# CONCURRENCY CONTROL ON TRANSACTIONAL REPLICATION

**SAINT SAINT SAN**

**M.C.Sc.** **SEPTEMBER 2022**

# CONCURENCY CONTROL ON TRANSACTIONAL REPLICATION

BY

SAINT SAINT SAN

B.C.Sc. (Hons.)

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Computer Science

(M.C.Sc.)

University of Computer Studies, Yangon

SEPTEMBER 2022

# Acknowledgements

I thank all my teachers who taught and help me during the period of studies in University of Computer Studies Yangon.

I would like to express my gratitude and my sincere thanks to **Dr. Mie Mie Khin,** Rector, University of Computer Studies, Yangon, for great suggestion and kind permission to develop this thesis.

My heartfelt thanks and respect go to **Dr. Si Si Mar Win, Professor** and Dean of Master Degree Course, and **Dr. Tin Zar Thaw** Faculty of Computer Science, and University of Computer Studies, Yangon, for her valuable guidance and administrative support throughout the development of the thesis.

I am deeply thankful to my supervisor, **Daw Kyi Kyi Win**, Associate Professor, Faculty of Computing Mathematics, University of Computer Studies, Yangon, for her valuable guidance, encouragement, superior suggestion and supervision on the accomplishment of this thesis.

I would like to thank **Daw Hnin Yee Aung**, Lecturer, Department of English, University of Computer Studies, Yangon, for editing my thesis from the language point of view.

Finally, I am grateful to my colleagues and all my friends for their cooperation and help for the completion of 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 | Saint Saint San |

# ABSTRACT

Nowadays, commerce is larger and larger and it affects to open new branch offices in different locations. Therefore, it is important to handle the transaction (Write/Update) of each branch by the head office. There are two databases: original database and replica /(Key/value store) database. When the original database handles read/write transactional application workloads while the copy information base handles read-just responsibilities from similar applications over the information recreated from the first data set. The principal necessity is guaranteeing the utilization of the reports on the copy data set in precisely the same request they were executed in the first data set, which is called execution-defined request. The essential server executes the activities and sends duplicates of the refreshed information to the copies. This framework presents a novel concurrency control algorithm to solve the concurrency problem in the hotel reservation system. This system is implemented using C# programming language with Microsoft SQL server database engine.

## Contents

LIST OF FIGURES

# CHAPTER 1
# INTRODUCTION

Replication is mainly used to store some (or all) data items redundantly at multiple sites. Its goal is to increase system reliability. Databases are widely used by enterprises as the preferred storage media for their data and their management. Thus, storing data at multiple sites allows the system to continue working even though some sites may have failed. A replicated middleware architecture providing object state persistence may be viewed as a replicated object database providing persistence. All these advantages are not for free, replication has some problems, such as data consistency and fault-tolerance.

The system must introduce an additional overhead for maintaining replicated data consistency. Applications must introduce additional software in order to access distributed resources, thus increasing application development complexity. Data consistency is granted by a particular consistency protocol. They may be eager, if update propagation takes place at commit time (to all alive nodes or the primary copy), or lazy, if it happens on demand of a node requesting data, using pull and push strategies. All these combinations provide a set of consistency protocols that features a set of advantages and drawbacks that greatly depends on the kind of application used.

Besides data access is performed concurrently among several users, usually in a transactional manner, thus this consistency protocols must guarantee the transaction isolation. One of the key issues of replicated architectures, as it has been previously highlighted, is data availability. The system must continue accomplishing its tasks, even though a node fails. Group membership monitors are used to detect node failures or network partitions. Steps to be done when a node fails vary from system reconfiguration after the failure, passing by partition merge to bring data "*up-to-date" after it* recovers from a crash by a previously alive node.

For this concurrency protocol, data replication proposal will not complete unless not providing a recovery protocol. Reconfiguration is needed when the number of sites increases is a far more complex task than that necessary when the number of nodes decreases. In particular, before a node can execute transactions, an "*up-to-date" node has to provide the current* data state to the joining node.

Increasingly more associations are utilizing various information base as opposed to attempting to fit one data set to all information the executive's needs. The explanation is to lessen responsibility on single expert information base. The conditional updates in the first data set due to compose exchange, are transported to the key-esteem store and applied in a similar request to ensure the right state for the copy, which is called execution-defined request. Exchanges might interleave during their execution against the first data set. The replication ought to ensure that the subsequent serialization request for exchanges in the reproduction is the very same as the serialization request in the first data set and no other serialization request is satisfactory.

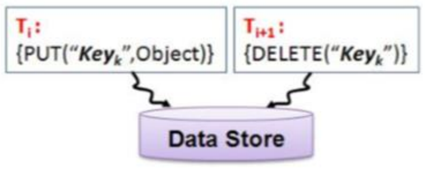


**Figure 1: Serialization Order on System State**

If $T_i$ is executed before $T_{i+1}$ then the data item (Key $_k$) will not exist in the data store. On the other hand, if $T_{i+1}$ is executed before $T_i$ the data item will exist in the data store [figure1]. Subsequently, the subsequent execution is not right according to serialization perspective, the subsequent execution isn't satisfactory since it doesn't bring about the right state considering the predefined execution request.

## 1.2 Objective of Thesis

- The objectives of our thesis are as follows:
- To increase system reliability
- To implement middleware architecture that to provide object state persistence may be viewed as a replicated object database providing persistence
- To study data replication techniques and their infrastructure
- To reduce the work load on single server
- To provide anytime anyplace service for the customer

- To study data replication technique and infrastructure.
- To reduce the work load on single server.
- To present a replication system that is to exploits concurrent execution while guaranteeing the execution-defined order and allowing read-only application workloads on the replica database.
- In order to distribute database load to different database servers rather having a single mainframe server than the system replicate data.
- To allow various sites (replicas) to work autonomously and at a later time updates into a single, multicast the uniform result to each replica.

## 1.2 Related Works

The related works of the system are discussed in this session.

In the development of applications for companies that have several branch offices, such as banks, hypermarkets, etc. In such settings, several applications typically use on-site generated data in local branches, while other applications also use information generated in other branches and offices. The services provided by COPLA enable an efficient catering for both local and non-local data querying and processing [1].

Current work in consistency protocols for repli- cated databases can be found using either eager (Agrawal et al.; Kemme and Alonso; Wiesmann et al.,) or lazy protocols (Breitbart et al.; Ferrandina et al.; Mun˜oz-Esco´ı et al [2].). Each has its pros and cons, as described in (Gray et al.). Eager protocols usually hamper the update performance and increase transaction response times but, on the positive side, they can yield serializable execution of multiple transactions with- out requiring too much effort. On the other hand, lazy protocols may answer read requests by stale data versions (or at least they require extra work to avoid that), but they improve transaction response times and allow disconnected operation.

Although COPLA provides a frameworkable to accept different consistency protocols (indeed, lazy protocols have also been designed for this environment (Mun˜oz-Esco´ı et al.,)), the presented approach uses an eager replication alternative. A good classification of eager protocols is presented in (Wiesmann et al.), according to three parameters: server architecture (primary copy vs. update everywhere), server interaction (constant vs. linear) and transaction termination (voting vs. non-voting). Among the eight alternatives resulting from combining these three parameters, only two of them seem to lead to a good balance of scalability and efficiency: those based on “up- date everywhere” and “constant interaction”. This is mainly due to the load distribution achievable with the “update everywhere” approach, i.e., a delegate server executes the transaction and broadcasts the changes everywhere. The election of such a delegate server is dynamic. Each transaction can choose a different delegate. Moreover, low communication costs result from a “constant interaction”, where the update broadcast is done just once, either at the beginning or end of the transaction, rather than in each transactional operation, as is the case in the “linear interaction” approach.

## 1.3 Organization of Thesis

This thesis is organized in five chapters. They are as follows:

**In Chapter 1**, introduction of concurrency control, objectives of the thesis and thesis organization are described.

**In Chapter 2**, presents the background theory of the replication system.

**In Chapter 3**, discusses Concurrency Control and Recovery in a Middleware Replication Software Architecture.

**In Chapter 4**, expresses the design and implementation of the proposed system.

Finally, **Chapter 5** presents the conclusions of this thesis, and showing advantages.

# CHAPTER 2
# THEORICAL BACKGROUND

Imagine a scenario that if all the company’s staff will use to perform different tasks, this application is developed. Each person has a laptop and will be connected to the company’s network. This type of application can be developed in two different ways.

One of those is the traditional approach of separating the tables from the other objects in the database so that the data can reside in a back-end database on a network server, or on the Internet or an intranet, while the queries, forms, reports, macros, and modules reside in a separate front-end database on the user's computer. The objects in the front-end database are based on tables that are linked to the back-end database. When users will retrieve or update information in the database, they use the front-end database.

By creating a single database that contains both the data and objects, the second way enables that is to take a new approach to building this solution. Using Database replication, you can then make replicas of the database for each user and synchronize each replica with the Design Master on a network server. In this scenario, can choose to replicate only a portion of the data in the Design Master, and it can replicate different portions for different users by creating partial replicas. By using partial replicas, it can duplicate only the data that each user actually needs. A complete set of data is still contained in the Design Master, but each replica handles only a subset of that data. The Design Master is the first member in a replica set and it is used in the creation of the first replica in a replica set. This can make changes to the database structure only with the Design Master. Replicas in the same replica set can take turns being the Design Master, but there can be only one Design Master at a time in each replica set. [1]

## 2.1. The concept of Replication

To better understand the method behind Database Replication, it can start the term “Replication” which represents the process of sharing information to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility. It could be data replication if the same data is stored on multiple storage devices, or computation replication if the same computing task is executed many times. The access to a replicated entity is typically uniform with access to a single, non-replicated entity. The replication itself should be transparent to an external user. In addition, in a failure scenario, a failover of replicas is hidden as much as possible. In systems that replicate data the replication itself is either active or passive.

In an active replication when the same request is processed at every replicated instance and about passive replication when each request is processed on a single replica and then its state is transferred to the other replicas. If at any time one master replica is designated to process all the requests, then we are talking about the primary-backup scheme (master-slave scheme) predominant in high-availability clusters. On the other side, if any replica processes a request and then distributes a new state, then this is a multi-primary scheme (called multi-master in the database field). [2] Even thought, the process of Data Replication it’s used to create instances of the same or parts of the same data, we must not confuse it with the process of backup since replicas are frequently updated and quickly lose any historical state. Backup on the other hand saves a copy of data unchanged for a long period of time.

## 2.2. What Database Replication is

Database replication is the process of creating and maintaining multiple instances of the same database and the process of sharing data or database design changes between databases in different locations without having to copy the entire database.

In most implementations of database replication, one database server maintains the master copy of the database and the additional database servers maintain slave copies of the database. The two or more copies of a single database remain synchronized. [3] The original database is called a Design Master and each copy of the database is called a replica. Together, the Design Master and the replicas make up a replica set. There is only one Design Master in a replica set. Synchronization is the process of ensuring that every copy of the database contains the same objects and data. When the data is synchronized in a replica set, only the data that has changed is updated.

In the Design Master, the synchronize is changed that is make to the design of the objects [1] Database writes are sent to the master database server and are then replicated by the slave database servers. Database reads are divided among all of the database servers, which results in a large performance advantage due to load sharing. In addition, database replication can also improve availability because the slave database servers can be configured to take over the master role if the master database server becomes unavailable. [3].

## 2.3. When to choose Database Replication

Implementing and maintaining replication might not be a simple proposition. If there are numerous database servers that need to be involved in various types of replication, a simple task can quickly become complex.

Implementing replication can also be complicated by the application architecture. However, there are numerous scenarios in which replication can be utilized. [4]

Database replication is well suited to business solutions that need to:

- **Share data among remote offices**

Database replication is used to create copies of a corporate database to send to each satellite office across a wide area network (WAN). Each location enters data in its replica, and all remote replicas are synchronized with the replica at corporate headquarters. Individual replicas can maintain local tables that contain information not included in the other replicas in the set.

- **Share data among dispersed users**

  New information that is entered in the database while users are out of the office can be synchronized any time the users establish an electronic link with the corporate network. As part of their workday routine, users can dial in to the network, synchronize the replica, and work on the most current version of the database. Because only the incremental changes are transmitted during synchronization, the time and expense of keeping up-to-date information are minimized. By using partial replicas, it can synchronize only specified parts of the data.

- **Make server data more accessible**

  If the solution does not need to have immediate updates to data, the use of database replication is to reduce the network load on the primary server. Introducing a second server with its own copy of the database improves response time. In the schedule for synchronizing the replicas, and it can adjust that schedule to meet the changing needs of the users. Replication requires less centralized administration of the database while offering greater access to centralized data.

- **Distribute solution updates**

  When the solution is replicated, it automatically replicates not only the data in the tables but also in the solution's objects. If there are changes to the design of the database, the changes are transmitted during the next synchronization; so, it does not need to distribute complete new versions of the software.

- **Back up data**

  At first glance, database replication might appear to be very similar to copying a database. However, while replication initially makes a complete copy of the database, thereafter it simply synchronizes that replica's objects with the original objects at regular intervals. This copy can be used to recover data if the original database is destroyed. Furthermore, users at any replica can continue to access the database during the entire backup process.

- **Provide Internet or intranet replication**

  For propagating changes to participating replicas, it needs to configure an Internet or intranet server to be used as a hub [1]

## 2.4. When Database Replication should not be used

Although database replication has many benefits and can solve many problems in distributed-database processing, it should be recognized the fact that in some situations replication is less than ideal. Database Replication is not recommended if:

**There are frequent updates of existing records at multiple replicas**

Solutions that have a large number of record updates in different replicas are likely to have more record conflicts than solutions that simply insert new records in a database. If changes are made to the same record by different users and at the same time then record conflicts will definitely appear. This can be real time consuming because the conflicts must be resolved manually.

