Finance Department 25

4.3 Authorities of Operation Department 25

4.4 Three-Phase Locking Algorithm 26

4.5 Implementation of the System 28

4.6 The Login Page of the Proposed System 32

4.7 Main Page of the Proposed System 33

4.8 Staff Registration 33

4.9 The Detail Process of HR Department 34

4.10 Concurrency Controlling in Same Attribute 37

**CHAPTER 5 CONCLUSION, LIMITATIONS AND FURTHER EXTENSIONS** 39

5.1 Benefit of using Attribute Level Locking 39

5.2 Limitations and Further Extensions 40

**AUTHOR'S PUBLICATION** 41

**REFERENCES** 42

## LIST OF FIGURES


| Figure | | Page |
|---|---|---|
| Figure 2.1 | Transaction U Loses an Update | 6 |
| Figure 2.2 | Transaction A Performs an Inconsistent Analysis | 7 |
| Figure 2.3 | Transaction A Becomes Dependent on an Uncommitted Change | 8 |
| Figure 2.4 | Phantom Read Case | 8 |
| Figure 2.5 | Architecture of a Distributed Database System | 9 |
| Figure 2.6 | Distributed Transaction Structure | 9 |
| Figure 2.7 | Distributed Database Management (DBMS) Model Structure | 11 |
| Figure 4.1 | The System Flow Diagram of the Proposed System | 31 |
| Figure 4.2 | Login Page of the System | 32 |
| Figure 4.3 | Login Page of the System with Invalid User | 32 |
| Figure 4.4 | Main Page of the System | 33 |
| Figure 4.5 | Staff Registration Form | 33 |
| Figure 4.6 | HR Manager View | 34 |
| Figure 4.7 | HR Manager's Access Grant | 35 |
| Figure 4.8 | Restriction on Own Data Editing [HR as Manager] | 35 |
| Figure 4.9 | Record Deletion Permission is not Allowed on as Manager Level | 36 |
| Figure 4.10 | Read Only Permission of Operation Department | 37 |
| Figure 4.11 | Notifications for Concurrent Accessing on Same Attribute | 37 |
| Figure 4.12 | Data Update Applying | 38 |

## LIST OF TABLES


| Table | | Page |
|---|---|---|
| Table 3.1 | Uncommitted Dependency Problems in Basic Timestamp Ordering (BTO) | 19 |
| Table 4.1 | Types of Departments | 23 |
| Table 4.1 | Granted Permission of Admin | 24 |
| Table 4.2 | Granted Permission of HR Department's Authorize Person on Other Departments | 24 |
| Table 4.3 | Granted Permission of Finance Department's Authorize Person on Other Departments | 25 |
| Table 4.4 | Access Permissions of Operation Department | 26 |
| Table 4.5 | Concurrency Control in the Same Data Item | 30 |
| Table 4.6 | Lock Control for Three-Phase Locking | 29 |
| Table 4.7 | Lock Control for Each Transaction | 29 |

# CHAPTER 1
# INTRODUCTION

A distributed database system (DDBS) is a collection of several logically related databases which are physically distributed in different computers (otherwise called sites) over a computer network. In the database management systems, concurrency control is an important problem for the simultaneous execution of transaction over a shared database that can create several data integrity and consistency problem. In order for transactions to operate concurrently on a shared database, a concurrency control algorithm must be adopted to coordinate their activities.

Traditional concurrency control algorithms can be broadly classified as either pessimistic or optimistic. Pessimistic Concurrency Control (PCC) algorithms avoid concurrent execution of transactions as soon as potential conflicts between these transactions are detected. On the contrary, Optimistic Concurrency Control (OCC) algorithms allow such transaction to proceed at the risk of having to restart them in case these suspected conflicts materialize.

## 1.1 Objectives of the Thesis

The objectives of the thesis are as follows:

- To study how an Attribute Level Locking with Three-Phase Locking algorithm works to improve data availability in a concurrent situation
- To understand how consistency is important in database system with multiple users
- To illustrate read data has no block condition in any situation by using attributes as a lock unit
- To apply the write intent lock including in Three-Phase Locking which has pre-commit phase to prevent block state

## 1.2 Related Works

The problem of data availability and the degree of concurrent transactions have been discussed by several researchers [3, 4, 9, and 11] who concentrated on a strategy of dividing the database into variable size units. The size of such units is

dynamically managed by the lock manager based on user needs and competition. This competition increases more in a distributed database system than in a centralized one because of the higher number of users.

Concurrency control is a challenge issue to distributed network. In airline reservation system, users may access data concurrently and they must get consistent data [12]. The system embedded locking technique provides consistent data for concurrent users. It is the system that can be accessed concurrently because three sites are parallel running. This air-line reservation system uses Basic two-phase locking technique.

## 1.3 Overview of the Thesis

In two-phase locking rules, it is simply states that no transaction should request a lock after it releases one of its locks. Alternatively, a transaction should not release a lock until it is certain that it will not request another lock. So, it can occur deadlock. This system presents a new approach for increasing the data availability by suggesting the attribute as a new lockable database unit to access the several transactions in the same database row simultaneously, which may increase the degree of concurrency and the availability of data. This system presents an implementation of employee management system. In this system, each user level can perform the respective authorize tasks from each department of the system. In HR Department, the authorize person selects one row to update the detail data. At the same time, another Department of authorize ones can choice the same row. In this situation, multiple transactions may occur for the management process and may lead to concurrency problem on a specific code of data. Although there may be concurrent transaction processing, this system can manage the concurrent transaction by using three phase locking instead of blocking methods with the combination of attribute level locking. This system uses three-phase locking instead of two-phase locking protocol. Three-phase locking protocol has a pre-commit phase to prevent the blocking state. Locking can be granted on some attributes of a row, including the key of that row if no conflicts among transactions could occur.

## 1.4 Organization of the Thesis

The book is organized in five chapters. They are as follows:

In Chapter 1, introduction of concurrency control, objectives of the thesis and organization of the thesis are described. Chapter 2 presents the overview of the concurrency control method, concurrent transactions, and typical problems between the concurrent executions of transactions. In Chapter 3, the detailed discussion of the concurrency control approaches are described. In Chapter 4, expresses the design and implementation of the proposed system. Finally, Chapter 5 concludes this thesis with the advantages of three-phase Locking, its limitation and further extension.

# CHAPTER 2

# THEORETICAL BACKGROUND

Process of managing simultaneous execution of transactions in a shared database, to ensure the serializability of concurrent transactions, is known as concurrency control. Concurrent processing is a computing model in which multiple processors execute instructions simultaneously for better performance. Concurrent means something that happens at the same time as something else. Tasks are broken down into subtasks that are assigned to separate processors to perform simultaneously, instead of sequentially as they would have to be carried out by a single processor. Concurrent processing is sometimes said to be synonymous with parallel processing [4].

## 2.1 Concurrency Control

From the past years, Distributed Databases have taken attention in the database research community [6]. Data distribution and replication offer opportunities for improving performance through parallel query execution and load balancing as well as increasing the availability of data. In fact, these opportunities have played a major role in motivating the design of the current generation of database machines.

Concurrency is defined as concurrent execution of multiple concurrent transactions [4]. For this property of concurrency, it is considered a good way to improve the performance of the database. But, there are some problems produced by applying concurrency. Some of the concurrency problems are transactions conflict and deadlock. Transactions conflict produced due to conflicting multiple transactions on the same data. This leads to roll backing some of the conflicting transactions to enable the others from executing their operations. Also, deadlock problem produced from an infinite wait of transactions for data lock. To apply concurrency control, there are some of traditional concurrency control algorithms designed such as two-phase locking (2PL), Timestamp based Concurrency Control (TCC), and Optimistic Concurrency Control (OCC). Reasons for using Concurrency control method in DBMS is:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues

- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability.

## 2.2 Typical Problems between the Concurrent Executions of Transactions

Concurrency means that different users have access the database at the same time. The task of a concurrency control mechanism is to ensure the consistency of the database while allowing a set of transactions to execute concurrently. In concurrent situations, transaction problems may occur due to the following reasons.

- Lost or Buried Updates
- Inconsistent analysis (Non Repeatable Read)
- Uncommitted dependency (Dirty Read)
- Phantom reads

### 2.2.1 Lost or Buried Updates

Lost or buried updates problem occur when two or more transactions are trying to update same row without being aware of each other. The last update overwrites updates made by the other transactions, which results in lost data. If a second transaction read an item for update after the first transaction has read it, but before the first transaction has committed. Whichever of the transaction commit first, that update will be lost and this is shown in Figure 2.1.

| BEGIN_TRANSACTION(T);<br>K: BOOK(A1);<br>K: SOLD(A2);<br>END_TRANSACTION(T); | | BEGIN _TRANSACTION(U);<br>K: READ(A1,A2);<br>…<br>END_TRANSACTION(U); | |
|---|---|---|---|
| **Operations** | **Status** | **Operation** | **Status** |
| A1.status ← A1.read() | AVAIL | | |

| | | | |
|---|---|---|---|
| A1.Booked() ← | Booked | | |
| | | Status ← A1.read() | BOOKED |
| | | Status ← A2.read() | AVAIL |
| A2.status ← A2.read() | AVAIL | | |
| A2.SOLD | SOLD | (not update and restart to other so) | |
| | | A2.SOLD | Invalid |

**Figure 2.1 Transaction U Loses an Update**

### 2.2.2 Inconsistent Analysis (Non Repeatable Read)

A transaction, if it reads the same data item more than once, should always read the same value. Non repeatable read arises when a second transaction accesses the same the data item several times and reads different data each time because another transaction has been updated this item while the second transaction is reading. It is similar to dirty read. Inconsistent analysis involves multiple read (two or more) of the same term is non- repeatable read. Two transaction A and B operation on account (ACC) records: transaction A is summing account balance, transaction B is transferring an amount 10 from account 3 to account 1. Transaction A has read item x before addition of (10) by transaction B at time $t_{1,}$ and read item z after subtraction of (10) by transaction B at time $t_3$. The result produced by transaction A is obviously incorrect; if transaction A were to go on to write that result back into the database, it would actually leave the database in an inconsistent state because the read item x by transaction A is not repeatable and transaction B commits all of its updates before the transaction A has read item z. Figure 2.2 shows that the transaction **A** performs an inconsistent analysis at time $t_{3.}$

| ACC1<br>X:=40; | ACC2<br>Y:=50; | ACC3<br>Z:=30; |
|---|---|---|
| **Transaction A** | **Time** | **Transaction B** |
| -<br>**SUM:=0;**<br>**read_ item(x);**<br>**SUM:=SUM +x;** | **t1** | -<br>-<br>-<br>- |

| | | |
|---|---|---|
| - | | - |
| **read_item(Y);**<br>**SUM:=SUM +Y;**<br>- | **t2** | -<br>-<br>- |
| -<br>- | **t3** | **read_item(z);**<br>**z:=z-10;** |
| -<br>- | **t4** | **write_item(z);**<br>- |
| -<br>- | **t5** | **read_item(x);**<br>**X:=x+10;** |
| -<br>- | **t6** | **write_item(x);**<br>- |
| -<br>- | **t7** | **Commit**<br>- |
| **read_item(z);**<br>**SUM:=SUM +z;** | **t8** | -<br>- |