- **Data consistency is critical at all times**

Solutions that rely on information being correct at all times, such as funds transfers, airline reservations, and the tracking of package shipments, usually use a transaction method. Although transactions can be processed within a replica, there is no support for processing transactions across replicas. The information exchanged between replicas during synchronization is the result of the transaction, not the transaction itself.

## 2.5. Methods of performing Database Replication

Database replication can be performed in at least three different ways:

- **Snapshot replication:** Data on one database server is plainly copied to another database server, or to another database on the same server.
- **Merging replication:** Data from two or more databases is combined into a single database.
- **Transactional replication:** Users obtain complete initial copies of the database and then obtain periodic updates as data changes.

### 2.5.1. Snapshot replication

This type of Database Replication is one of the simplest method to set up, and perhaps the easiest to understand.

The snapshot replication method functions by periodically sending data in bulk format. Usually it is used when the subscribing servers can function in read- only environment, and also when the subscribing server can function for some time without updated data. Functioning without updated data for a period of time is referred to as latency.

For example, a retail store uses replication as a means of maintaining an accurate inventory throughout the district. Since the inventory can be managed on a weekly or even monthly basis, the retail stores can function without updating the central server for days at a time. This scenario has a high degree of latency and is a perfect candidate for snapshot replication.

Additional reasons to use this type of replication include scenarios with low-bandwidth connections. Since the subscriber can last for a while without an update, this provides a solution that is lower in cost than other methods while still handling the requirements.

Snapshot replication also has the added benefit of being the only replication type in which the replicated tables are not required to have a primary key. Snapshot replication works by reading the published database and creating files in the working folder on the distributor. These files are called snapshot files and contain the data from the published database as well as some additional information that will help create the initial copy on the subscription server. [5]

Snapshot replication is often used when needing to browse data such as price lists, online catalogs, or data for decision support, where the most current data is not essential and the data is used as read-only.

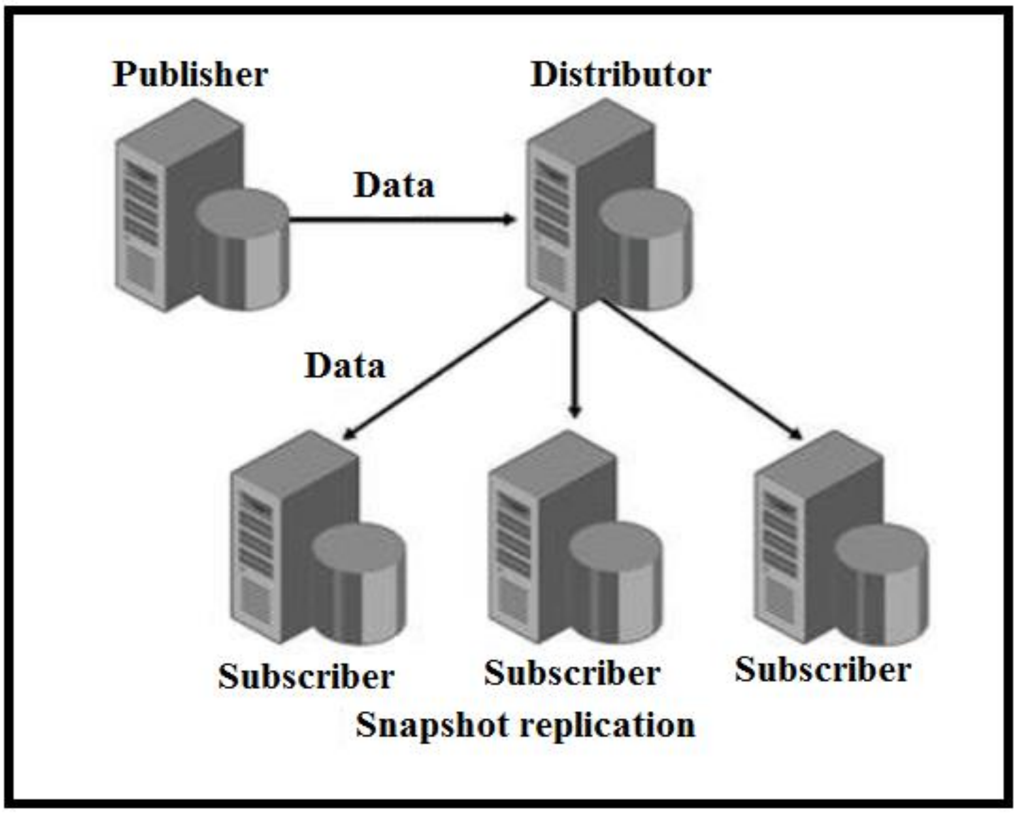


**Figure 2.1. Snapshot Replication**

**Snapshot replication is helpful when:**

- Data is mostly static and does not change often.
- It is acceptable to have copies of data that are out of date for a period of time.
- Replicating small volumes of data in which an entire refresh of the data is reasonable.

### 2.5.2. Merging replication

Merge replication is the process of distributing data from Publisher to Subscribers, allowing the Publisher and Subscribers to make updates while connected or disconnected, and then merging the updates between sites when they are connected.

Merge replication allows various sites to work autonomously and at a later time merge updates into a single, uniform result. The initial snapshot is applied to Subscribers, and then changes are tracked to publish data at the Publisher and at the Subscribers. The data is synchronized between servers continuously, at a scheduled time, or on demand. Because updates are made at more than one server, the same data may have been updated by the Publisher or by more than one Subscriber. Therefore, conflicts can occur when updates are merged.

Merge replication includes default and custom choices for conflict resolution that you can define as you configure a merge publication. When a conflict occurs, a resolver is invoked by the Merge Agent and determines which data will be accepted and propagated to other sites.

**Merge Replication is helpful when:**

Multiple Subscribers need to update data at various times and propagate those changes to the Publisher and to other Subscribers.

- Subscribers need to receive data, make changes offline, and later synchronize changes with the Publisher and other Subscribers.
- When data is updated at multiple sites, it will not expect many conflicts (because the data is filtered into partitions and then published to different Subscribers or because of the uses of application). However, if conflicts do occur, violations of **ACID** properties are acceptable. [1]

### 2.5.3. Transactional replication

In what could be considered the opposite of snapshot replication, transactional replication works by sending changes to the subscriber as they happen. As an example, SQL Server processes all actions within the database using Transact-SQL statements. Each completed statement is called a transaction.

In transactional replication, each committed transaction is replicated to the subscriber as it occurs. The replication process is controlled so that it will accumulate transactions and send them at timed intervals or transmit all changes as they occur. In having a lower degree of latency and higher bandwidth connections, it is used this type of replication in environments Transactional replication requires a continuous and reliable connection, because the Transaction Log will grow quickly if the server is unable to connect for replication and might become unmanageable. Transactional replication begins with a snapshot that sets up the initial copy. That copy is then later updated by the copied transactions. It can choose how often to update the snapshot or choose not to update the snapshot after the first copy.

Once the initial snapshot has been copied, transactional replication uses the Log Reader agent to read the Transaction Log of the published database and stores new transactions in the DISTRIBUTION Database. The Distribution agent then transfers the transactions from the publisher to the subscriber.

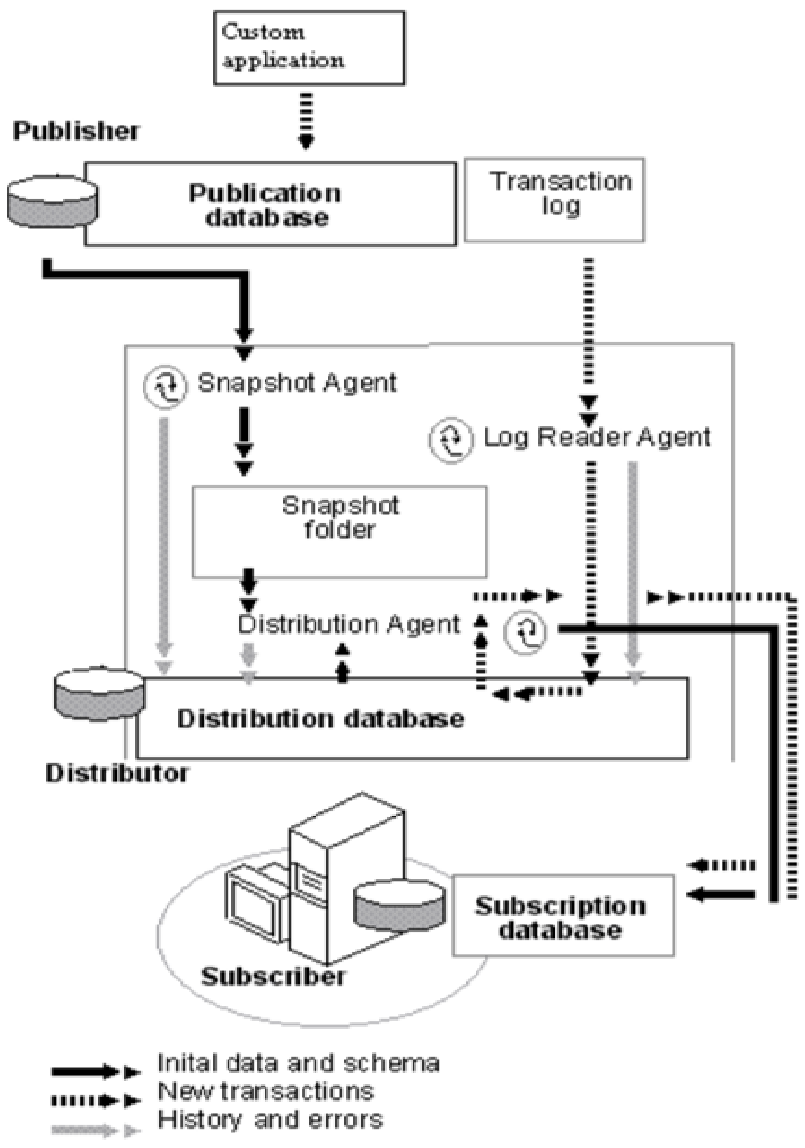


**Figure 2.2. How it works: Transactional Replication**

Transactional replication with updating subscribers: An offshoot of standard transactional replication, this method of replication basically works the same way, but adds to subscribers the ability to update data. When a subscriber makes a change to data locally, SQL Server uses the Microsoft Distributed Transaction Coordinator (MSDTC), a component included with SQL Server, to execute the same transaction on the publisher. This process allows for replication scenarios in which the published data is considered read-only most of the time but can be changed at the subscriber on occasion if needed.

Transactional replication with updating subscribers requires a permanent and reliable connection of medium to high bandwidth. [5] Transactional replication is helpful when:

- Incremental changes to be propagated to Subscribers as it occurs.
- It is needed the transactions to adhere to ACID properties.
- Subscribers are reliably and/or frequently connected to the Publisher. [6]

## 2.6. Distributed Transaction Management

The goal of transaction is to ensure that all of the objects managed by a server remain in a consistent state in the presence of server crashes. The server must guarantee that both transactions are carried out and the results recorded in permanent storage, or in the case of crashes, the effects are completely erased. The object that can be recovered after the server crashes is called recoverable object.

A client transaction becomes distributed if it invokes operations in several different servers. There are two different types of distributed transactions:

- Flat transaction
- Nested transaction [2]

In a flat transaction, a client makes requests to more than one server. For example, in figure 1, transaction T is a flat transaction that invokes operations on objects in servers X, Y and Z. A flat client transaction completes each of its requests before going on to the next one. Therefore, each transaction accesses servers' objects sequentially. [2]

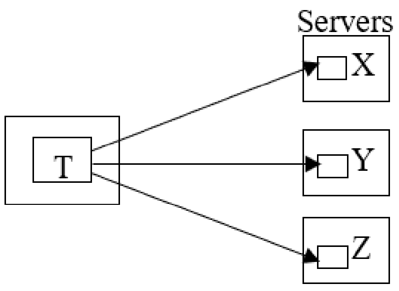


Figure 2.3. Flat Distributed Transaction

In a nested transaction, the top-level transaction can open sub-transactions and each sub-transaction can open further sub-transactions down to any depth of nesting. [2]

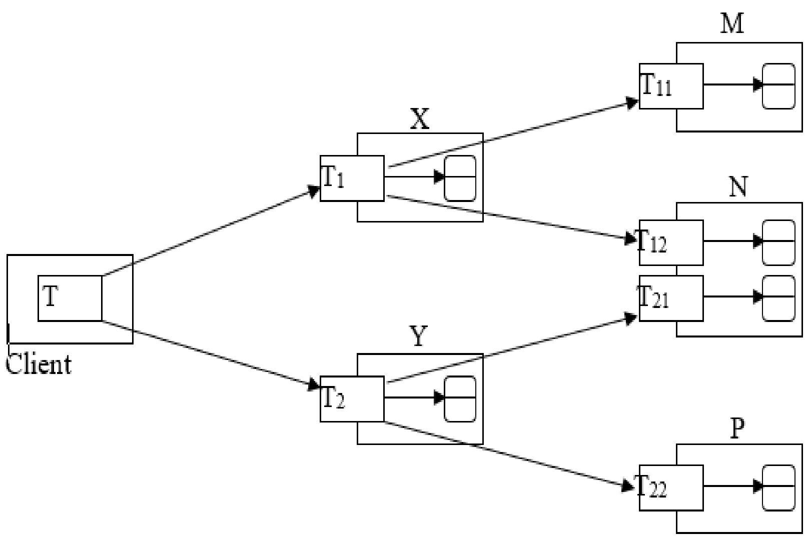


**Figure 2.4. Nested Distributed Transaction**

Figure 2 shows a client’s transaction T that opens two sub-transactions T1 and T2, which access objects at server X and Y. The sub-transactions T1 and T2 open further sub-transactions T11, T12, T21 and T22, which access objects at servers M, N and P. In the nested case, sub-transactions at the same level can run concurrently, so T1 and T2 are concurrent, and, they can run in parallel. The four sub-transactions T11, T12, T21 and T22 also run concurrently. [2]

# CHAPTER (3)
# CONCURRENCY CONTROL AND REPLICATION ARCHITECTURE

The concurrency control part of the replicated system that controls the execution order of these operations. Concurrency control manages simultaneous access to a database in a database management system (DBMS). Two users from editing the same record at the same time is prevented and also serializes transactions for backup and recovery. Publish-subscribe architecture builds replication around a centralized. Every directory server communicates with a central service and it uses the central service to publish its own changes and to receive notification about changes on other directory servers.

## 3.1. Database replication Middleware

Database replication is typically used to improve either read performance or write performance, while improving both read and write performance simultaneously is a more challenging task.

Figure 3.1 depicts master-slave replication, a popular technique used to improve read performance. In this scenario, read-only content is accessed on the slave nodes and updates are sent to the master. If the application can tolerate loose consistency, any data can be read at any time from the slaves given a freshness guarantee. As long as the master node can handle all updates, the system can scale linearly by merely adding more slave nodes. Examples of commercial products providing asynchronous master-slave replication are Microsoft SQL Server replication, Oracle Streams, Sybase Replication Server, MySQL replication, IBM DB2 DataPropagator, GoldenGate TDM platform, and Veritas Volume Replicator.

A special instance of read throughput improvement relates to legacy databases: often an old DB system is faced with increased read performance requirements, that can no longer satisfy, yet replacing the DB is too costly. Recently emerged strategies, such as satellite databases [29], offer a migration path for such cases. In the case of an e-commerce application, the main legacy database is preserved for all critical operations, such as orders, but less critical interactions, such as catalog browsing, can be offloaded to replicas. Such configurations typically use partial replication—all orders could be solely on the main legacy database, while only the catalog content is replicated. As an application might also be using multiple database instances inside the same RDBMS, the user can choose to replicate only specific database instances.

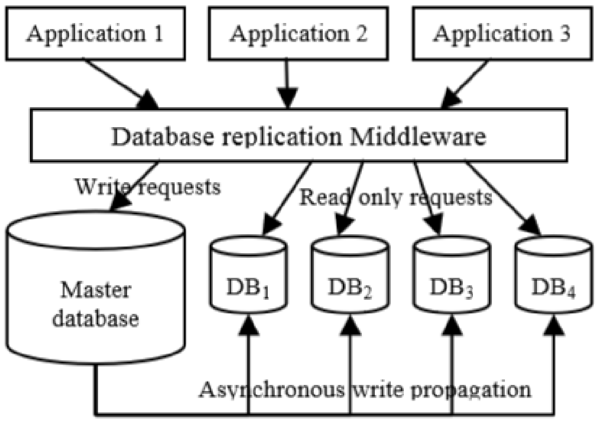


**Figure 3.1. Database Scale-out Scenario**

## 3.2. Multi-master Replication

Multi-master replication allows each replica owning a full copy of the database to serve both read and write requests. The replicated system then behaves as a centralized database, which theoretically does not require any application modifications. Replicas, however, need to synchronize in order to agree on a serializable execution order of transactions, so that each replica executes the update transactions in the same order. Also, concurrent transactions might conflict, leading to aborts and limiting the system's scalability [18]. Even though real applications generally avoid conflicting transactions, there are still significant research efforts trying to solve this problem in the replication middleware layer. The volume of update transactions, however, remains the limiting performance factor for such systems. As every replica has to perform all updates, there is a point beyond which adding more replicas does not increase throughput, because every replica is saturated applying updates.

### 3.3. Data Partitioning

Data partitioning techniques can be used to address write scalability. Figure 3.2 shows an example where data is logically split into 3 different partitions, each one being replicated. Common partitioning criteria are based on a table primary key and include techniques such as range partitioning, list partitioning and hash partitioning. The benefits of this approach are similar to RAID-0 for disks: updates can be done in parallel to partitioned data segments. Read latency can also be improved by exploiting intra-query parallelism and executing the sub-queries in parallel on each partition.

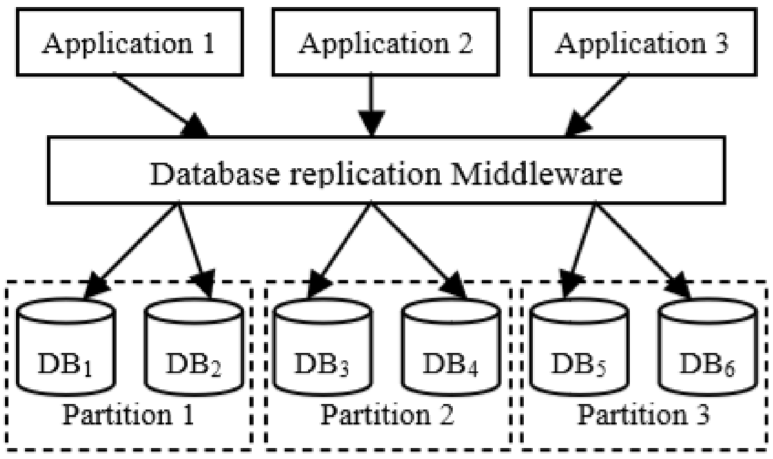


**Figure 3.2. Database Partitioning for Increased Write Performance**

### 3.4. Middleware-level Challenges

Middleware-based replication uses a middleware layer between the application and the database engines to implement replication, and there are multiple design choices for how to intercept client queries and implement replication across multiple nodes. We describe the most common alternatives with their pros and cons.

#### 3.4.1. Intercepting Queries

Query interception needs may force driver changes on the application side as database protocols evolve over time. A new driver on the application side might offer new functionality, such as support for transparent failover or load balancing. Moreover, protocol version implementation may vary from one platform and language to another. For example, each MySQL JDBC, ODBC and Perl driver has its own bugs and ways to interpret the protocols.

### 3.4.2. Statement vs. Transaction Replication

Multi-master replication can be implemented either by multicasting every update statement (i.e., statement replication) or by capturing transaction write-sets and propagating them after certification (i.e., transaction replication). Both approaches face significant challenges when put in production with real applications.

Non-deterministic queries are an important challenge: statement- based replication requires that the execution of an update statement produce the same result on each replica. However, SQL statements may legitimately produce different results on different replicas if they are not pre-processed before being issued.

Time-related macros such as ‘now’ or ‘current timestamp’ are likely to produce a different result, even if the replicas are synchronized in time. Simple query rewriting techniques can circumvent the problem by replacing the macro with a hard-coded value that is common to all replicas. Of course, all replicas must still be time-synchronized and set in the same time-zone, so that read queries provide consistent results.

**Write-set extraction** is usually implemented using triggers, to prevent database code modifications. This requires declaring additional triggers on every database table, as well as changing triggers every time the database schema is altered. This can be problematic both from an administrative as well as a performance standpoint when applications use temporary tables. If the application already uses triggers, write-set extraction through triggers might require an application rewrite. Materialized views also need special handling, to avoid duplicate write-set extraction by the triggers on the view and those on the underlying tables. Write-set extraction does not capture changes like auto- incremented keys, sequence values, or environment variable updates. Queries altering such database structures change the replica they execute on and can contribute to cluster divergence. Moreover, most of these data structures cannot be rolled back (for instance, an auto-incremented key or sequence number incremented in a transaction is not decremented at rollback time). Statement-based replication, at least, ensures that all these data structures are updated in the same order at all replicas. With transaction replication, if no coordination is done explicitly from the application, the cluster can end up in an endless effort to converge conflicting key values from different replicas.

**Locking and performance** are harder issues in statement-based replication. In particular, locking granularity is usually at the table level, as table information can be obtained through simple query parsing; however, this limits performance. Finer granularity (i.e., row level) would require re-implementing a large fraction of the database logic inside the middleware. Moreover, the middleware locking regime might not be compatible with the underlying database locking, leading to distributed deadlocks between the databases and the middleware.

## 3.5. Consistency Problems

- Consistency problems caused by concurrent processing include-
- Lost or buried Updates
- Inconsistent Analysis (Non-repeatable Read)
- Uncommitted Dependency (Dirty Read)
- Phantom Reads

### 3.5.1. Lost or buried Updates

This problem occurs when two or more transactions are read and update on the same data item at the share database. Each transaction is unaware of other transactions.

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.

### 3.5.2. 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 data item several times and reads different data each time because another transaction has been updated this item while the second transaction is reading. Inconsistent analysis involves multiple read (two or more) of the same item and each time the information is changed by another transaction; thus, this term is non-repeatable read.

### 3.5.3. Uncommitted Dependency (Dirty Read)

A transaction, if it retrieves or updates a data item that has been update by another transaction but not 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.

### 3.5.4 Phantom Reads

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 cause when insert or delete action is performed against a row that belongs to the range of rows being by a transaction.

### 3.5.5 Consistency Modes

A session can be considered as a sequence of "transactions" made in the same database connection. Each of this "transactions" can be made in one of the following consistency modes:

**Plain consistency** - This mode does not allow any write access on objects. It guarantees that all read accesses made in this mode follow a causal order.

On the other hand, this mode imposes no restriction on the current objects being read. Thus, they may be outdated.

**Checkout consistency** - This mode is similar to the traditional sequential consistency, although it does not guarantee isolation. Thus, if several sessions have read a given object, one of these sessions is allowed to promote its access mode to "writing". However, if two of these sessions have promoted their access modes from reading to writing, one of them will be aborted.

**Transaction consistency -** In this mode, the usual transaction guarantees: atomicity, sequential consistency, isolation and durability, are enforced.

A session always starts in plain mode. If the guarantees provided in this mode are not sufficient for the application, it can promote its consistency mode to checkout or transaction.

In these two modes, all accesses are temporarily stored until an explicit call to the commit () or rollback () operations is made (with the usual meaning of such operations). Once one of these operations have been made, the session returns automatically to plain mode. Thus, the programmer is able to choose the consistency mode of each session that composes her or his application, and this consistency mode can be variegated as needed while a session is running.

## 3.7. The Proposed System: COPLA Architecture

In the development of the COPLA (Common Object Programmer Library Access) architecture that is very interested in a serializable transactional behavior that guarantees an eager replication to all nodes, as well as in the development of recovery techniques so no back-up copies are necessary and alive nodes may continue working independently of a node failure.

COPLA consists of a middleware architecture providing transparency for persistent object state replication while guarantees several consistency levels: *transactional* (serializable), *checkout (similar to the concurrent version* system guarantees) and *plain (read-only).* The COPLA architecture consists of the three layers as depicted in figure 3.3. From bottom to top, they are the following:

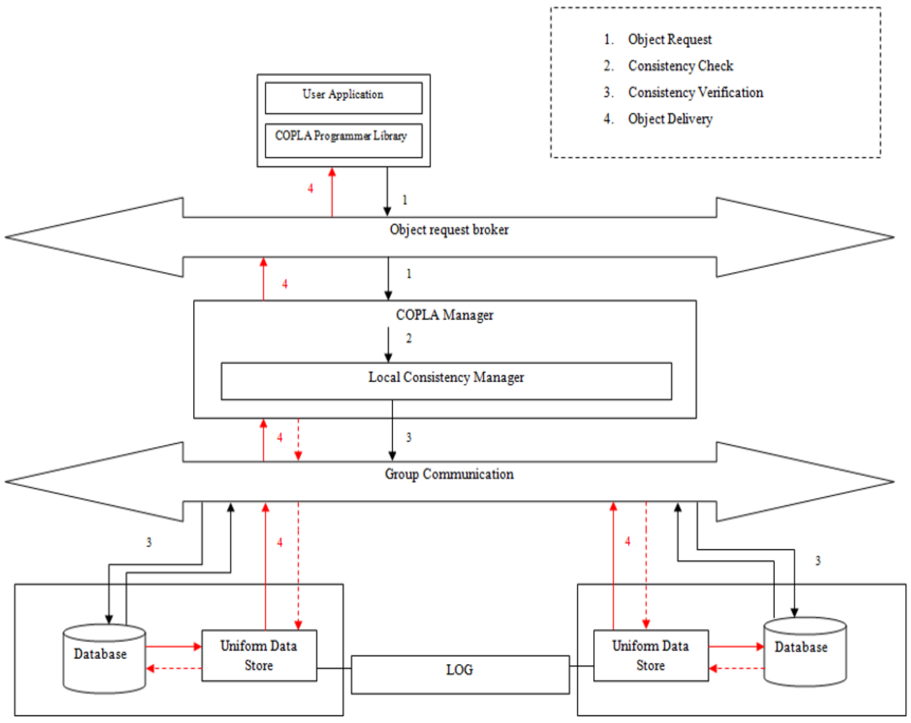


**Figure 3.3: The Proposed System COPLA Architecture**

### 3.8. Uniform Data Store (UDS)

This component manages the persistent data of a Global Data system. It interacts directly with a relational DBMS, storing there the persistent objects of the given application and the metadata of the consistency protocol. It isolates the upper layers from the actual storage system used. In practice, support for different RDBMSs will be provided in the final release of the UDS.

### 3.9. COPLA Manager

The COPLA manager is the core component of the COPLA architecture. It manages database sessions (which may include multiple sequential transactions, working in different consistency modes) and controls the set of database replicas that compose the Global Data system. This manager also provides some caches to improve the efficiency of the database accesses.