**Figure 2.2 Transaction A Performs an Inconsistent Analysis**

## 2.2.3 Uncommitted Dependency (Dirty Read)

A transaction, if it retrieves or updates a data item that has been update by other transactions but yet committed by that other transaction. Dirty read is like to inconsistent analysis, the item read by the one transaction was committed by the other transaction that made the change. When a second transaction selects row/s being updated by another transaction. The second transaction is reading the data that has not yet been committed and may be changed by the transaction executing updates on the row as shown in Figure 2.3.

| BEGIN TRANSACTION(T);<br>K: BOOK(A1);<br>K:SOLD(A2);<br>END_TRANSACTION(T); | | BEGIN_TRANSACTION(U);<br>K:READ(A1,A2)<br>…<br>END_TRANSACTION(U); | |
|---|---|---|---|
| **Operation** | **Status** | **Operation** | **Status** |

| | | | |
|---|---|---|---|
| A1.status ← A1.read() | AVAIL | | |
| A1.Booked; | BOOKED; | | |
| | | status ← A1.read() | BOOKED |
| | | status ← A2.read() | AVAIL |
| A2.status ← A2.read() | AVAIL | | |
| A2.SOLD | SOLD | (not update the restart to other) | |
| | | ….A2.SOLD | Invalid |

**Figure 2.3 Transaction A Becomes Dependent on an Uncommitted Change**

### 2.2.4 Phantom Reads

The transaction's first read of the range of rows shows a row that no longer exists in the second or succeeding read, as a result of a deletion by a different transaction. A transaction re-executes a query, finding a set of data not equal to a previous one although the search condition is unchanged. Phantom reads may causes when insert or delete action is performed against a row that belongs to the range of rows being by a transaction. Suppose user A retrieves the set of all rows that satisfy some condition (e.g.; all suppliers the condition that the city in Paris). Suppose that user B start and inserts a new row satisfying that same condition. If transaction A, now repeats its retrieval request, it will see a row that did not previously exist as shown in figure 2.4.

| Time | User A(User A's Transaction) | User B(User B's Transaction) | Result |
|---|---|---|---|
| T1 | Result the number of students who live in Paris; | - | 120 |
| T2 | Make Ferry Card for 120 students | Add to new students{John live in Paris} | |
| T3 | Distributed the ferry card for each | | |

**Figure 2.4 Phantom Read Case**

## 2.3 Distributed Database

Distributed Databases have taken attention in the database research community. Data distribution and replication offer opportunities for improving performance through parallel query execution and load balancing as well as increasing the availability of data. In fact, these opportunities have played a major role in motivating the design of the current generation of database machines.

A distributed database management system (DDBMS) involves a collection of sites interconnected by a network. Each site runs one or more of the following software modules: a transaction manager (TM), a data management (DM), and a concurrency control scheduler (or simply scheduler). In a client-server model, a site can function as a client or a server, or both. A client runs only the TM module, and a server runs only the DM and scheduler modules. Each server stores a portion of the database. Each data item may be stored at any server or redundantly at several servers. Figure 2.5 shows the system architecture of a Distributed Database System.

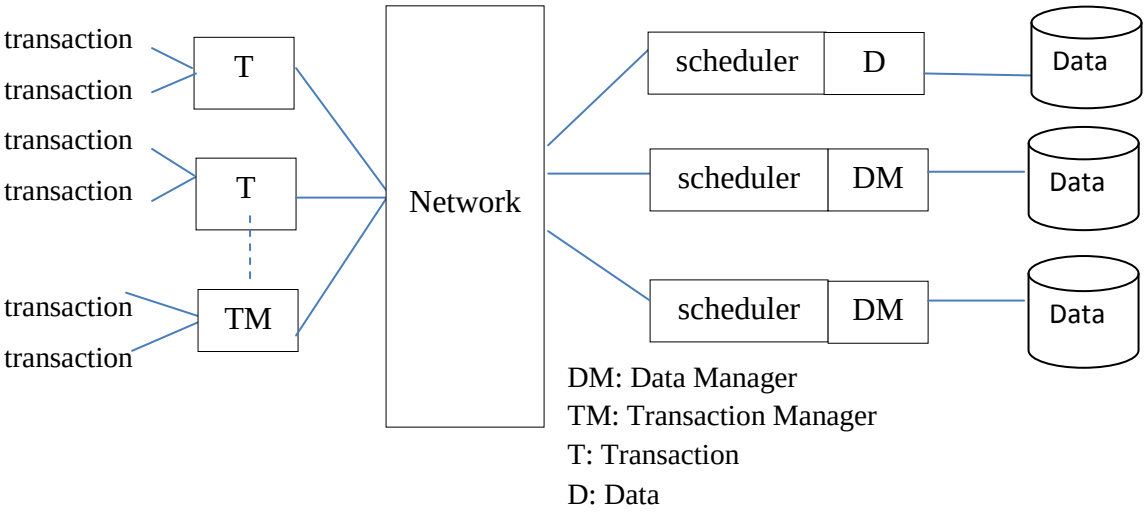



**Figure 2.5 Architecture of a Distributed Database System**

## 2.4 The Structure of Distributed Transactions

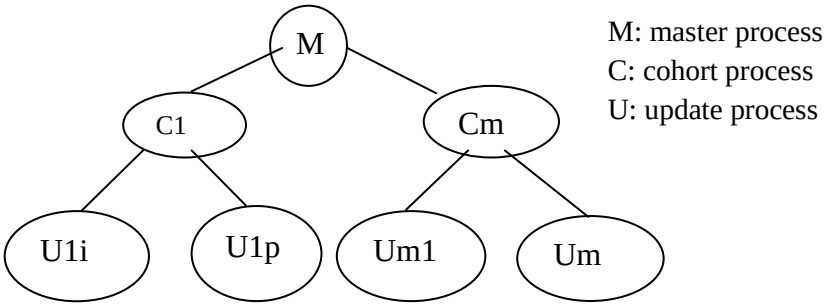



**Figure 2.6 Distributed Transaction Structure**

A general distributed transaction of the processes involved in its execution as shown in Figure 2.6. Each transaction has a master process (M) that runs at its site of origination. The master process sets up a collection of Cohort processes(C) to perform the actual processing involved in running the transaction. Since virtually all query processing strategies for distributed database systems involve accessing data at the site(s) where it resides, rather than accessing it remotely. There is at least one such cohort for each site where data is accessed by the transaction. In general, data may be replicated, in which case each cohort that update any data items is assumed to have one or more update processes associated with it at other sites. In particular, a cohort will have an update process at each remote site that stores a copy of the data items that it updates.

It communicates with its update processes (U) for concurrency control purposes, and it also sends those copies of the relevant updates during the first phase of the commit protocol. The centralized two-phase commit protocol will be used in conjunction with each of the concurrency control algorithms examined. When the master has received such a message from each cohort, it will initiate the commit protocol by sending "prepare to commit" messages to all sites. Assuming that a cohort wishes to commit, it sends a "prepared" message back to the master, and the master will send "commit" messages to each cohort after receiving prepared messages from all cohorts. The protocol ends with the master receiving "committed" messages from each of the cohorts. If any cohort is unable to commit, it will return a "cannot commit" message instead of a "prepared" message in the first phase, causing the master to send "abort" instead of "commit" messages in the second phase of the protocol. When replica update processes are present, the commit protocol becomes a nested two-phase commit protocol. Messages flow between the master and the cohorts, and the cohorts in turn interact with their updaters. That is, each cohort sends "prepare to commit" messages to its updaters after receiving such a message from the master, and it gathers the responses from its updaters before sending a "prepared" message back to the master; phase two of the protocol is similarly modified.

Distributed transactions deal with a physical division of transactions caused by the need to access distributed resources. Special distributed algorithms are needed to handle locking of data and committing of transactions.

## 2.5 Modeling A Distributed DBMS

Figure 2.7 shows the general structure of the distributed DBMS model. Each site in the model has four components: a source, which generates transactions and also maintains transaction-level performance information for the site, a transaction manager, which models the execution behavior of transactions, a concurrency control manager, which implements the details of a particular concurrency control algorithm, and a resource manager, which models the CPU and I/O resources of the site. In addition to these per-site components, the model also has a network manager, which models the behavior of the communications network.

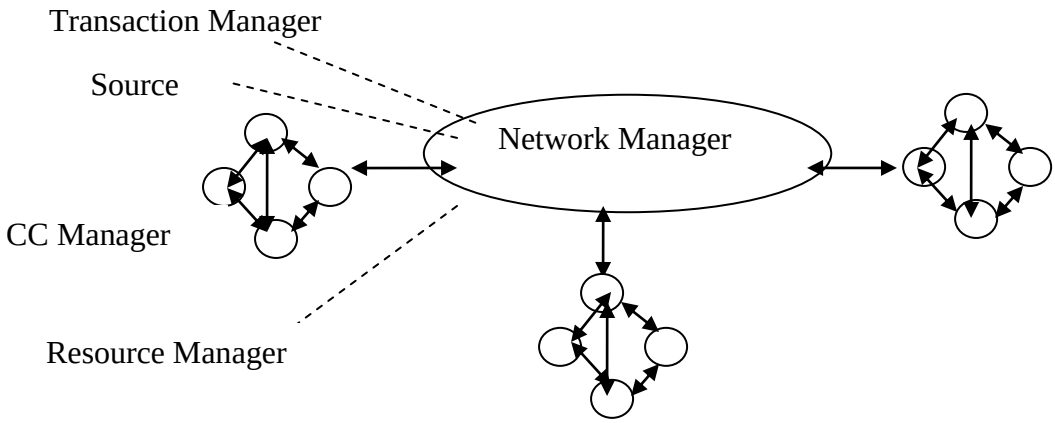



**Fig 2.7 Distributed Database Management (DBMS) Model Structure**

## 2.6 Distributed Concurrency Control Algorithms

Concurrency control ensures the consistency and reliability properties of transaction [13].Simultaneous executions of transactions in distributed system create several mutual exclusions and deadlock problems. Four algorithms of distributed concurrency control are described as follows:

- Distributed Two-Phase Locking (2PL)
- Wound-Wait (WW)
- Basic Timestamp ordering (BTO)
- Distributed Certification (OPT)
- Three-Phase Locking(3PL)

### 2.6.1 Distributed Two-Phase Locking (2PL)

The first algorithm is the distributed “read any, write all” two-phase locking algorithm. Transactions set read locks on items that they read, and they convert their read locks to write locks on items that need to be updated. To read an item, it suffices to set a read lock on any copy of the item, so the local copy is locked; to update an item, write locks are required on all copies. Write locks are obtained as the transaction executes, with the transaction blocking on a write request until all of the copies of the item to be updated have been successfully locked. All locks are held until the transaction has successfully committed or aborted. Deadlock is a possibility, Local deadlocks are checked for any time a transaction blocks, and are resolved when necessary by restarting the transaction with the most recent initial startup time among those involved in the deadlock cycle. (A cohort is restarted by aborting it locally and sending an “abort” message to its master, which in turn notifies all of the processes involved in the transaction.) Global deadlock detection is handled by a “Snoop” process, which periodically requests waits-for information from all sites and then checks for and resolves any global deadlocks (using the same victim selection criteria as for local deadlocks). Instead, each site takes a turn being the “Snoop” site and then hands this task over to the next site. The “Snoop” responsibility thus rotates among the sites in a round-robin fashion, ensuring that no one site will become a bottleneck due to global deadlock detection costs.

### 2.6.2 Wound-Wait (WW)

The second algorithm is the distributed wound-wait locking algorithm, again with the “read any, write all” rule. It differs from 2PL in its handling of the dead lock problem: Rather than maintaining waits-for information and then checking for local and global deadlocks, deadlocks are prevented via the use of timestamps. Each transaction is numbered according to its initial startup time, and younger transactions are prevented from making older ones wait. If an older transaction requests a lock, and if the request would lead to the older transaction waiting for a younger transaction, the Approach for concurrency control in distributed database system younger transaction is “wounded” it is restarted unless it is already in the second phase of its commit protocol (in which case the “wound” is not fatal, and is simply

ignored). Younger transactions can wait for older transactions so that the possibility of deadlocks is eliminated.

### 2.6.3 Basic Timestamp Ordering (BTO)

The third algorithm is the basic timestamp ordering algorithm. Like wound-wait, it employs transaction startup timestamps, but it uses them differently. Rather than using a locking approach, BTO associates timestamps with all recently accessed data items and requires that conflicting data accesses by transactions be performed in timestamp order. Transactions that attempt to perform out-of-order accesses are restarted. When a read request is received for an item, it is permitted if the timestamp of the requester exceeds the item's write timestamp. When a write request is received, it is permitted if the requester's timestamp exceeds the read timestamp of the item; in the event that the timestamp of the requester is less than the write timestamp of the item, the update is simply ignored (by the Thomas write rule). For replicated data, the "read any, write all" approach is used, so a read request may be sent to any copy while a write request must be sent to (and approved by) all copies.

### 2.6.4 Distributed Certification (OPT)

The fourth algorithm is distributed timestamp-based, optimistic concurrency control algorithm, which operates by exchanging certification information during the commit protocol. For each data item, a read timestamp and a write timestamp are maintained. Transactions may read and update data items freely, storing any updates into a local workspace until commit time. For each read, the transaction must remember the version identifier (i.e., write timestamp) associated with the item when it was read. Then, when all of the transaction's cohorts have completed their work, and have reported back to the master, the transaction is assigned a globally unique timestamp. This time stamp is sent to each cohort in the "prepare to commit" message, and it is used to locally certify all of its reads and writes as follows: A read request is certified if the version that was read is still the current version of the item, and no write with a newer timestamp has already been locally certified. A write request is certified if no later reads have been certified and subsequently committed, and no later reads have been locally certified already updaters. As described earlier, the master resides at the site where the transaction was submitted. Each cohort makes

a sequence of read and writes requests to one or more files that are stored at its site; a transaction has one cohort at each site where it needs to access data. Cohorts communicate with their updaters when remote write access permission is needed for replicated data, and the updaters then make the required write requests for local copies of the data on behalf of their cohorts. A transaction can execute in either a sequential or parallel fashion, depending on the execution pattern of the transaction class.

### 2.6.5 Three-Phase Locking (3PL)

Three-phase locking protocol is a non-blocking protocol because it includes a pre-commit phase (write-intent locking phase) to prevent the blocking state. This phase is reached if all transaction participants have voted to commit. Otherwise, and if this state is not reached. In the write-intent mode, the processes on the data are virtual (because of the processing is in the pre-commit phase.) Dependent on the nature of the data and the environment: changing data, many users. Types of locks used: Read, Write, Write Intent.

# CHAPTER 3
# OPTIMISTIC CONCURRENCY CONTROL

The concurrency control is the coordination of the simultaneous execution of a transaction in a multiuser database system. The concurrency control can ensure the serializability of the transaction in a multiuser database environment and that is the main objective of concurrency control. Deadlock avoidance works to avoid deadlock but it does not totally prevent it. The basic idea here is to allocate resources only if the resulting global state is a safe state. In other words, unsafe states are avoided, meaning that deadlock is avoided as well.

## 3.1 Types of Concurrency Control Techniques

Various concurrency control algorithms differ in the time when conflicts are detected, and in the way they are resolved conflicts [4].Concurrency Control methods can be classified into two categories as follows:

1. Pessimistic Concurrency Control Method
2. Optimistic Concurrency Control method

### 3.1.1 Pessimistic Concurrency Control Method

The main feature of the pessimistic approach is to prevent possible conflicts. Transaction get access to data only if this will not cause possible conflict situation later. If it is not possible immediately the transaction should wait until it will become possible. Most of pessimistic algorithms are based on locks. The classical pessimistic algorithm is the widely used two phase locking (2PL). Pessimistic Concurrency Control, avoid any concurrent executions of transactions as soon as conflicts that might result in future inconsistencies are detected. Pessimistic algorithms synchronize concurrent execution of transactions early in their execution life cycle are validation (V) read(R) computation(C) write (W).

In pessimistic system, actions can delay unnecessarily. In addition, pessimistic systems can get into deadlock situations, where a group of actions is unable to proceed because each action in the group. In a pessimistic system with locking, it is necessary to acquire the appropriate lock (by sending a lock request) before accessing

an object. Thus, a round-trip network delay is required even when reading a cached object. This delay is necessary to ensure two things: the action must read (or modify) an up-to-date copy of the object and the locking rules must be maintained. Pessimistic Concurrency Control can be classified as follows:

1. Two-Phase Locking (2PL)
2. Timestamp Ordering

### 3.1.2 Optimistic Concurrency Control Method

Optimistic Concurrency Control (OCC) allows concurrent transactions to proceed at the risk of having to restart them in case these suspected inconsistencies materialize [4]. Optimistic algorithms delay the synchronization of transaction until their termination: read(R) computation(C) validation (V) write (W).In optimistic systems, processes that fail validation abort and restart, redoing work that would not be redone in a pessimistic system. In addition, optimistic system can prevent deadlock. The ever-increasing demand for higher throughput for more complex transactions in online transaction processing leads to an increase in the degree of transaction concurrency control and a higher lock contention level, which manifests itself by increased frequency in transaction blocking due to lock conflicts and restarts(deadlocks).

In fact, as concurrency increases, there may be a sudden reduction in the number of active transactions due to transaction blocking, which eventually leads to a severe degradation in performance which is referred to as thrashing. In Optimistic Concurrency Control (OCC), transactions are allowed to execute unhindered until they reach their commit point, at which time they are validated. Optimistic Concurrency Control (OCC) provides freedom from deadlock, thus saving the expense that the deadlock detection usually required in locking algorithms. Optimistic Concurrency Control (OCC) transactions involve these phases:

- Begin: Record a timestamp marking the transaction's beginning.
- Modify: Read database values, and tentatively write changes.
- Validate: Check whether other transactions have modified data that this transaction has used (read or written). This includes transactions that completed after this transaction's start time, and optionally, transactions that are still active at validation time.

- Commit/Rollback: If there is no conflict, make all changes take effect. If there is a conflict, resolve it, typically by aborting the transaction, although other resolution schemes are possible.

## 3.2 Three Phases of Optimistic Concurrency Control (OCC)

The basic idea of an Optimistic Concurrency Control mechanism is that the execution of a transaction consists of three phases

1. Read Phase
2. Validation Phase
3. Write Phase

### 3.2.1 Read Phase

A transaction first copies the data objects from the database to its own private buffer not accessible by other transaction[8]. Updates are applied to a local copy of the data and announced to the database system by an operation named pre-write. Transactions run independently at each site until they reach the end of their read phases.

For each transaction, two sets are maintained: a read and a write set. The read set (RS) is the set of data items that are read by a transaction, consists of all selected relations accessed by a transaction. The name of the database relation of which the selected relation is part of and the selector defining the selection predicate are kept in the read set. The write set (WS) is the set of data items, determines the objects written by a transaction.

The read phase is the normal execution of the transactions. Write operations are performed on private data copies in the local workspace of the transaction. This kind of operation is called pre-write. Identification of all pre-check conflicts against other active transactions.

### 3.2.2 Validation Phase

At the end of the read phase, the transaction gets into the validation phase, where it checks whether or not it was in conflict with any other transactions operating in parallel. The key component in Optimistic Concurrency Control (OCC) algorithms is the validation of the serial equivalence criterion, and need to check the execution

sequence of concurrent transactions. The basic idea is that if transaction $T_i$ should not affect the read phase $T_j$ and Ti should not overwrite $T_j$. To satisfy this requirement, one of the following conditions must hold.

1. If all transactions $T_k$ where $ts(T_k) < ts(T_i)$ have completed their write phase before $T_j$ has started its read phase, then validation succeeds.
2. If there is any transaction $T_k$ such that $ts(T_k) < ts(T_i)$ and which completes its write phase while $T_j$ is in its read phase, then validation succeeds if $WS(T_k) = RS(Tj)$
3. If there is any transaction $T_k$ such that $ts(T_k) < ts(T_j)$ and which completes its read phase before $T_j$ completes its read phase ,the validation succeeds if $WS(T_k) = RS(Ti)$

Depending on the kind of examination during validation phase all optimistic protocols can be divided into two classes: forward validation and backward validation protocols commit finished transaction and aborts still working that conflict with it.

### 3.2.3 Write Phase

If validation is successful, transaction updates applied to database otherwise updates are discarded and transaction is aborted and restarted.

## 3.3 Basic Timestamp Ordering (BTO) Algorithm

The timestamp (ts) is a number associated with each transaction. Timestamp is not necessarily real time and can be assigned by logical counter. Timestamp is a unique for each transaction should be assigned in an increasing order for each new transaction. Timestamp based on read any, write all. Read is performed locally on any site [9]. Write is sent to all sites. BTO all sent write access requests between a cohort and its updaters when a write request for replicated data is received at the cohort site. BTO uses timestamps to order transactions a prior restarting transaction when conflicting out-of-order accesses occur read requests must occasionally block when they request data from pending, uncommitted updates.

In the basic timestamp ordering of distributed concurrency control, as in following table. Ticket No: A1 is booked at Site T. For this operation; A1 is read and then updated (written) as ‘BOOKED’. Then at Site U, Ticket A1 is read and it is read as ‘BOOKED’ and Ticket A2 is read as ‘AVAIL’. At other timestamp, A2 is read at

Site T. Then A2 is booked at Site T, but it is not updated as 'BOOKED' at site U. In Site U, A2 is read as AVAIL and A2 is read uncommitted data. So A2 booked at site U is Invalid and uncommitted dependency is occur. So, site U must terminate with ROLLBACK as shown in Table 3.1.