### 3.10. A local consistency manager

A local consistency manager - is included in this layer. Multiple consistency protocol objects may be used in this component, but only one is allowed at a time. All consistency protocols share some characteristics. Hence, the consistency checks must be done at commit time. If a session is allowed to commit, its updates are multicast by its local consistency component to all consistency components placed in other Global Data nodes.

The way depends on the consistency protocol is being used. All of the communication among Global Data databases is managed by this component.

## 3.11. COPLA Programmer Library

COPLA programmer library is the layer used in Global Data applications to access system services. It also provides some cache support and multithreading optimizations that improve the overall system performance.

## 3.12. Actions Performed by the Algorithm

The algorithm includes concurrency control and recovery tasks in behalf of cleanliness, it may be better explained, grouping both tasks in different parts: first, the concurrency control accomplished by our algorithm proposal, and, next, the recovery process.1)

**Concurrency Control**: Each time a session requests a lock, no matter what kind of lock is requested, a lock request action is executed. This action checks whether the lock assignment is compatible with the current sessions holding the lock. In such a case the lock is granted and the session may continue requesting new locks (running state), otherwise the session must be blocked or aborted. This is determined by the deadlock prevention function.

Whenever a session becomes blocked, it cannot request any lock until the session is woken up again. This event may occur whenever a release lock operation is performed in the object where the session is waiting. At this point, it will be briefly outline the release lock policy followed in our algorithm.

The only case where there are several sessions assigned on an object is when they request a read-lock on it. Thus, no waiting session will access that object until all read locks are released. Special cases arise when the session assigned to an object is a write-lock or copy-lock. Quite often there will be sessions waiting to acquire a read-lock or a copy-lock on that object. It will not happen that a write-lock is waiting to be assigned, since it does not allow blind-writes. Due to our deadlock prevention policy, the copy-lock, if it exists, will be located at the last position of the queue.

A session may be aborted due to two reasons: the final user decides to abort the current session; or, the deadlock prevention function determines that the given session must be aborted in order to prevent a deadlock. In the first situation, the session is always local and the tasks to be done are: aborting changes in the RDBMS, releasing all locks held by the session and switching the session state to abort.

Aborts induced by the deadlock prevention technique are treated quite different, since they usually involve message exchange with other COPLA sites.

**Recovery Actions**

The notion of consistency that the use for defining the targets of recovery is tied to the transaction paradigm, which we have encapsulated in the "ACID principle." According to this definition, a database is consistent if and only if it contains the results of successful transactions. Such a state will hereafter be called transaction consistent or logically consistent.

A transaction, in turn, must not see anything but effects of complete transactions (i.e., a consistent database in those parts that it uses), and will then, by definition, create a consistent update of the database. In the moment of ignore transactions being aborted during normal execution and consider only a system failure (a crash). It might be encountered the situation depicted in Figure 3.1. Transactions T1, T2, and T3 have committed before the crash, and therefore will survive.

Recovery after a system failure must ensure that the effects of all successful transactions are actually reflected in the database. But what is to be done with T4 and T5? Transactions have been defined to be atomic; they either succeed or disappear as though they had never been entered. There is therefore no choice about what to do after a system failure; the effects of all incomplete transactions must be removed from the database. Clearly, a recovery component adhering to these principles will produce a transaction consistent database. Since all successful transactions have contributed to the database state, it will be the most recent transaction consistent state.

# CHAPTER 4

# DESIGN AND IMPLEMENTATION OF THE SYSTEM

The proposed system replicates the data change of original database in the key/value stores and prevents all read transactions from hitting the original database. All update transactions are sent to the corresponding master. The master database has a set of slaves that are its replicas, serve the read-only transactions in the system. Updates are disseminated (propagate) from master to its slave nodes by eagerly upon their arrival several updates and applying them together as shown in figure 4.1.

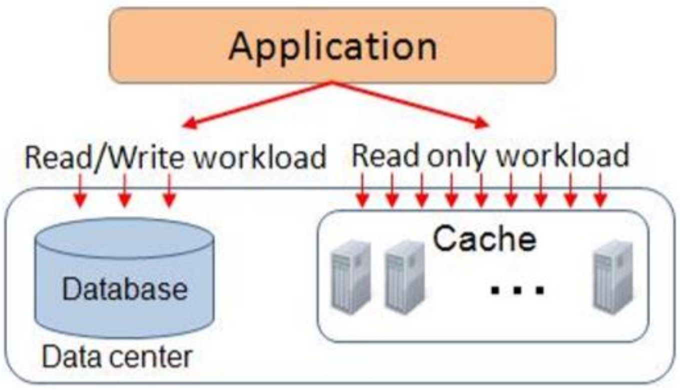
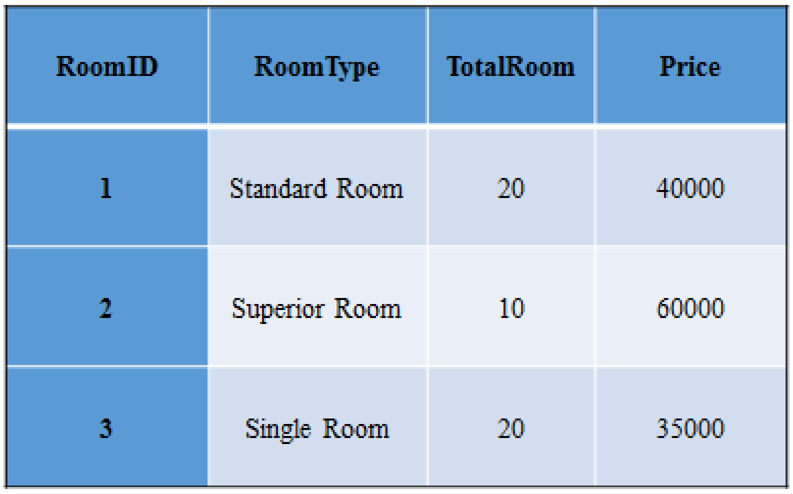


**Figure 4.1: The System Overview**

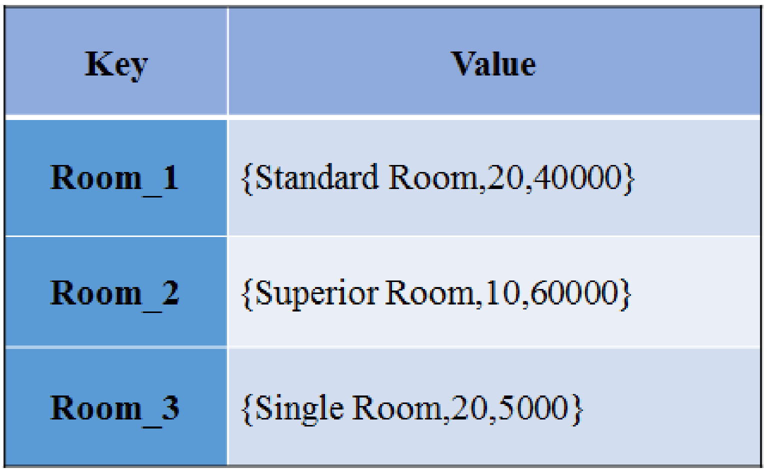
## 4.1 The Proposed System Architecture

Relational Data Over Key-value Store: In the relational data in RDBMS into a key-value store, the data layout on these two stores are different, need to provide a mapping scheme to map the relational data layout into the key-value data layout.



| RoomID | RoomType | TotalRoom | Price |
|---|---|---|---|
| 1 | Standard Room | 20 | 40000 |
| 2 | Superior Room | 10 | 60000 |
| 3 | Single Room | 20 | 35000 |

**Figure 4.2: Tuples in Room Table**



| Key | Value |
|---|---|
| Room_1 | {Standard Room,20,40000} |
| Room_2 | {Superior Room,10,60000} |
| Room_3 | {Single Room,20,5000} |

**Figure 4.3: Key/value objects for the tuples in ROOM table**

A standard relational database is the database that stores all the persistent application data and responsible for handling read/write transactional workload. The database system for certain application workloads, the database is replicated into a distributed key-value store. The replicated key-value store plays similar role to cache for the database, is used to handle the read only workload while the read/write work-load is run directly on the original database as shown in figure 4.3 and figure 4.4. Between the relational database and the key-value store, main component of system (Replication Middleware) that are responsible for synchronizing the key-value store with the relational database.

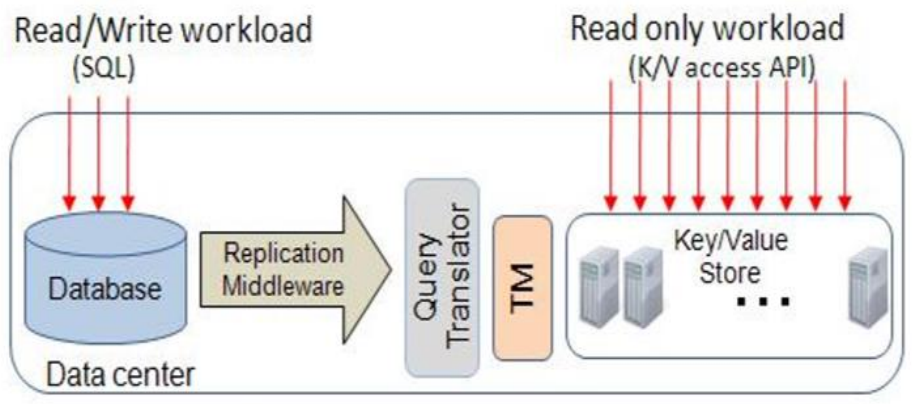


**Figure 4.4: Architecture of Transactional Replication Using Key/Value Store**

**Query Translator (QT):** component is responsible for translating the update only SQL statements into key/value store that can be directly executed on the key-value store. The replication workload only contains the write operations to key-value store.

**The Transaction Manager (TM)**: component is used to apply the transactions to the key-value store concurrently. The TM component essentially implements the proposed concurrent replication method. When the transactions reach the TM (transaction Manager) they are in the form of key-value store as they have been translated by QT component.

**Replication Middleware:** The Replication Middleware component is responsible for shipping the transactions (Write/Update) from the relational database to the replica in the key-value store. It periodically reads the transaction log in the database; the new updated transactions are ships to the key-value store. The transactions only include write statements and there is no need to apply read statements from the relational database in the replica.

The data in the key-value store is the replication of the data in the original database. To maintain the replicated data in the key-value store synchronized with the original data in the relational database, the system uses the replication middleware. When a transaction processing is committed, the system’s middleware must send committed data update to all replicas. Then, the middleware checks whether all of the replicas have committed data updated or not. If all of the replicas receive the data update, the middleware send the committed transaction to the completed transaction list. If not, the middleware will restart the replication to replicate the data update to all replicas.

## 4.2 Concurrency Control for Replication

Novel concurrency control mechanism for replicated system is different from the ordinary concurrency control systems (not replicated system) because of the execution- defined order of transactions. In an ordinary concurrency control algorithm, when a set of transactions are executed, the result of the execution is directly acceptable because of such system no need to propagate the execution result and the system can immediately accept for the new request.

To control concurrency in replicated system with ordinary control, the system must adopt the other replication mechanisms such as active or passive. However, in the propose concurrency control algorithm has to guarantee the consistency between original and key/value store. The result of committed transactions must be exactly the same result of the system replicas' (key/value store) data. So, the system halt to execute the new request until the earlier execution result is completely updated on key/value stores.

The priority queue, which is referred to as the "CommitReqQ' in the algorithm, is responsible for keeping the order of transactions based on ascending order of their sequence numbers.

Committed transaction list: A committed transaction is the one that does not have any conflict with its predecessors. However, the updates in its buffer has not been applied to the key-value store. Such committed transactions are stored in a list which is called "Committed transaction list".

Completed transaction list: A completed transaction is a committed transaction that the updates in its buffer have been applied to the key-value store. Such complete transaction is stored in a list which is called "Completed transaction list". But the completed transaction list has a limited amount of data store. So, the completed transaction list must be periodically cleaned to store the new completed transaction for future.

## 4.3. The Novel Concurrency Control Algorithm of the Proposed System

CRQ = Commit Request Queue

CTL = Committed Transaction List (But not finish multicasting to all replicas)

CPTL = Completed Transaction List (Finished the task of multicasting to All replicas)

```
T i  = user requested transaction variable
T j belong to the transactions in CTL
BEGIN
T i  = T1 (First transaction for the CRQ);
Remove T1 from CRQ;
If (Ti is conflict with Tj) //Tj is committed but have not been multicast to al replicas
        {
Ti is added to the restart list;
                    }
Else
{
    Ti is added to the CTL;
    After the transaction Ti is completed and its effect is applied to the Key-Value store. (Replicas)
}
Check the restart list;
        If (Restart is not empty)
{
Restart the transaction to be committed transaction;
}
Add completed transaction to CPTL;
END
```

**The algorithm also uses two lists:**

*The committed transaction list* holds the transactions that are in COMMITTED state and have committed successfully.

*The completed transaction list* contains the transactions that are in COMPLETED state which are the committed transactions that have also been applied to the key-value store. The concurrency control algorithm is started by checking the first transaction in the CommitReqPQ. If this transaction's sequence number is not the expected sequence number the algorithm does nothing and waits until the transaction with the expected sequence number is put into the CommitReqPQ.

If the transaction in the head of queue has the expected sequence number it is removed from the queue and is examined for conflict. When expected transaction is on top of the CommitReqPQ it means that all the preceding transactions have been evaluated by the algorithm and are in COMMITTED or COMPLETED state. The system flow is shown in figure6.