**Table 3.1 Uncommitted Dependency Problems in (BTO)**

| BEGIN_TRANSACTION(T);<br>K: BOOK(A1);<br>K: SOLD(A2);<br>END_TRANSACTION(T); | | BEGIN _TRANSACTION(U);<br>K: READ(A1,A2);<br>…<br>END_TRANSACTION(U); | |
|---|---|---|---|
| **Operations** | **Status** | **Operation** | **Status** |
| A1.status ← A1.reaad() | AVAIL | | |
| A1.Booked() [Commit] | Booked | | |
| | | Status ← A1.read() | BOOKED |
| | | Status ← A2.read() | AVAIL |
| A2.status ← A2.read() | AVAIL | | |
| A2.Booked(() [commit] | BOOKED | (not update and restart to other so) | |
| | | A2.Booked() | {Booked }is Invalid |
| | | Uncommitted (because of uncommitted data or stale data) | |
| | | Uncommitted Dependency Problem in BTO | |

## 3.4 Local and Global Set of Optimistic Approach

Transaction validation is performed at two levels: local and global.

1. The local validation level involves acceptance of each sub transaction locally.
2. The global validation level involves acceptance of a distributed transaction on the basis of local acceptance of all sub-transactions.
3. Local set -The time-stamp is an integer and assigned in ascending order (i.e. using system clock).

4. Global set-In a distributed database system, the site ID may be added to the time-stamp (lower order) to make it globally unique.

## 3.5 Advantages of the Optimistic Approach

According to the optimistic approach the load of the system is maximized. More advantageous had outcome because it tries to execute all transactions simultaneously**,** this gives a better chance for one of the conflicting transaction to commit. The optimistic approach is better than pessimistic for flat transaction. In a pessimistic system (with locking), it is necessary to acquire the appropriate lock (by sending a lock request) before accessing an object. Thus, a round-trip network delay is requiring even when reading an object. This delay is necessary to ensure two things: the action must read (or modify) an up-to-date copy of the object and the locking rules must be maintained. A network lock request is almost as costly as a request for a copy of the object, since the cost of a message is independent of message size, for objects of reasonable size .In optimistic system an action can read objects without using any network messages.

Optimistic system can avoid the delay associated with sending lock requests and reduce the number of messages that have to be processed by the system: as more messages are sent, message transmission times and message processing times both increase. In pessimistic system, must send one message for each lock request, along with some messages at commit time. But optimistic approach, only send messages at commit time. The optimistic commit-time messages would be larger than the pessimistic commit-time messages, since they must include some extra information used for validation. However, the number of messages is more important than the size of each message.

## 3.6 Lockable Unit in the Traditional Locking Mechanisms

In the study of locking techniques, the size of the lockable units clearly has a major effect on the concurrency control and the availability of data, because while the database unit is locked, it will be unavailable for a time. Thus, if the locked unit is a table, then no other transaction can access that table in a conflict mode until the lock is released. This system presents an approach for increasing the data availability by suggesting the attribute as a new lockable database unit. This technique may be

implemented by increasing the database tables' attributes as lockable units instead of the entire row. The proposed approach may allow several transactions to access the same database row simultaneously, which may increase the degree of concurrency and the availability of data. (i.e. while a transaction holds the data item by read lock or write intent lock , the other transaction can get read lock on the same data item without blocking.)

## 3.7 Attribute Level locking in the proposed System

This approach aims to include the attributes that would be the new lockable units for allowing several transactions to access the same database row concurrently. This approach may increase the database resources, which would increase the concurrency and throughput in the system and decrease deadlock occurrences. In the suggested attributes as new lockable units; the attributes may be locked individually when a transaction requires only some attributes of the database row. This locking can be performed as explained in the following steps:

1. The database row is locked in an intent exclusive mode (IX).
2. The key of that row is locked in a shared mode (S).
3. The required attributes can be locked by the requested transaction in read or write.

## 3.8 Three-Phase Locking (3PL) Mechanism

Three-Phase Locking Mechanism is as follow:

Num-of-sites (M) = Number of sites in the system

Num-of-DB = Number of databases in each site →1

OP-Mode = Operation mode → R, RW, W

R = Read mode → Read Lock

RW = Read / Write mode → Write-intent Lock

W = Write mode →Write Lock

**Phase 1: Read mode → Read Lock**

- User sign-in and request data from the database.
- Read lock is applied to the data.
- Once the data is read, the Read lock is dropped.(There can be multiple Read locks by multiple users on the same data/tuple.)

**Phase 2: Read / Write mode → Write-intent Lock**

- When the user indicates that he wishes to edit the data, take out a WRITE-INTENT lock.
- UPDATE: Write Intent Lock also known as Change Lock or Protect Lock.
- Other users can still obtain a Read Lock on the data.
- Write Lock allowed only to the user who has the Write Intent Lock.
- If data locked with Write Intent Lock, then no further Write Intent Locks can be applied on it.

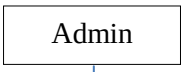
**Phase 3: Write mode →Write Lock**

- When user finishes editing the data and submits the changes, immediately Write Lock is applied on the data.
- Write Intent Lock is unlocked.
- Transaction is committed and Write Lock is unlocked

# CHAPTER 4
# IMPLEMENTATION OF THE SYSTEM

Concurrency control is mostly important in distributed system. This system implemented as an employee management system to control the concurrent situation by using attribute level locking. In this system, the HR Department and Finance Department have the similar authorities to update the same data items. Although HR Department's Manager selects one row to update (i.e. holding write intent lock), Finance Department of Assistant Manager can also access the same row to read without block condition. While the HR Department holding the write lock, other Departments can still obtained the read access with consistence state. So, the proposed system is non-blocking protocol and prevents the deadlock situation (including write-intent lock). This system includes three types of departments as described in Table 4.1.

**Table 4.1 Types of Departments**




| **HR Department** | **Finance Department** | **Operation Department** |
|---|---|---|
| Manager | Manager | Manager |
| Assistant Manager | Assistant Manager | Assistant Manager |
| Senior Staff | Senior Staff | Senior Staff |
| Junior Staff | Junior Staff | Junior Staff |

The admin user has the top granted level on all user levels of each department. The admin has any access grant on all user levels of all departments. The detail access grant permissions of the admin user are described in the Table 4.2.

**Table 4.2 Granted Permission of Admin**

| Action User | HR Department | | | | Finance Department | | | | Operation Department | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Manager | AS Manager | Senior Staff | Junior Staff | Manager | AS Manager | Senior Staff | Junior Staff | Manager | AS Manager | Senior Staff | Junior Staff |
| Admin | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

## 4.1 Authorities of HR Department

Access granted user level of HR Department is clearly defined in Table 4.3. HR Department manager has highest administrative grant on other levels of users (AS Manager, Senior Staff and Junior Staff). But HR Department manager has no permission on financial access grant on all types of users and his/her own profile is only granted on read only permission (Deny on own data editing. For HR Department Assistant Manager Level, access grants on other users' level are almost the same as HR Manager. But, HR Assistant Manager's access grant on Finance Department Manager data and Finance Department Assistant Manager data accessing are denied because of this system is restricted on lower to higher level data accessing. Senior Staff of HR Department has read only permission on own data and lower levels users. Reading on higher level user (reading on HR Manager's data and HR AS Manager's data) is not allowed. Junior Staff of HR Department has read only permission on own data.

**Table 4.3 Granted Permission the HR Department's Authorize Person on Other Departments**

| Action User | | HR Department | | | |
|---|---|---|---|---|---|
| | | Manager | AS Manager | Senior Staff | Junior Staff |
| HR Manager | Manager | Read(own Data) | R/W(Salary edit is denied) | R/W(Salary edit is denied) | R/W(Salary edit is denied) |

| | | | | | |
|---|---|---|---|---|---|
| | AS Manager | - | Read(Own Data) | R/W(Salary edit is denied)(Record deletion is denied) | R/W(Salary edit is denied)(Record deletion is denied) |
| | Senior Staff | - | - | Read(Own Data) | Read |
| | Junior Staff | - | - | | Read(Own Data) |

## 4.2 Authorities of Finance Department

Each user level access grants of Finance Department are clearly defined in Table 4.4. Finance Department manager has highest financial access grant on other levels of users (AS Manager, Senior Staff and Junior Staff). But Finance Department manager has no permission on administrative access grant on all types of users and no granted on HR Manager's financial data accessing (No Read/Write) Figure 4.9. For Finance Department Assistant Manager Level, access grants on other users' level are almost the same as Finance Manager. But, Finance Assistant Manager's access grant on HR Manager and HR Assistant Manager are denied because this system is restricted on lower to higher level and same level data accessing. Then, record deletion is also restricted on others (Higher to Lower and Lower to Higher). Senior Staff of Finance Department has read only permission on own data and financial data of lower levels users. Reading on higher level user financial data (reading on HR Manager's data and HR AS Manager's data) is not allowed. Junior Staff of Finance Department has read only permission on HR Junior Staff's financial data.

**Table 4.4 Granted Permission of Finance Department's Authorize Person on Other Departments**

| Action User | | HR Department | | | |
|---|---|---|---|---|---|
| | | Manager | AS Manager | Senior Staff | Junior Staff |
| Finance Manager | Manager | - | R/W(Salary edit only is allowed) | R/W(Salary edit only is allowed) | R/W(Salary edit only is allowed) |
| | AS Manager | - | - | R/W(Salary edit only is allowed) | R/W(Salary edit only is allowed) |
| | Senior Staff | - | - | - | Read |
| | Junior Staff | - | - | - | Read(Own Data) |

## 4.3 Authorities of Operation Department

Operation Department has the lowest access grant than the two other Departments: HR Department and Finance Department. Operation Department has no administrative access grant and financial access grant on all types of user levels. The Operation department only stands to operate the operational process of office. The Manager of this department has only read accessed permission on own departments' user data and operational data as shown in Figure 4.10. The detailed access grants of the operational department are clearly defined in Table 4.5.