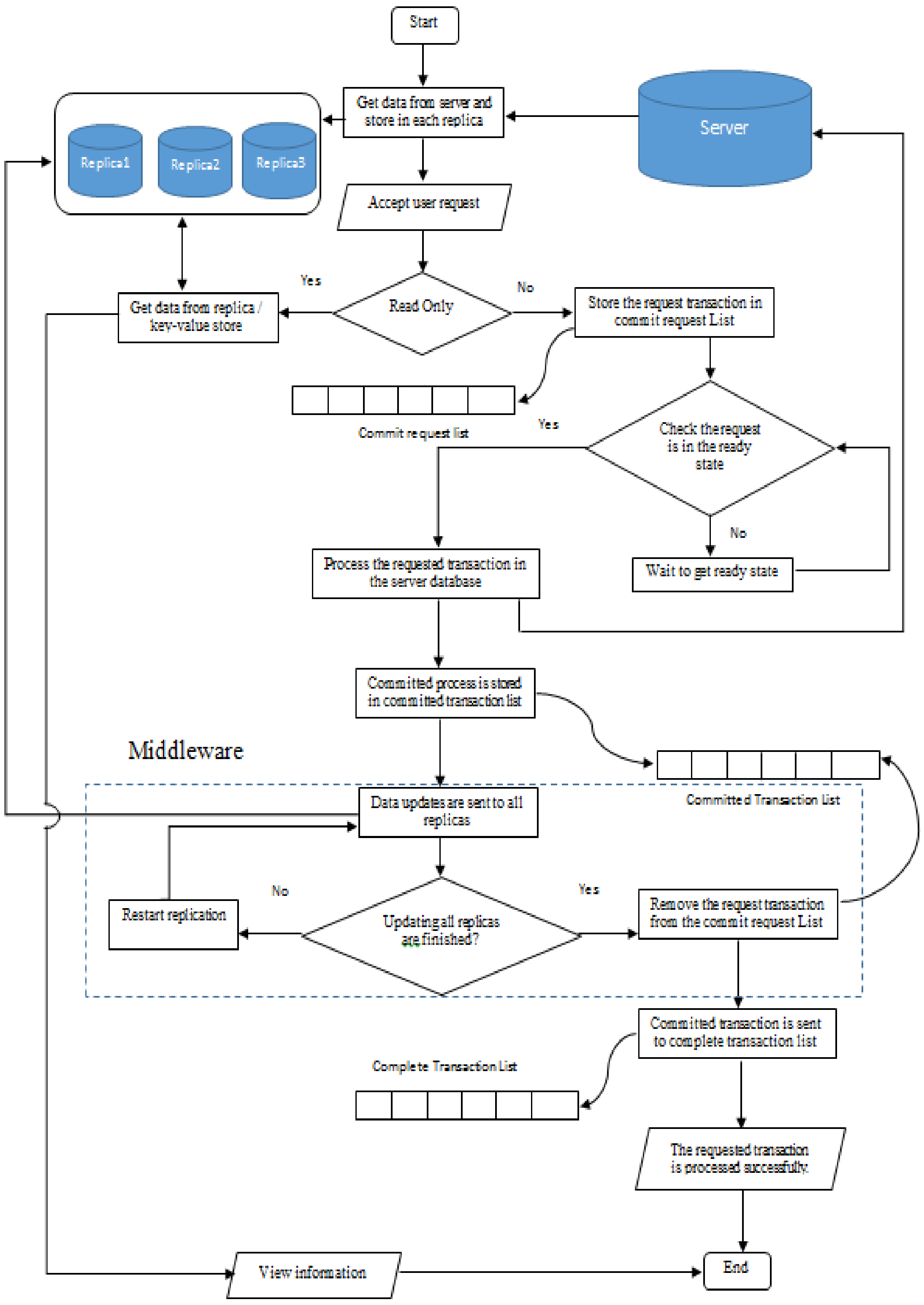

Start
Get data from server and store in each replica
Server
Replica1
Replica2
Replica3
Accept user request
Read Only
Yes
No
Get data from replica / key-value store
Store the request transaction in commit request List
Commit request list
Yes
Check the request is in the ready state
No
Process the requested transaction in the server database
Wait to get ready state
Committed process is stored in committed transaction list
Middleware
Committed Transaction list
Data updates are sent to all replicas
No
Yes
Restart replication
Updating all replicas are finished?
Remove the request transaction from the commit request List
Committed transaction is sent to complete transaction list
Complete Transaction List
The requested transaction is processed successfully.
View information
End

**Figure 4.5: The System Flow**

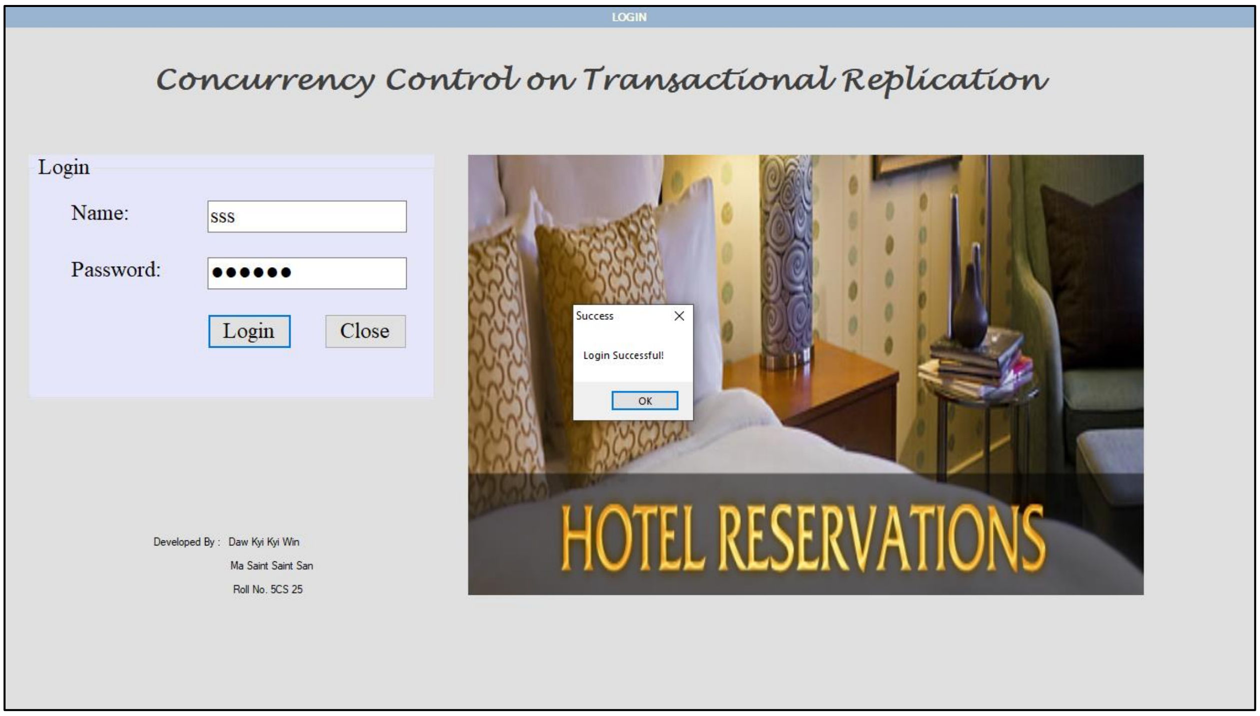
## 4.4. Discarding Completed Transactions

In propose concurrency control algorithm the Completed Transaction List is the last list that stores transactions. However, by processing more and more transactions this list will grow larger. Therefore, the system need to limit the size if this list and remove the completed transactions from the list if there is no need for them.

A completed transaction Ti is stored in the Completed Transaction List. If another transaction Tj starts before the completion of Ti, and Tj has conflict with Ti, then there is a possibility that Tj did not use the updated data resulted from Ti. Thus, in order to make sure that Tj observes the results of Ti, it does not need to make sure that Tj starts after completion of Ti. Based on this assumption, if there is no active transaction that has started before completion of a transaction Ti and then completed Ti can safely be removed from the Completed Transaction List.

## 4.5. System Designs

### Login Form



Name: sss

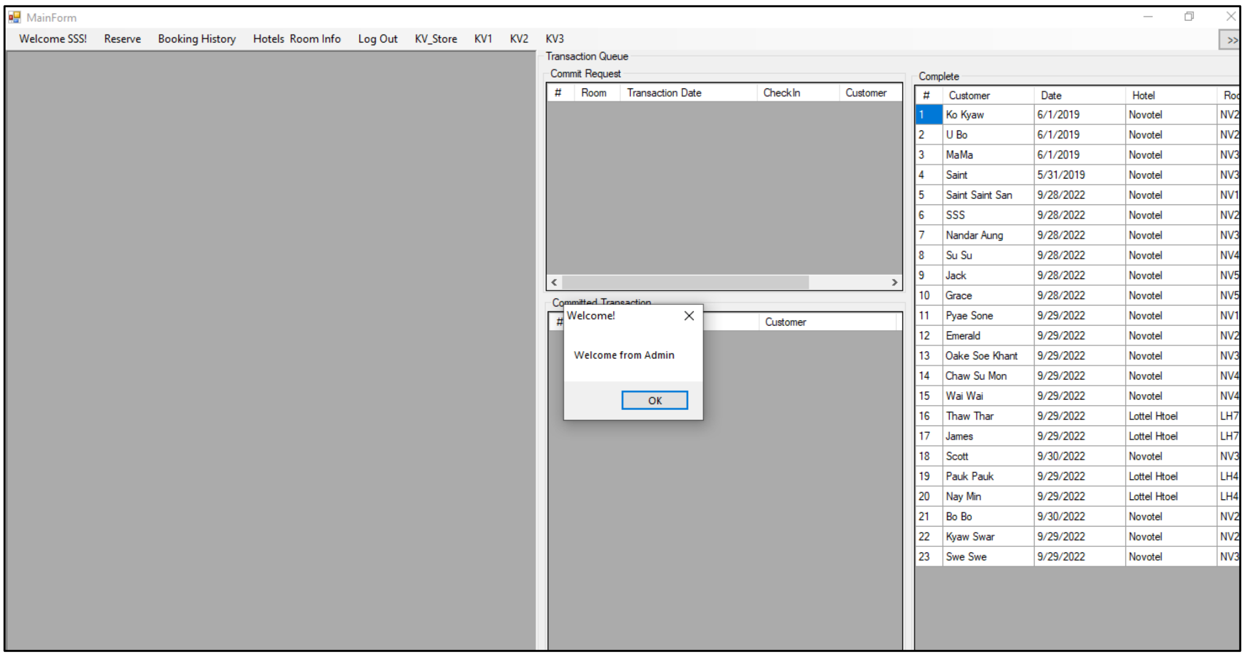
Password: ●●●●●●

Login Successful!

Developed By : Daw Kyi Kyi Win
Ma Saint Saint San
Roll No. 5CS 25

Figure 4.6: Login Form

### Successful Login Information



| # | Room | Transaction Date | CheckIn | Customer |
|---|---|---|---|---|

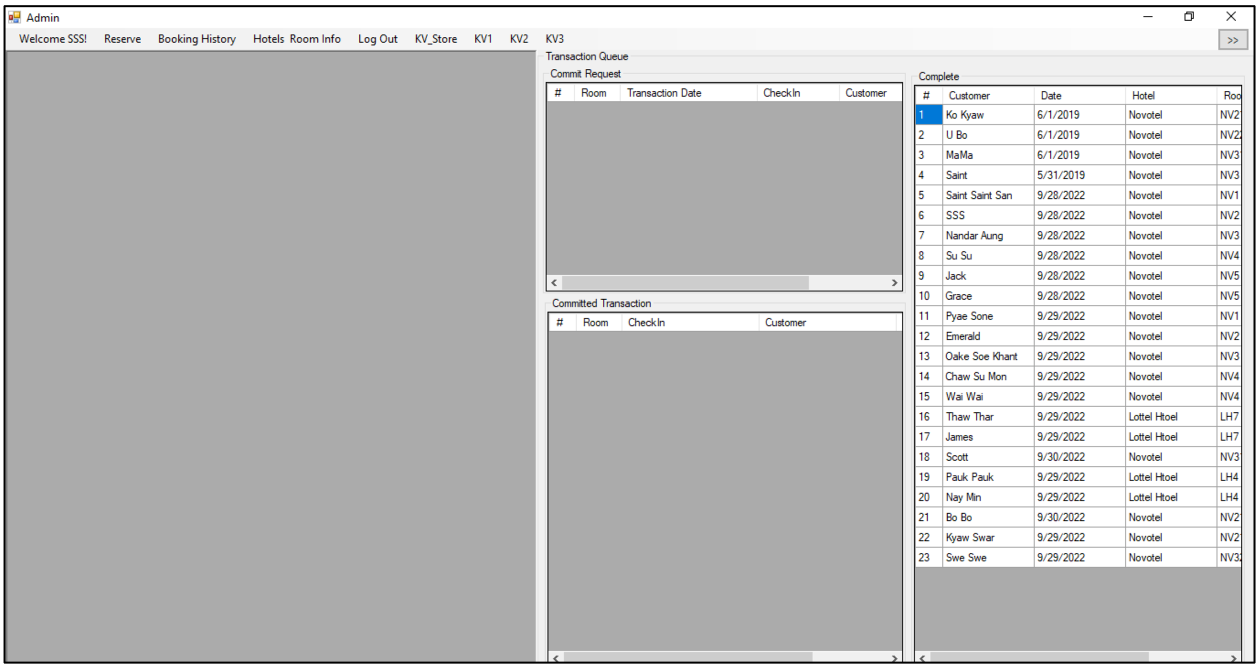
Welcome from Admin

Complete

| # | Customer | Date | Hotel | Ro |
|---|---|---|---|---|
| 1 | Ko Kyaw | 6/1/2019 | Novotel | NV2 |
| 2 | U Bo | 6/1/2019 | Novotel | NV2 |
| 3 | MaMa | 6/1/2019 | Novotel | NV3 |
| 4 | Saint | 5/31/2019 | Novotel | NV3 |
| 5 | Saint Saint San | 9/28/2022 | Novotel | NV1 |
| 6 | SSS | 9/28/2022 | Novotel | NV2 |
| 7 | Nandar Aung | 9/28/2022 | Novotel | NV3 |
| 8 | Su Su | 9/28/2022 | Novotel | NV4 |
| 9 | Jack | 9/28/2022 | Novotel | NV5 |
| 10 | Grace | 9/28/2022 | Novotel | NV5 |
| 11 | Pyae Sone | 9/29/2022 | Novotel | NV1 |
| 12 | Emerald | 9/29/2022 | Novotel | NV2 |
| 13 | Oake Soe Khant | 9/29/2022 | Novotel | NV3 |
| 14 | Chaw Su Mon | 9/29/2022 | Novotel | NV4 |
| 15 | Wai Wai | 9/29/2022 | Novotel | NV4 |
| 16 | Thaw Thar | 9/29/2022 | Lottel Htoel | LH7 |
| 17 | James | 9/29/2022 | Lottel Htoel | LH7 |
| 18 | Scott | 9/30/2022 | Novotel | NV3 |
| 19 | Pauk Pauk | 9/29/2022 | Lottel Htoel | LH4 |
| 20 | Nay Min | 9/29/2022 | Lottel Htoel | LH4 |
| 21 | Bo Bo | 9/30/2022 | Novotel | NV2 |
| 22 | Kyaw Swar | 9/29/2022 | Novotel | NV2 |
| 23 | Swe Swe | 9/29/2022 | Novotel | NV3 |

Figure 4.7: Successful Login Information

## System Main Form



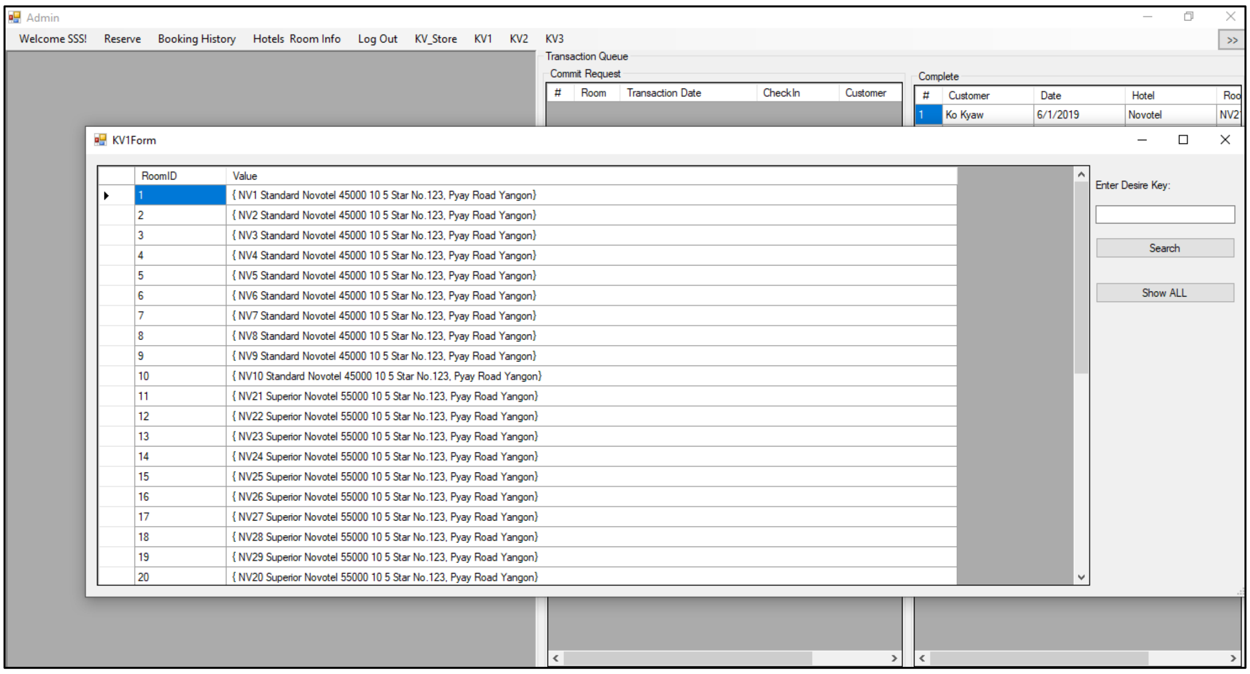
| # | Room | Transaction Date | CheckIn | Customer |
|---|---|---|---|---|

| # | Room | CheckIn | Customer |
|---|---|---|---|

| # | Customer | Date | Hotel | Roo |
|---|---|---|---|---|
| 1 | Ko Kyaw | 6/1/2019 | Novotel | NV2 |
| 2 | U Bo | 6/1/2019 | Novotel | NV2 |
| 3 | MaMa | 6/1/2019 | Novotel | NV3 |
| 4 | Saint | 5/31/2019 | Novotel | NV3 |
| 5 | Saint Saint San | 9/28/2022 | Novotel | NV1 |
| 6 | SSS | 9/28/2022 | Novotel | NV2 |
| 7 | Nandar Aung | 9/28/2022 | Novotel | NV3 |
| 8 | Su Su | 9/28/2022 | Novotel | NV4 |
| 9 | Jack | 9/28/2022 | Novotel | NV5 |
| 10 | Grace | 9/28/2022 | Novotel | NV5 |
| 11 | Pyae Sone | 9/29/2022 | Novotel | NV1 |
| 12 | Emerald | 9/29/2022 | Novotel | NV2 |
| 13 | Oake Soe Khant | 9/29/2022 | Novotel | NV3 |
| 14 | Chaw Su Mon | 9/29/2022 | Novotel | NV4 |
| 15 | Wai Wai | 9/29/2022 | Novotel | NV4 |
| 16 | Thaw Thar | 9/29/2022 | Lottel Htoel | LH7 |
| 17 | James | 9/29/2022 | Lottel Htoel | LH7 |
| 18 | Scott | 9/30/2022 | Novotel | NV3 |
| 19 | Pauk Pauk | 9/29/2022 | Lottel Htoel | LH4 |
| 20 | Nay Min | 9/29/2022 | Lottel Htoel | LH4 |
| 21 | Bo Bo | 9/30/2022 | Novotel | NV2 |
| 22 | Kyaw Swar | 9/29/2022 | Novotel | NV2 |
| 23 | Swe Swe | 9/29/2022 | Novotel | NV3 |

Figure 4.8: System Main Form

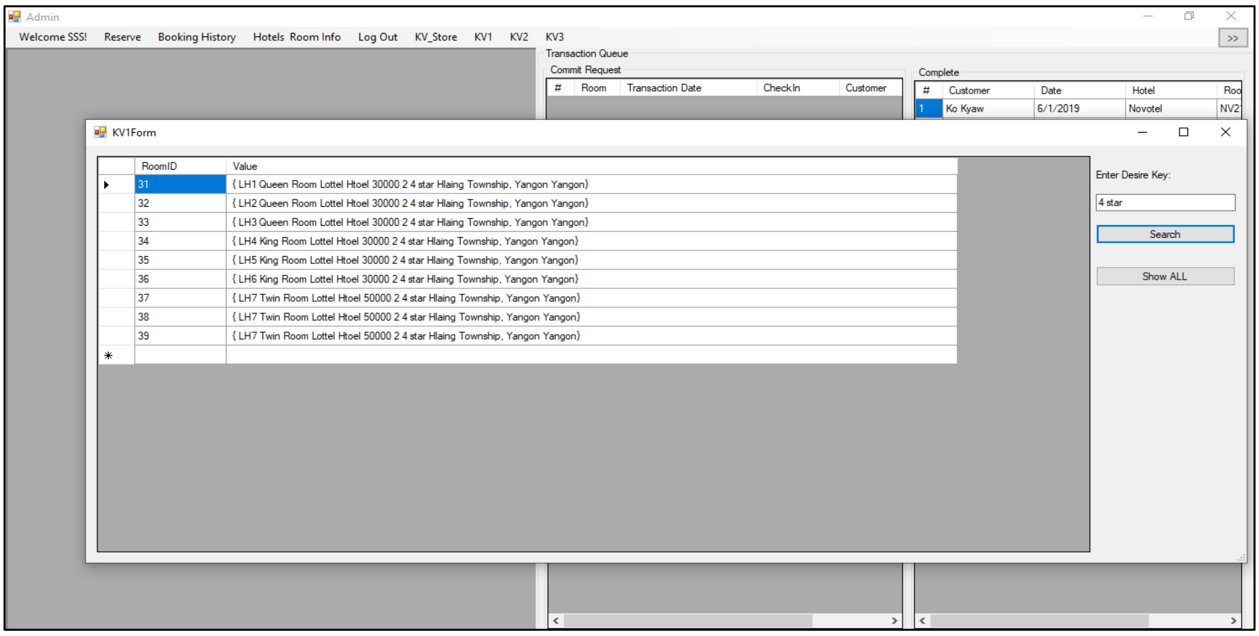
## Key-Value Data Information



| RoomID | Value |
|---|---|
| 1 | { NV1 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 2 | { NV2 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 3 | { NV3 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 4 | { NV4 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 5 | { NV5 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 6 | { NV6 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 7 | { NV7 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 8 | { NV8 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 9 | { NV9 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 10 | { NV10 Standard Novotel 45000 10 5 Star No.123, Pyay Road Yangon} |
| 11 | { NV21 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |
| 12 | { NV22 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |
| 13 | { NV23 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |
| 14 | { NV24 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |
| 15 | { NV25 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |
| 16 | { NV26 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |
| 17 | { NV27 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |
| 18 | { NV28 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |
| 19 | { NV29 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |
| 20 | { NV20 Superior Novotel 55000 10 5 Star No.123, Pyay Road Yangon} |

Figure 4.9: Key-Value Data Information

## Searching Key with 4 Stars Hotel Information



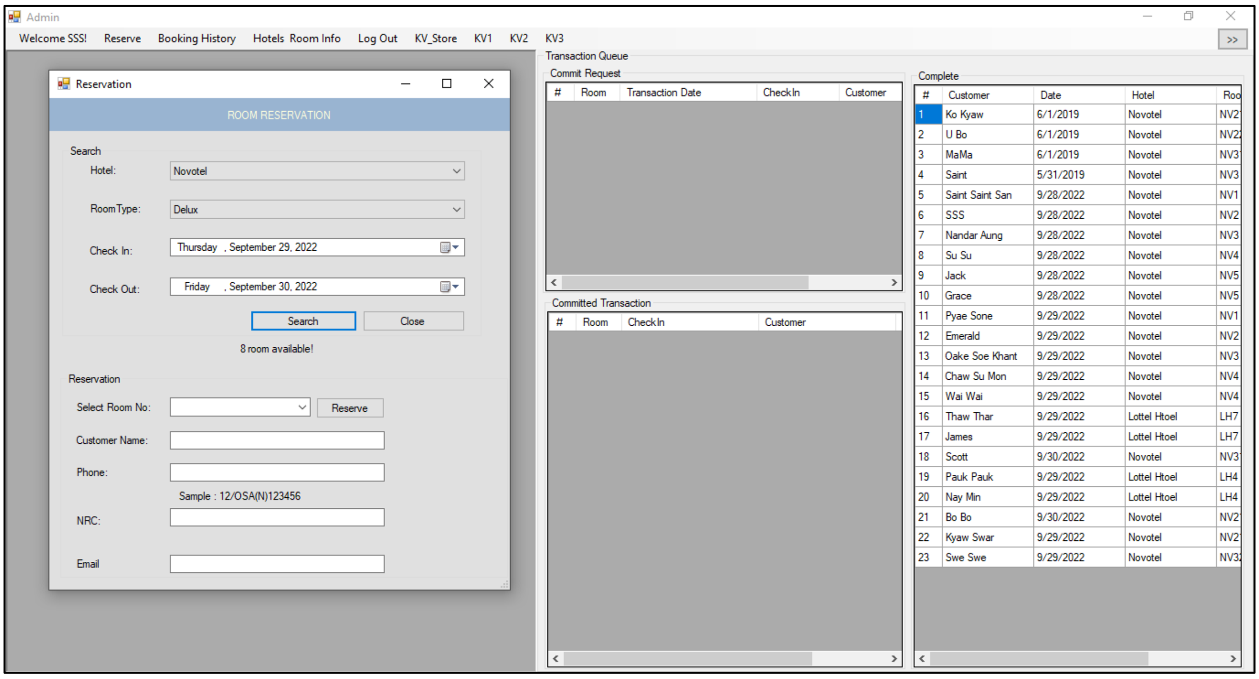
KV1Form

Enter Desire Key: 4 star

| | RoomID | Value |
|---|---|---|
| ▶ | 31 | { LH1 Queen Room Lottel Htoel 30000 2 4 star Hlaing Township, Yangon Yangon} |
| | 32 | { LH2 Queen Room Lottel Htoel 30000 2 4 star Hlaing Township, Yangon Yangon} |
| | 33 | { LH3 Queen Room Lottel Htoel 30000 2 4 star Hlaing Township, Yangon Yangon} |
| | 34 | { LH4 King Room Lottel Htoel 30000 2 4 star Hlaing Township, Yangon Yangon} |
| | 35 | { LH5 King Room Lottel Htoel 30000 2 4 star Hlaing Township, Yangon Yangon} |
| | 36 | { LH6 King Room Lottel Htoel 30000 2 4 star Hlaing Township, Yangon Yangon} |
| | 37 | { LH7 Twin Room Lottel Htoel 50000 2 4 star Hlaing Township, Yangon Yangon} |
| | 38 | { LH7 Twin Room Lottel Htoel 50000 2 4 star Hlaing Township, Yangon Yangon} |
| | 39 | { LH7 Twin Room Lottel Htoel 50000 2 4 star Hlaing Township, Yangon Yangon} |
| * | | |

**Figure 4.10: Searching Key with 4 Stars Hotel Information**

## Reservation Form (Admin Perspective View)



Reservation

ROOM RESERVATION

Search

| | |
|---|---|
| Hotel: | Novotel |
| RoomType: | Delux |
| Check In: | Thursday , September 29, 2022 |
| Check Out: | Friday , September 30, 2022 |

8 room available!