**Table 4.5 Access Permissions of Operation Department**

| Action User | | HR Department | | | | Finance Department | | | | Operation Department | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Manager | AS Manager | Senior Staff | Junior Staff | Manager | AS Manager | Senior Staff | Junior Staff | Manager | AS Manager | Senior Staff | Junior Staff |
| Operation Department | Manager | - | - | - | - | - | - | - | - | Read(Own Data) | Read | Read | read |
| | AS Manager | - | - | - | - | - | - | - | - | | Read(Own Data) | Read | Read |
| | Senior Staff | - | - | - | - | - | - | - | - | - | - | Read(Own Data) | Read |
| | Junior Staff | - | - | - | - | - | - | - | - | - | - | - | Read(Own Data) |

## 4.4 Three-Phase Locking Algorithm

```
BEGIN
Accept the lock request from user.
If (Type of requested Lock = = "Read Lock")
{
Check the type of existing lock on the current requested data;
If (Existing-Transaction.OP-Mode = = R)
{
        Grant (ReadLock);
        Release (ReadLock);
}
```

```
Else If (Existing-Transaction.OP-Mode = =RW)
{
        Grant (ReadLock);
        Release (ReadLock);
}
Else If (Existing-Transaction.OP-Mode = = Write)
{
        Grant (ReadLock); // [except existing Write Lock holding attributes]
Release (ReadLock);

}

}
End If
Else If (Type of requested Lock = = “Write intent Lock”)
{
Check the type of existing lock on the current requested data;
If (Existing-Transaction.OP-Mode = = RW)
{
        Deny (Requested Transaction.OP-Mode);
        Message “Grant for Read Only Mode”;
}
Else If (Existing-Transaction.OP-Mode = = Write)
{
        Deny (Requested Transaction.OP-Mode);
        Message “Grant for Read Only Mode”; // [except existing Write Lock holding
attributes]
}
Else If (Existing-Transaction.OP-Mode = = “Null”)
{
        Grant for requested Write intent Lock;
}
End If
}
 Else If (Type of requested Lock = = “Write Lock”)
```

```
{
If (Write Lock requested user has already hold by Write intent Lock = = Yes)
{
        Grant for requested Write Lock;
}
Else If (Write Lock requested user has already hold by Write intent Lock = = No)
{
        Request Write intent lock;
}
End If
}
END
```

## 4.5 Implementation of the System

This system mentions the three-phase locking algorithm with the combination of attribute level locking by implementing employee management system. In this system, the authorities are divided by each user level in each department respectively. The concurrent transactions may occur because the HR and Finance Department possess the similar power to update the same data items. Although there may be concurrent transaction processing, this system can control the concurrent transaction by using three-phase locking with incorporation of attribute level locking. In the concurrent modes, Table 4.6 and Table 4.7 are explained the lock nature of the proposed system (three-phase locking algorithm).

In the proposed system, there is an organization which has three departments. They are HR Department, Finance Department, and Operation Department. Among them, HR Department and Finance Department have only the update authorities on the same rows. Operation Department has no authority to update the data. They can read data on their department only. In Table 4.8, site1, site2, site3 are represented to user1, user2, and user3. When user1 request read lock at time T1 and user3 request read access on the same row simultaneously and then grant read lock at time T3 for each requested user, respectively. After accessed the read lock, the system automatically release the requested lock of each user at time T4. Once the data is read, the Read lock is dropped. There can be multiple Read locks by multiple users on the

same data/tuple. When the user1 requests to edit the specific attribute of the row at time T6, the system take out the write-intent lock at time T7. After holding the write-intent lock by user1, then no further write-intent locks can be applied on it. But, other user can still obtain the read access. This is one of the advantages of the proposed system. Write Lock allowed only to the user who has the write-intent lock as described in Table 4.8. When the user keeping the write-lock on the definite attribute of the row, the another user can still obtained the read access except the write lock holding attribute. This system can be improved data availability and the degree of concurrency control because of accessing the attribute level lock.

**Table 4.6 Lock Control for Three-Phase Locking**

| | Read-Lock | Write-Intent-Lock | Write-Lock |
|---|---|---|---|
| Read-Lock | Grant | Grant | Deny |
| Write-Intent-Lock | Grant | Deny | Deny |
| Write-Lock | Deny | Deny | Deny |

**Table 4.7 Lock Control for Each Transaction**

| Tuple | Transaction A | Transaction B | Lockable status |
|---|---|---|---|
| Tuple 1 | Read-Lock | Read-Lock | Ok |
| Tuple 1 | Write-Intent-Lock | Read-Lock | Ok |
| | Write-Intent-Lock | Write-Intent-Lock | Deny |
| Tuple 1 | Write-Lock | Read-Lock | Deny |
| | Write-Lock | Write-Intent-Lock | Deny |
| | Write-Lock | Write-Lock | Deny |

**Table 4.8 Concurrency Control in the Same Data Item**

| Time | Site 1 | Site 2 | Site 3 |
|---|---|---|---|
| T1 | Request(A) | | Request(A) |
| T2 | Read-Lock (A) | | Read-Lock (A) |
| T3 | Read(A) | Request(A) | Read(A) |
| T4 | Release-Read-Lock(A) | Read-Lock (A) | Release-Read-Lock(A) |
| T5 | - | Read(A) | - |
| T6 | Request for Write-intent Lock | Release-Read-Lock(A) | Request for Write-intent Lock |
| T7 | Write-intent Lock(A.Attribute1) | - | Write-intent Lock(A.Attribute2) |
| T8 | Update(A.Attribute1) | Request for Write-intent Lock | Update(A.Attribute2) |
| T9 | - | Wait (Write-intent Lock(A.Attribute1)) | - |
| T10 | Finishes editing the data | - | Finishes editing the data |
| T11 | Request for Write Lock | - | Request for Write Lock |
| T12 | Release for Write-intent Lock | - | Release for Write-intent Lock |
| T13 | Write Commit(A.Attribute1) | - | Write Commit(A.Attribute2) |
| T14 | Release Write Lock(A.Attribute1) | - | Release Write Lock(A.Attribute2) |
| T15 | Propagate Update A.Attribute1 to other sites | - | Propagate Update A.Attribute2 to other sites |
| T16 | | Get Updated data(A.Attribute1) | |
| T17 | | Write-intent Lock(A.Attribute1) | |
| T18 | | Update(A.Attribute1) | |
| T19 | Request(A.Attribute2) | - | |
| T20 | Read-Lock (A.Attribute2) | Finishes editing the data | |
| T21 | Read(A.Attribute2) | Request for Write Lock | Request(A.Attribute3) |
| T22 | Release-Read-Lock(A.Attribute2) | Release for Write-intent Lock | Read-Lock (A.Attribute3) |
| T23 | | Write Commit(A.Attribute1) | Read(A.Attribute3) |
| T24 | | Release Write Lock(A.Attribute1) | Release-Read-Lock(A.Attribute3) |
| T25 | | Propagate Update A.Attribute1 to other sites | |

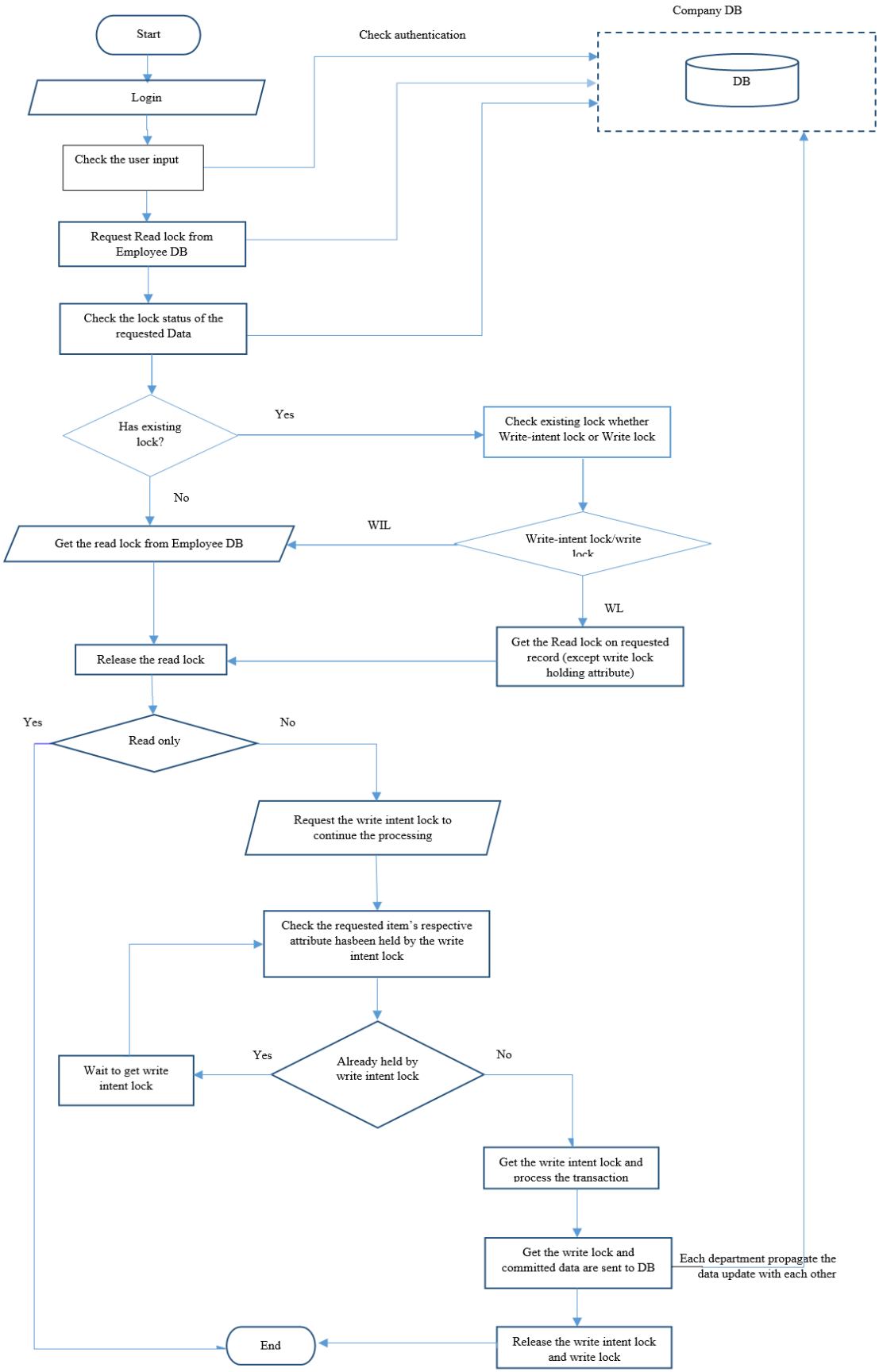

Company DB
Start
Check authentication
DB
Login
Check the user input
Request Read lock from Employee DB
Check the lock status of the requested Data
Has existing lock?
Yes
Check existing lock whether Write-intent lock or Write lock
No
WIL
Get the read lock from Employee DB
Write-intent lock/write lock
WL
Release the read lock
Get the Read lock on requested record (except write lock holding attribute)
Yes
Read only
No
Request the write intent lock to continue the processing
Check the requested item's respective attribute hasbeen held by the write intent lock
Wait to get write intent lock
Yes
Already held by write intent lock
No
Get the write intent lock and process the transaction
Get the write lock and committed data are sent to DB
Each department propagate the data update with each other
End
Release the write intent lock and write lock