Reservation

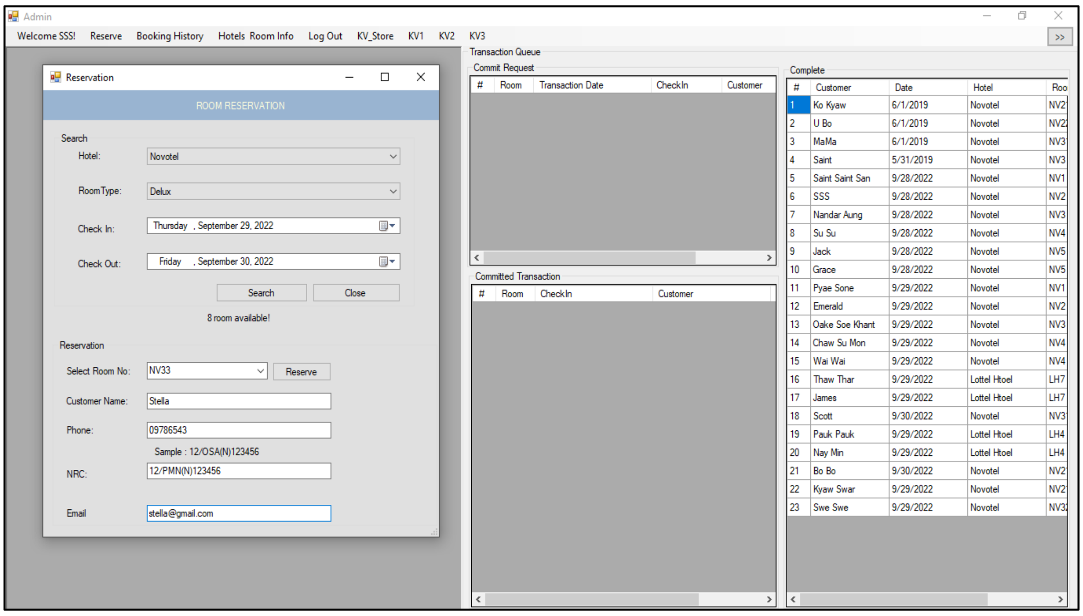
| | |
|---|---|
| Select Room No: | |
| Customer Name: | |
| Phone: | |
| | Sample : 12/OSA(N)123456 |
| NRC: | |
| Email | |

Complete

| # | Customer | Date | Hotel | Roo |
|---|---|---|---|---|
| 1 | Ko Kyaw | 6/1/2019 | Novotel | NV2 |
| 2 | U Bo | 6/1/2019 | Novotel | NV2 |
| 3 | MaMa | 6/1/2019 | Novotel | NV3 |
| 4 | Saint | 5/31/2019 | Novotel | NV3 |
| 5 | Saint Saint San | 9/28/2022 | Novotel | NV1 |
| 6 | SSS | 9/28/2022 | Novotel | NV2 |
| 7 | Nandar Aung | 9/28/2022 | Novotel | NV3 |
| 8 | Su Su | 9/28/2022 | Novotel | NV4 |
| 9 | Jack | 9/28/2022 | Novotel | NV5 |
| 10 | Grace | 9/28/2022 | Novotel | NV5 |
| 11 | Pyae Sone | 9/29/2022 | Novotel | NV1 |
| 12 | Emerald | 9/29/2022 | Novotel | NV2 |
| 13 | Oake Soe Khant | 9/29/2022 | Novotel | NV3 |
| 14 | Chaw Su Mon | 9/29/2022 | Novotel | NV4 |
| 15 | Wai Wai | 9/29/2022 | Novotel | NV4 |
| 16 | Thaw Thar | 9/29/2022 | Lottel Htoel | LH7 |
| 17 | James | 9/29/2022 | Lottel Htoel | LH7 |
| 18 | Scott | 9/30/2022 | Novotel | NV3 |
| 19 | Pauk Pauk | 9/29/2022 | Lottel Htoel | LH4 |
| 20 | Nay Min | 9/29/2022 | Lottel Htoel | LH4 |
| 21 | Bo Bo | 9/30/2022 | Novotel | NV2 |
| 22 | Kyaw Swar | 9/29/2022 | Novotel | NV2 |
| 23 | Swe Swe | 9/29/2022 | Novotel | NV3 |

**Figure 4.11: Reservation Form (Admin Perspective View)**

## Sample Booking Information



Admin

Welcome SSS! Reserve Booking History Hotels Room Info Log Out KV_Store KV1 KV2 KV3

Reservation

ROOM RESERVATION

Search

| | |
|---|---|
| Hotel: | Novotel |
| RoomType: | Delux |
| Check In: | Thursday , September 29, 2022 |
| Check Out: | Friday , September 30, 2022 |

Search Close

8 room available!

Reservation

| | |
|---|---|
| Select Room No: | NV33 |
| Customer Name: | Stella |
| Phone: | 09786543 |
| | Sample : 12/OSA(N)123456 |
| NRC: | 12/PMN(N)123456 |
| Email | stella@gmail.com |

Reserve

Transaction Queue

Commit Request

| # | Room | Transaction Date | CheckIn | Customer |
|---|---|---|---|---|

Committed Transaction

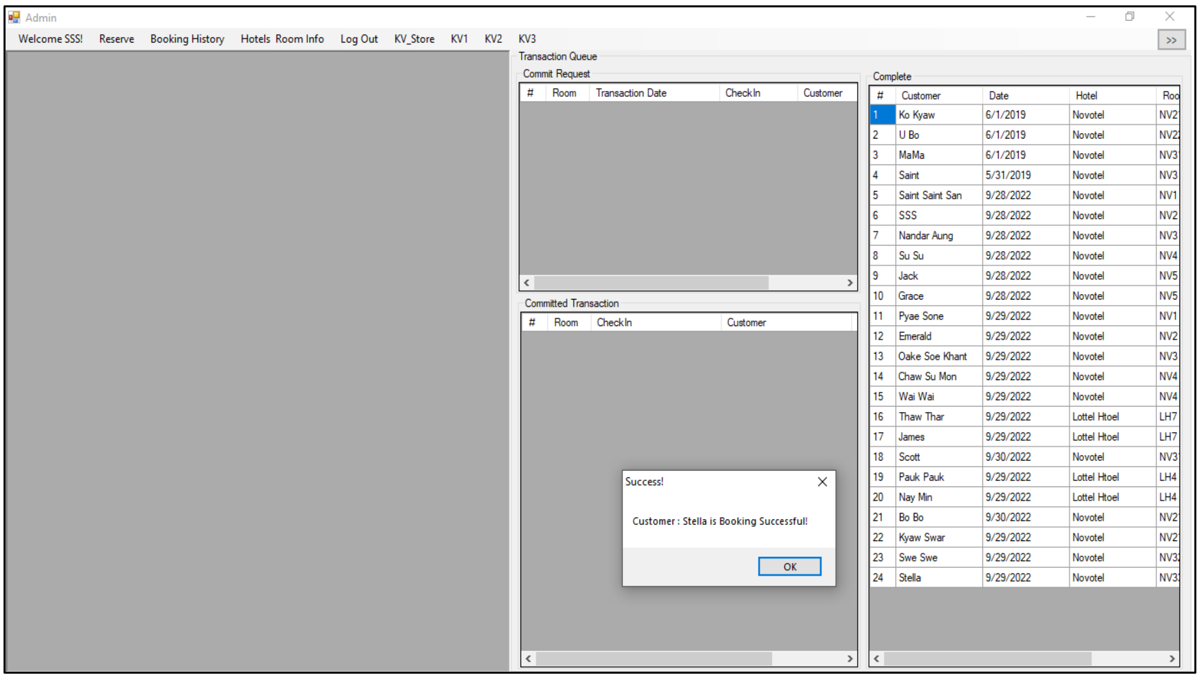
| # | Room | CheckIn | Customer |
|---|---|---|---|

Complete

| # | Customer | Date | Hotel | Roo |
|---|---|---|---|---|
| 1 | Ko Kyaw | 6/1/2019 | Novotel | NV2 |
| 2 | U Bo | 6/1/2019 | Novotel | NV2 |
| 3 | MaMa | 6/1/2019 | Novotel | NV3 |
| 4 | Saint | 5/31/2019 | Novotel | NV3 |
| 5 | Saint Saint San | 9/28/2022 | Novotel | NV1 |
| 6 | SSS | 9/28/2022 | Novotel | NV2 |
| 7 | Nandar Aung | 9/28/2022 | Novotel | NV3 |
| 8 | Su Su | 9/28/2022 | Novotel | NV4 |
| 9 | Jack | 9/28/2022 | Novotel | NV5 |
| 10 | Grace | 9/28/2022 | Novotel | NV5 |
| 11 | Pyae Sone | 9/29/2022 | Novotel | NV1 |
| 12 | Emerald | 9/29/2022 | Novotel | NV2 |
| 13 | Oake Soe Khant | 9/29/2022 | Novotel | NV3 |
| 14 | Chaw Su Mon | 9/29/2022 | Novotel | NV4 |
| 15 | Wai Wai | 9/29/2022 | Novotel | NV4 |
| 16 | Thaw Thar | 9/29/2022 | Lottel Htoel | LH7 |
| 17 | James | 9/29/2022 | Lottel Htoel | LH7 |
| 18 | Scott | 9/30/2022 | Novotel | NV3 |
| 19 | Pauk Pauk | 9/29/2022 | Lottel Htoel | LH4 |
| 20 | Nay Min | 9/29/2022 | Lottel Htoel | LH4 |
| 21 | Bo Bo | 9/30/2022 | Novotel | NV2 |
| 22 | Kyaw Swar | 9/29/2022 | Novotel | NV2 |
| 23 | Swe Swe | 9/29/2022 | Novotel | NV3 |

**Figure 4.12: Sample Booking Information**

## Successful Booking Information



Admin

Welcome SSS! Reserve Booking History Hotels Room Info Log Out KV_Store KV1 KV2 KV3

Transaction Queue

Commit Request

| # | Room | Transaction Date | CheckIn | Customer |
|---|---|---|---|---|

Committed Transaction

| # | Room | CheckIn | Customer |
|---|---|---|---|

Success!

Customer : Stella is Booking Successful!

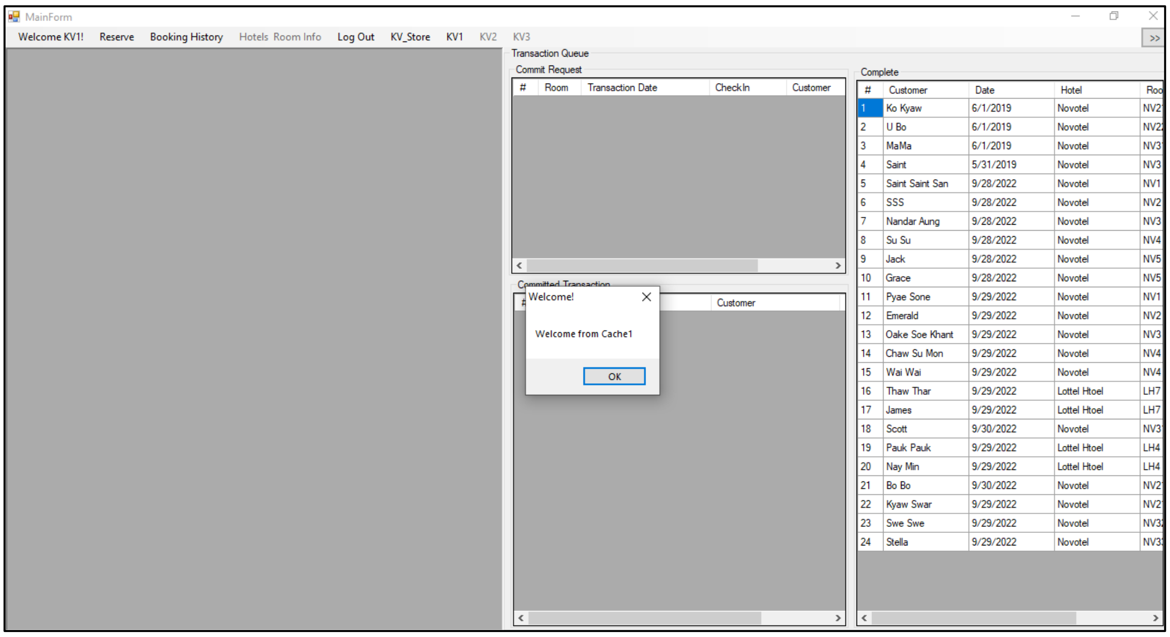
OK

Complete

| # | Customer | Date | Hotel | Roo |
|---|---|---|---|---|
| 1 | Ko Kyaw | 6/1/2019 | Novotel | NV2 |
| 2 | U Bo | 6/1/2019 | Novotel | NV2 |
| 3 | MaMa | 6/1/2019 | Novotel | NV3 |
| 4 | Saint | 5/31/2019 | Novotel | NV3 |
| 5 | Saint Saint San | 9/28/2022 | Novotel | NV1 |
| 6 | SSS | 9/28/2022 | Novotel | NV2 |
| 7 | Nandar Aung | 9/28/2022 | Novotel | NV3 |
| 8 | Su Su | 9/28/2022 | Novotel | NV4 |
| 9 | Jack | 9/28/2022 | Novotel | NV5 |
| 10 | Grace | 9/28/2022 | Novotel | NV5 |
| 11 | Pyae Sone | 9/29/2022 | Novotel | NV1 |
| 12 | Emerald | 9/29/2022 | Novotel | NV2 |
| 13 | Oake Soe Khant | 9/29/2022 | Novotel | NV3 |
| 14 | Chaw Su Mon | 9/29/2022 | Novotel | NV4 |
| 15 | Wai Wai | 9/29/2022 | Novotel | NV4 |
| 16 | Thaw Thar | 9/29/2022 | Lottel Htoel | LH7 |
| 17 | James | 9/29/2022 | Lottel Htoel | LH7 |
| 18 | Scott | 9/30/2022 | Novotel | NV3 |
| 19 | Pauk Pauk | 9/29/2022 | Lottel Htoel | LH4 |
| 20 | Nay Min | 9/29/2022 | Lottel Htoel | LH4 |
| 21 | Bo Bo | 9/30/2022 | Novotel | NV2 |
| 22 | Kyaw Swar | 9/29/2022 | Novotel | NV2 |
| 23 | Swe Swe | 9/29/2022 | Novotel | NV3 |
| 24 | Stella | 9/29/2022 | Novotel | NV3 |

**Figure 4.12: Successful Booking Information**

## Reservation Form (Replica Perspective View)



Welcome!

Welcome from Cache1

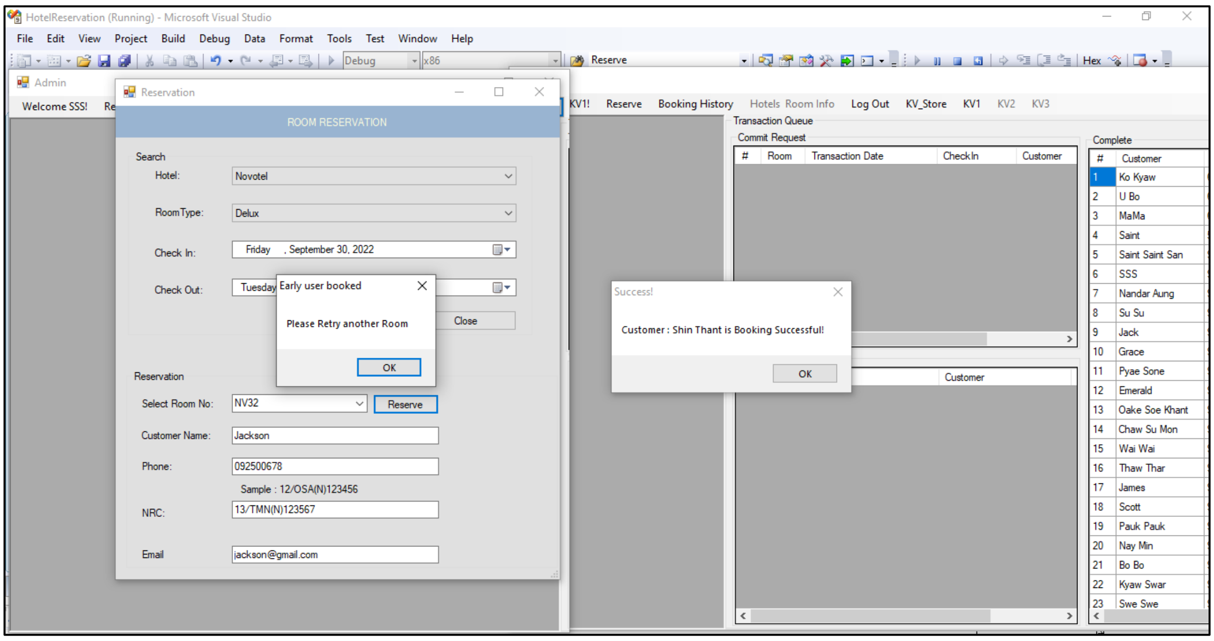
| # | Room | Transaction Date | CheckIn | Customer |
|---|---|---|---|---|

Complete

| # | Customer | Date | Hotel | Roo |
|---|---|---|---|---|
| 1 | Ko Kyaw | 6/1/2019 | Novotel | NV2 |
| 2 | U Bo | 6/1/2019 | Novotel | NV2 |
| 3 | MaMa | 6/1/2019 | Novotel | NV3 |
| 4 | Saint | 5/31/2019 | Novotel | NV3 |
| 5 | Saint Saint San | 9/28/2022 | Novotel | NV1 |
| 6 | SSS | 9/28/2022 | Novotel | NV2 |
| 7 | Nandar Aung | 9/28/2022 | Novotel | NV3 |
| 8 | Su Su | 9/28/2022 | Novotel | NV4 |
| 9 | Jack | 9/28/2022 | Novotel | NV5 |
| 10 | Grace | 9/28/2022 | Novotel | NV5 |
| 11 | Pyae Sone | 9/29/2022 | Novotel | NV1 |
| 12 | Emerald | 9/29/2022 | Novotel | NV2 |
| 13 | Oake Soe Khant | 9/29/2022 | Novotel | NV3 |
| 14 | Chaw Su Mon | 9/29/2022 | Novotel | NV4 |
| 15 | Wai Wai | 9/29/2022 | Novotel | NV4 |
| 16 | Thaw Thar | 9/29/2022 | Lottel Htoel | LH7 |
| 17 | James | 9/29/2022 | Lottel Htoel | LH7 |
| 18 | Scott | 9/30/2022 | Novotel | NV3 |
| 19 | Pauk Pauk | 9/29/2022 | Lottel Htoel | LH4 |
| 20 | Nay Min | 9/29/2022 | Lottel Htoel | LH4 |
| 21 | Bo Bo | 9/30/2022 | Novotel | NV2 |
| 22 | Kyaw Swar | 9/29/2022 | Novotel | NV2 |
| 23 | Swe Swe | 9/29/2022 | Novotel | NV3 |
| 24 | Stella | 9/29/2022 | Novotel | NV3 |

Figure 13: Reservation Form (Replica Perspective View)

## Not Successful Booking Information



Search

| | |
|---|---|
| Hotel: | Novotel |
| RoomType: | Delux |
| Check In: | Friday , September 30, 2022 |
| Check Out: | Tuesday |

Early user booked

Please Retry another Room

Reservation

| | |
|---|---|
| Select Room No: | NV32 |
| Customer Name: | Jackson |
| Phone: | 092500678 |
| | Sample : 12/OSA(N)123456 |
| NRC: | 13/TMN(N)123567 |
| Email | jackson@gmail.com |

Success!

Customer : Shin Thant is Booking Successful!

| # | Room | Transaction Date | CheckIn | Customer |
|---|---|---|---|---|

Complete

| # | Customer |
|---|---|
| 1 | Ko Kyaw |
| 2 | U Bo |
| 3 | MaMa |
| 4 | Saint |
| 5 | Saint Saint San |
| 6 | SSS |
| 7 | Nandar Aung |
| 8 | Su Su |
| 9 | Jack |
| 10 | Grace |
| 11 | Pyae Sone |
| 12 | Emerald |
| 13 | Oake Soe Khant |
| 14 | Chaw Su Mon |
| 15 | Wai Wai |
| 16 | Thaw Thar |
| 17 | James |
| 18 | Scott |
| 19 | Pauk Pauk |
| 20 | Nay Min |
| 21 | Bo Bo |
| 22 | Kyaw Swar |
| 23 | Swe Swe |

Figure 14: Reservation Form (Replica Perspective View)

# CHAPTER 5
# CONCLUSION

A system of this kind may be used in the development of applications for companies that have several branch offices. This system presented an architecture based on fully replication of relational database on key-value store system where the key-value store is used for read-only transactions. This proposed architecture ships transaction logs from relational database to the key-value store and applies them in such a way that the state of key-value store is exactly the same as the relational database. To reduce the replica lag in the key- value store side, proposed a novel concurrency control algorithm that guarantees a predefined serialization order (the one same as the order in transaction log).

## 5.1 Advantages of the System

Deadlock is avoided because only one transaction is allowed to proceed (serializable). To increase system reliability and availability. Storing data at multiple sites allows the system to continue working even though some sites may have failed. This system consists of a middleware architecture providing transparency for persistent object state replication while guarantees several consistency levels. This architecture that they are very interested in a serializable transactional behavior that guarantees an eager replication to all nodes, as well as in the development of recovery techniques so no back-up copies are necessary and alive nodes may continue working independently of a node failure.

Our main differences with protocols currently developed for the proposed system are:

**First**, it utilizes simple group communication primitives such as reliable FIFO multicast and unicast;

**Second**, it provides a deadlock prevention technique which is flexible enough in order to be based on serializable transaction characteristics and,

**Third**, it does not need to persistently store any data backup associated to the concurrency protocol.

## 5.2 Limitations and Further Extensions

This system can take only for booking. And, this system cannot contain any payment method with bank and cannot for transaction security. It Implements the ticket sales system for approving consistency and concurrency control at the distributed database by this architecture. Since it is the distributed database system, it depends on the server database and client database.

The maximum amount of tickets (especially pair seats); can be purchased as a further extension. This proposed system can be extended priority-based validation. Moreover, the system with caching data in client sides and working with those cached data while server is down will improve and solve the failures of central database system. Updating will be continued from this software, online reservation can be extended for forwarding payroll with other bank services.

## AUTHOR'S PUBLICATION

[1] Saint Saint San, Kyi Kyi Win, Concurrency Control on Transactional Replication, University of Computer Studies, Yangon, Myanmar, 2022.

## REFERENCES

[1] “A Lock Based Algorithm for Concurrency Control and Recovery in a Middleware Replication Software Architecture”, J.E. Armend´ariz and J.R. Gonz´alez de Mend´ıvil, Dpto. de Matem´atica e Inform´atica, Universidad P´ublica de Navarra, Campus Arrosad´ıa s/n, 31006 Pamplona, Spain, Email: {enrique.armendariz, mendivil}@unavarra.es

[2] B. Kemme, A. Bartoli, and ¨O. Babao˘glu. Online reconfiguration in replicated databases based on group communication. In Proc. of the International Conference on Dependable Systems and Networks (DSN 2001), Goteborg, Sweden, pages 117–127, June 2001.

[3] “COPLA: A Platform for Eager and Lazy Replication in Networked Databases.”, Francesc D. Mu˜noz-Esco´ı and Luis Ir´un-Briz and Hendrik Decker and Josep M. Bernab´eu-Aub´an, Institut Tecnol`ogic d’Inform`atica, Universitat Polit`ecnica de Val`encia.

[4] F.D. Mu˜noz-Esco´ı, L. Ir´un-Briz, P. Gald´amez, J.M. Bernab´eu-Aub´an,J. Bataller, and M.C. Ba˜nuls. GlobData: “Consistency protocols for replicated databases.”In Proc. of the IEEE-YUFORIC’, Valencia, Spain, pages 97–104.

[5] F.D. Mu˜noz-Esco´ı, L. Ir´un-Briz, P. Gald´amez, J.M. Bernab´eu-Aub´an, J. Bataller, and M.C. Ba˜nuls. GlobData: Consistency protocols for replicated databases. In Proc. of the IEEE-YUFORIC’2001, Valencia, Spain, pages 97–104, November 2001.

[6] J.E. Armend´ariz, J.R. Gonz´alez de Mend´ıvil, and F.D. Mu˜noz-Esco´ ı. Working on GlobData: An efficient software tool for global data access. In Proc. of Workshop on Research and Education in Control and Signal Processing (REDISCOVER 2004), Accepted, Cavtat, Croatia, June 2004. IEEE Press.

[7] J.E. Armend´ariz, J.J. Astrain, A. C´ordoba, J. Villadangos, and J.R. Gonz´alez de Mend´ıvil. A persistent object storage service on replicated architectures. In Proc. of VI Workshop Iberoamericano de Ingenier´ıa de Requisitos y Ambientes Software (IDEAS 03), Asunci´on, Paraguay, pages 133–144, April 2003.

[8] J.E. Armend´ariz, J.J. Astrain, A. C´ordoba, and J. Villadangos. Implementation of an object query language for replicated architectures. In Proc. of VIII

Jornadas de Ingenier´ıa del Software y Bases de Datos (JISBD 03), Alicante, Spain, pages 441–450, November 2003.

[9] L. Rodrigues, H. Miranda, R. Almeida, J. Martins, and P. Vicente. Strong replication in the globdata middleware. In Proc. Workshop on Dependable Middleware-Based Systems (Supplemental Volume of the 2002 Dependable Systems and Networks Conference), Washington D.C., USA, pages G96–G104, June 2002.

[10] L. Rodrigues, H. Miranda, R. Almeida, J. Martins, and P. Vicente. The GlobData fault-tolerant replicated distributed object database. In Proc. of the First Eurasian Conference on Advances in Information and Communication Technology, Teheran, Iran, October 2002.

[11] M.J. Carey, and M. Livny. Conflict detection tradeoffs for replicated data. ACM Trans. on Database Sys., 16(4):703–746, December 1991.

[12] M. Xiong, K. Ramamritham, J. Haritsa, and J.A. Stankovic. MIRROR: A state-conscious concurrency control protocol for replicated real-time databases. In Proc. of 5th IEEE Real-Time Technology and Applications Symposium, pages 100–110, Vancouver, Canada, June 1999.

[13] PostgreSQL Home Page. http://www.postgresql.org June 2004.

[14] P. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. Addison-Wesley, USA, 1987.

[15] V. Hadzilacos, and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. Distributed Systems, Second Edition. S. Mullender, ACM Press/Addison-Wesley, USA, 1993.