**Figure 4.1 System Flow Diagram of the Propose System**

## 4.6 The Login Page of the Proposed System

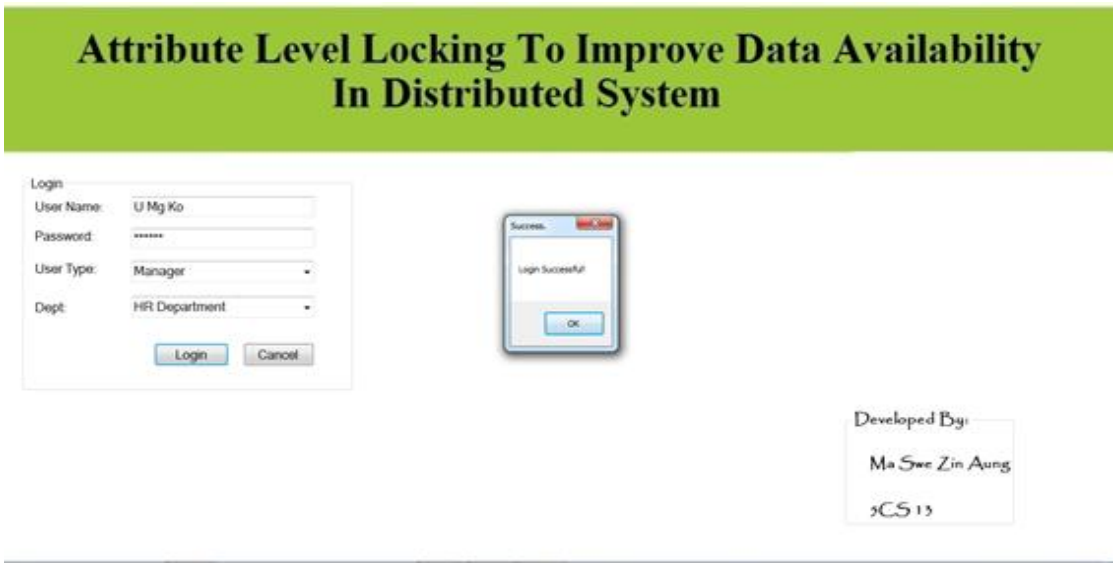



**Figure 4.2 Login Page of the System**

The proposed system login page is shown in Figure 4.2. The user level processing grant is managed by the user level based on role base. At the login phase, the invalid user insert to the system is restricted in Figure 4.3.

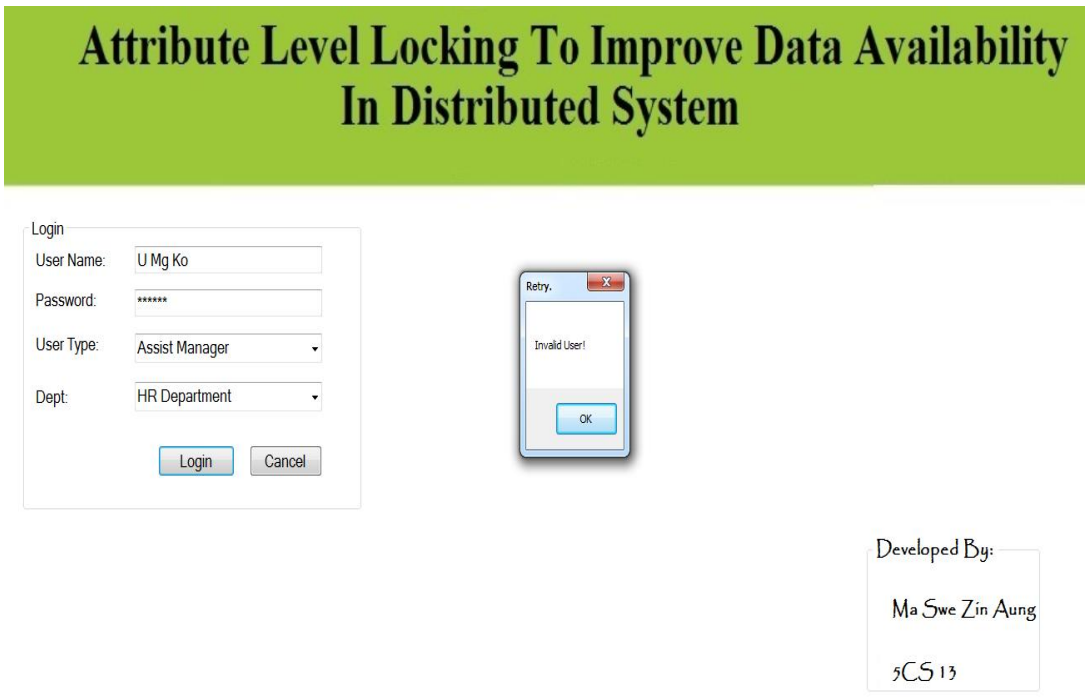



**Figure 4.3 Login Page of the System with Invalid User**

## 4.7 Main Page of the Proposed System


**Figure 4.4 Main Page of the System**

In the main page, the process of this page is performed by role. If the registration user is HR Department, it can access the staff registration and most of the other authorities can get. In Finance Department, it can only access on salary field. In Operation Department, all users in this department can only read access.

## 4.8 Staff Registration

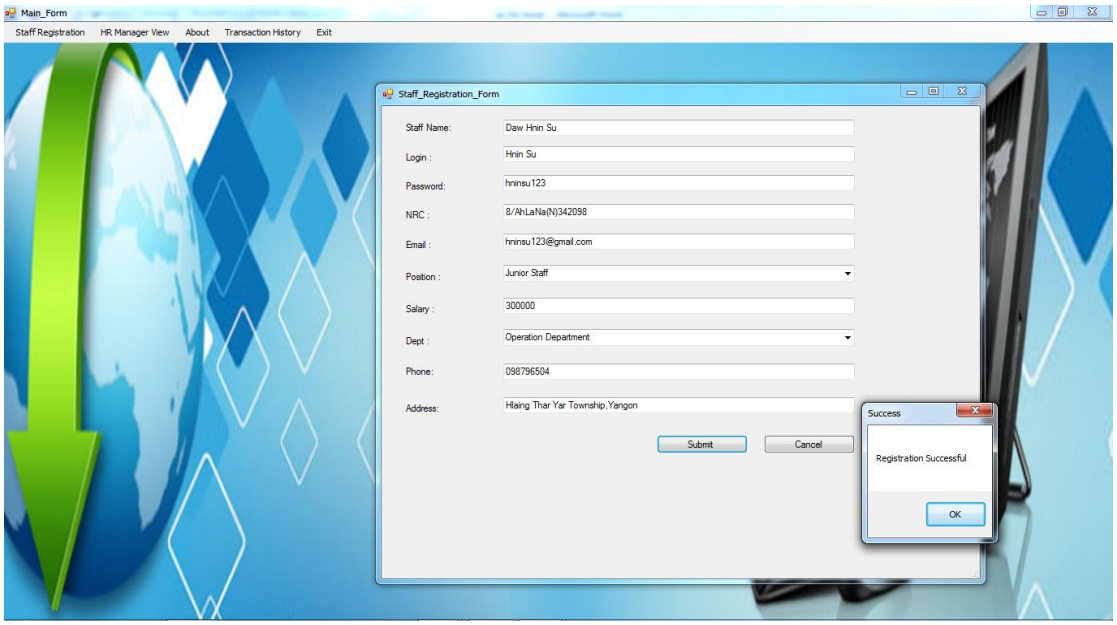



**Figure 4.5 Staff Registration Form**

The staff registration processes is only granted for HR Department. HR Department can be done the new staff registration for all departments (i.e.HR Department, Finance Department and Operation Department).

## 4.9 The Detail Process of HR Department

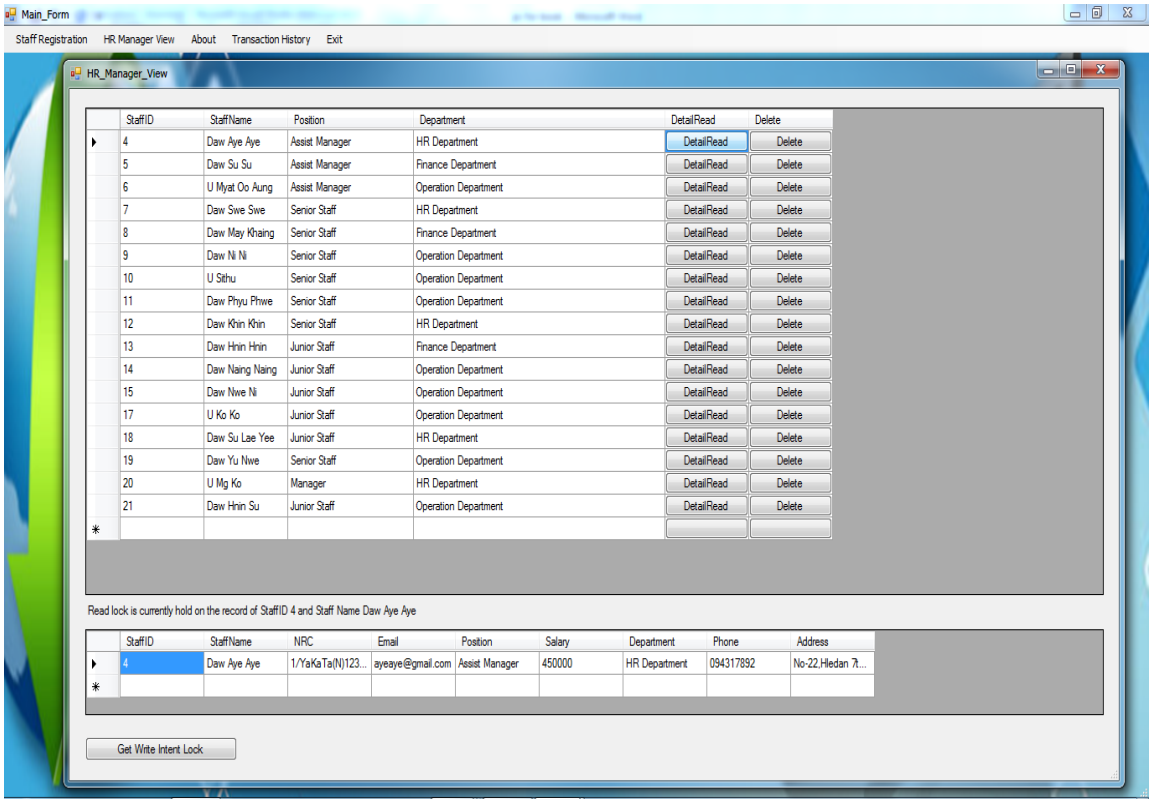



**Figure 4.6 HR Manager View**

HR Department of manager only has the delete permission to all departments of data items as shown in Figure 4.5. Firstly, press the detail read button to get the read lock. After the read lock requested on roll no: 4, the read lock is automatically release. So, the other users can access the multiple read accesses on the same row. When user wants to update the detail data, firstly write intent lock take out and other user can still obtained a read lock on the same data. If data locked with Write Intent Lock, then no further Write intent locks can be obtained on the same data item.

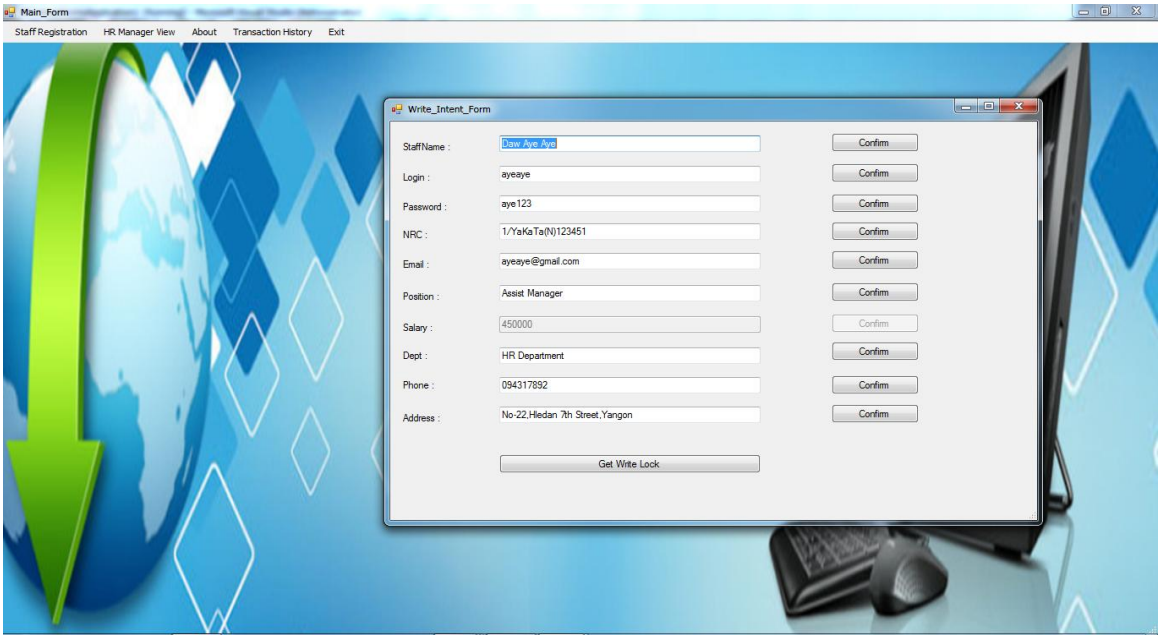



**Figure 4.7 HR Manager's Access Grant**

The salary column is restricted in HR Department because the attribute level locks control for each attributes as shown in Figure 4.6. Finance Department of authorize person can get the read access on the same row simultaneously. So, this situation prevents deadlock conditions to improve the data availability and the degree concurrency control.

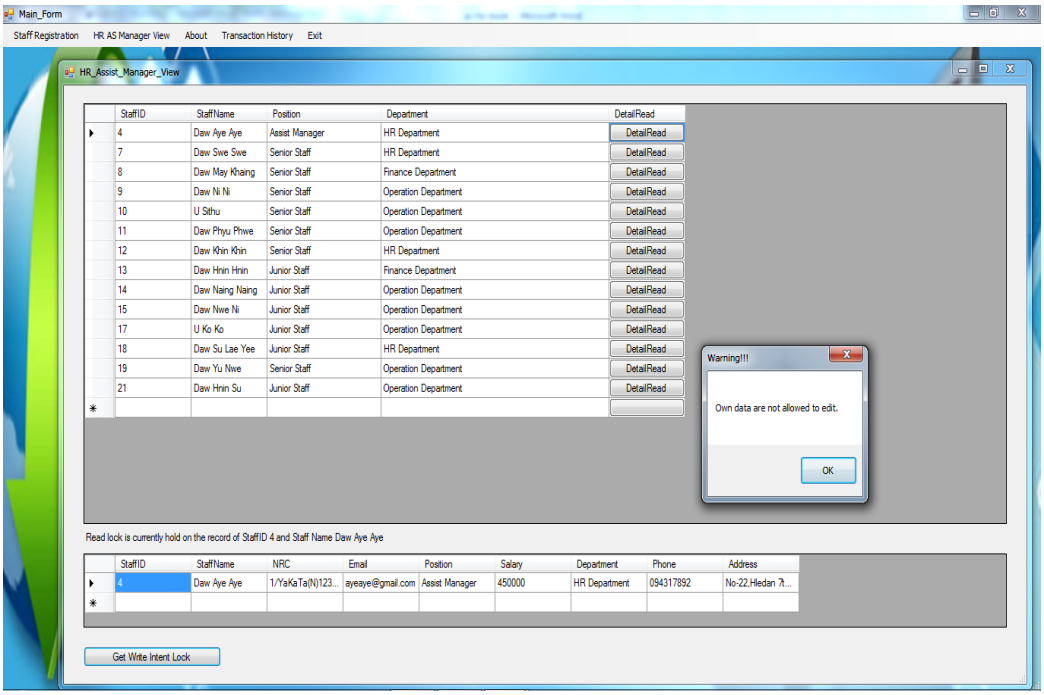



**Figure 4.8 Restriction on Own Data Editing [HR AS Manager]**

Own data editing is also restricted as shown in Figure 4.6. Then, record deletion is also restricted on others levels (Higher level to Lower level, same level and Lower to Higher) as shown in Figure 4.7.

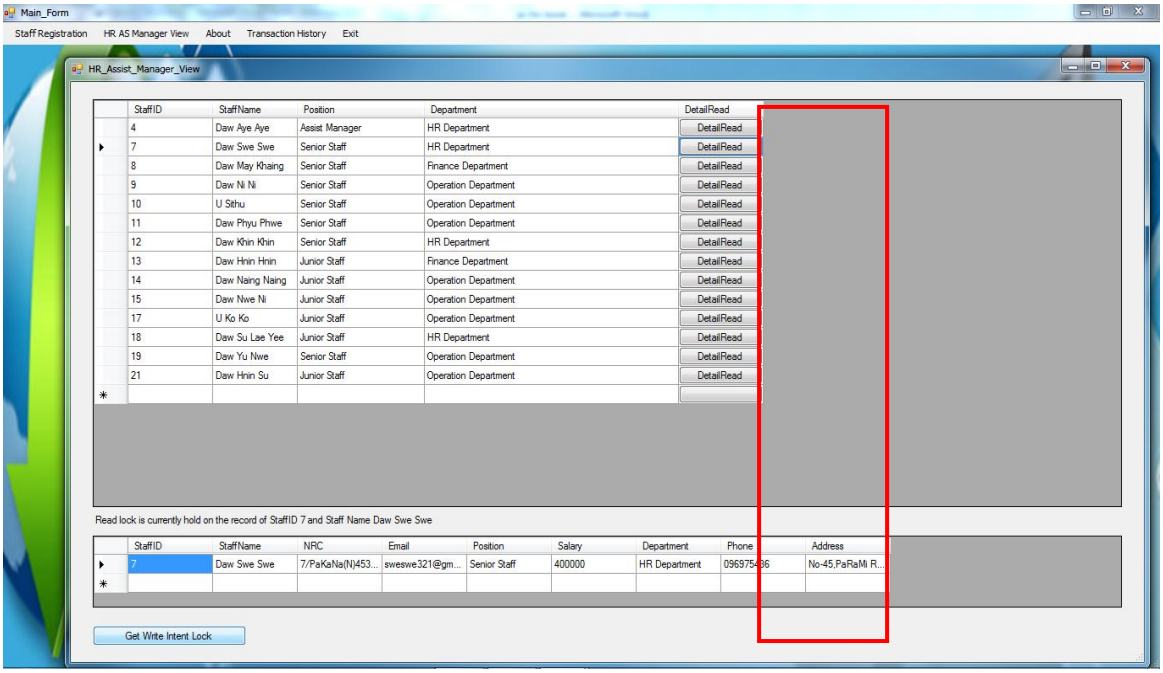



**Figure 4.9 Record Deletion Permission is not allowed on Assistant Manager Level in HR Department**

The record deletion permission is not allowed in Assistant Manager levels of any departments. In Figure 4.8, the delete button is not including because the system login user is Assistant Manager level of HR Department.

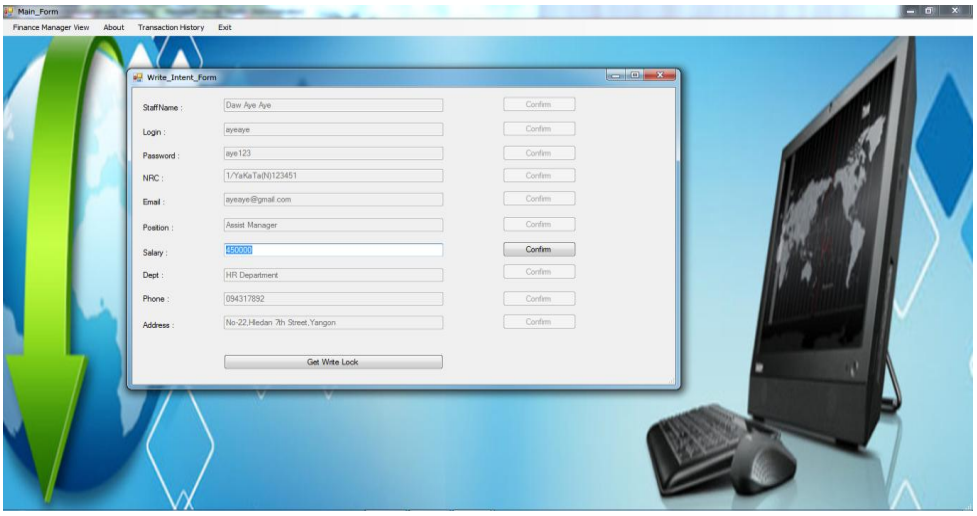



**Figure 4.9 Access Grant of Financial Data Editing of Finance Department**

In Finance Department, the access grant is only allowed on salary field because the attribute level locking mechanism is used. So, the system improves data availability and the degree of concurrency control.

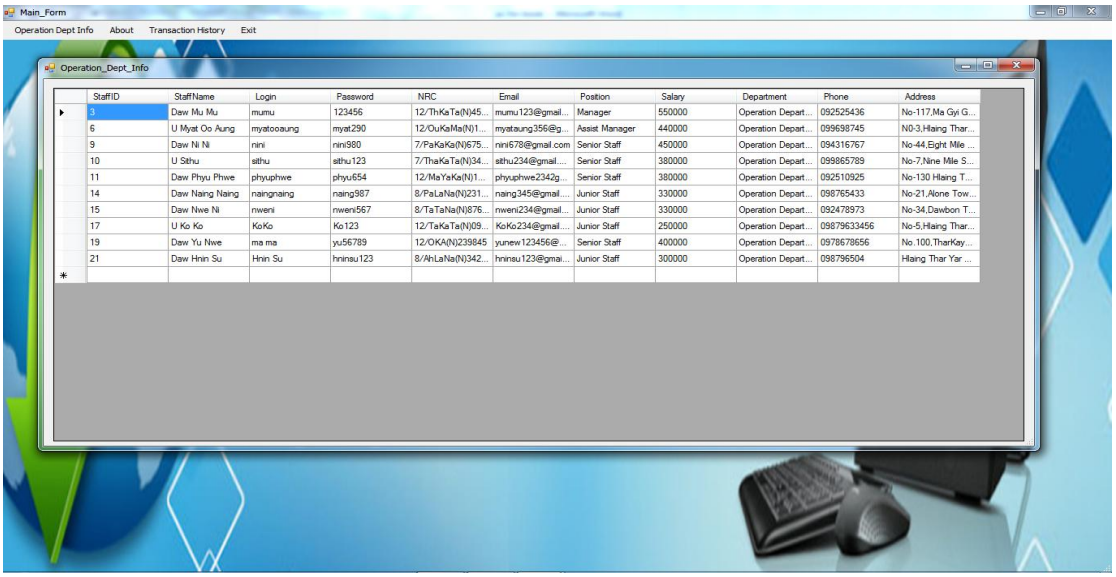



**Figure 4.10 Read Only Permission of Operation Department**

In the Operation Department, all types of user level can only access the read mode by the role level restriction as shown in Figure 4.10.

## 4.10 Concurrency Controlling in Same Attribute

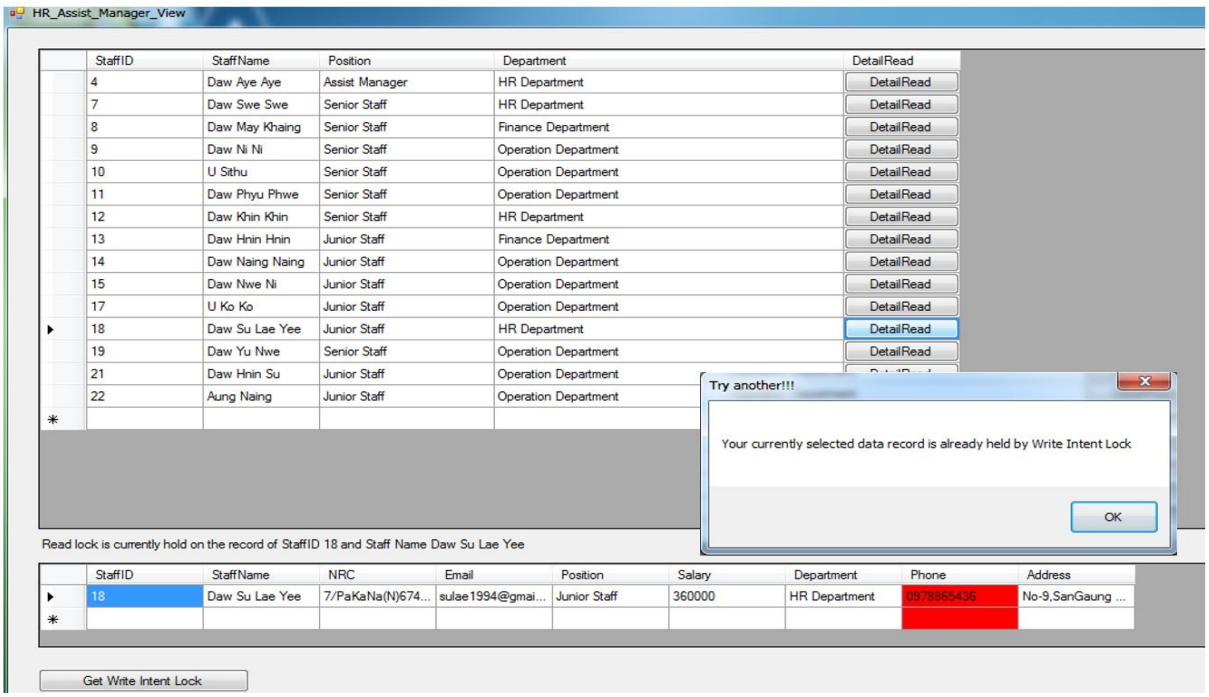



**Figure 4.11 Notifications for Concurrent Accessing on Same Attribute**

This system not only the multilevel user access control but also control the same attribute concurrent accessing. As shown in figure 4.11, more than one users are accessing the same attribute, this system will deny for the lately request user by showing the notification. After the early user transaction accessing is committed, the data updates are applied to the respective user for the updated data item as shown in figure 4.12.

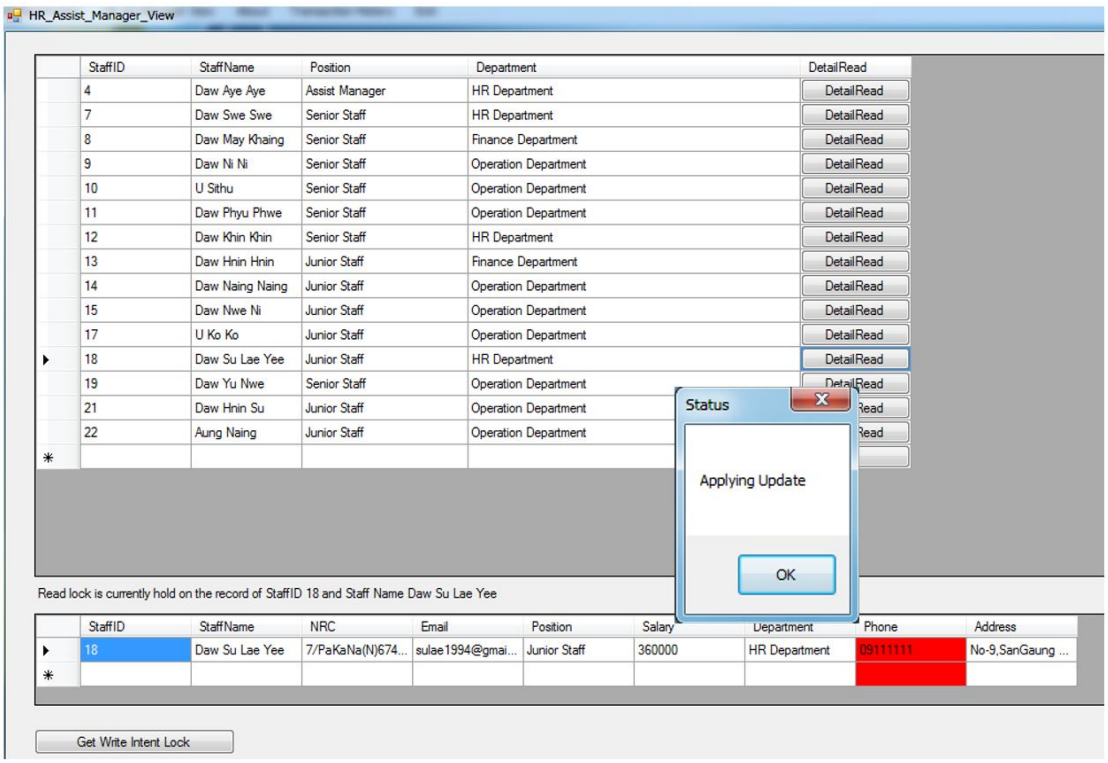



**Figure 4.12 Data Update Applying**

# CHAPTER 5

# CONCLUSION, LIMITATIONS AND FURTHER EXTENSIONS

Distributed systems are considered as crucial sources of information. Therefore, data contained in such systems must be available at all times as much as possible to satisfy the user's professional needs. To increase the data availability, this system proposes a new adaptive approach to increase the database items by reducing the size of the lockable units. This reduction can be carried out by locking the attributes instead of the database row, which remains as the other attributes become available for other transactions. The attribute-level locking increases the degree of concurrency by increasing the data availability. The overall system performance is also improved because the average waiting time is decreased. The increasing overhead is managed by returning the lock at the row level when a transaction requires many attributes of the same row. The proposed approach is suitable for short transactions of mixed read and writes operations, especially when the degree of replication is less than 50%.

## 5.1 Benefits of using Attribute Level Locking

1. This system can prevent deadlocks occurrences because additional locking mechanisms are unnecessary.
2. The blocking problem found in two-phase locking (2PL) can be avoided in the proposed system.
3. The locks are incrementally and dynamically acquired, so the transaction may request an attribute of the row while the other attributes may be available or acquired by another transaction.
4. Using attribute-level locking increase data availability, reliability, and throughput, as well as enhance overall system performance.
5. In any situations, multiple users can read the data without blocking because of using attribute level locking.

## 5.2 Limitations and Further Extension

By using the proposed system, multiple users can access the same data without conflicting. So, several transactions can manage on the same row simultaneously because this system uses attribute level locking. This system may increase the database resources, which would increase the concurrency and throughput in the system and decrease deadlock occurrences. In contrast, the overhead may be increased. The system overhead case should be reduced as the further study. This system is only managed for an office and not configured for distributed branches of the organizations. Further work for studying the proposed system could be implemented by having more sites, larger data set and a higher workload, as well as more practical examples, experiments and comparison with other technologies will also be studied as a future work to improve the quality of the research.

# AUTHOR'S PUBLICATION

[1] Swe Zin Aaung, Daw Khaing, "Attribute Level Locking to Improve Data Availability in a Distribute System", National Journal of Parallel Soft Computing (NJPSC 2020), Yangon, Myanmar, 2020.

# REFERENCES


[1] M. Atif, "Analysis and Verification of Two-Phase Commit &Three-Phase Commit Protocols", International Conference on Emerging Technologies (ICET), Islamabad, 19-20 Oct. 2009

[2] P. Bernstein and E. Newcomer, "Principles of Transaction Processing for the Systems Professional", 2nd edition, 2009

[3] K. Chandy, J. Misra and L. Hass, "Distributed Deadlock detection, ACM Transactions on Computer Systems", University of Texas, Vol. 1, No. 2, May 1983, Pages 144-156

[4] A. Croker, "Improvements in Database Concurrency Control with Locking", Journal of Management Information Systems; Vol. 4 Issue 2, 2001

[5] R. Elmasri and S. Nava, "Fundamentals of Database Systems", Pearson Addison Wesley, 7th edition, 2015

[6] J. M. Faleiro and D. J. Abadi, "A Distributed Database Performance Tradeoff", IEEE Data Engineering Bulletin, 38(1): 10-17, 2015

[7] J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques", 2nd edition, **July 2009**

[8] K. Norvag, O. Sandsta, and K. Bratbergsengen, "Concurrency Control in Distributed Object-Oriented Database Systems", Advances in Databases and Information Systems, 1997

[9] K. Maabreh and A. Hamami, "Increasing Database Concurrency Control based on Attribute Level Locking", International Conference on Electronic Design (ICED), December. 2008

[10] K. Maabreh and A. Hamami, "Implementing New Approach for Enhancing Performance and Throughput in a Distributed Database", The International Arab Journal of Information Technology, Vol. 10, No. 3, May 2013

[11] N. Matthias and J. Matthias, "Performance Modeling of Distributed and Replicated Databases", IEEE transactions on knowledge data engineering, Vol.12 No.4, pp. 645-672, July 2000

[12] Myat Nandar Oo, "Implementation of Basic 2PL Method in Locking Technique for Concurrency Control", M.C.Sc Thesis, University of Computer Studies, Yangon, January 2011

[13] T. Ozsu and P. Valduriez, ”Principles of Distributed Database Systems”, Springer science and business, 3rd edition, 2011

[14] A. Silberschatz, H. Korth and S. Sudarshan, “Database System Concepts”, 6th edition, 2010