A Common Database and a Transaction Pool to Reduce Data Redundancy and to Maximize Database Throughput in a Comprehensive Information System of a Particular Domain of Interests

Mohammad Ghulam Ali, *Member, IACSIT,* and Mohammad Ghulam Murtuza

***Abstract*—Main objective of writing this paper is to adopt a common database in a Comprehensive Information System of a particular domain of interests. This will result in reducing data redundancy, in reducing database maintenance time, in simplifying the development phase of the front-end interface, in minimizing the size of the system, in maximizing the database throughput and in implementing a transaction pool. The transaction pool will reduce the number of times new connection objects are created, will promote connection object reuse, will quicken the process of getting a connection and will control the amount of resources spent on maintaining connections. We also illustrate some examples of a transaction pool to connect transaction databases with a common database. We have also illustrated how to create non-visual transaction objects within the system.**

***Index Terms*—Relational database, information system, client-server, transaction pool, distributed databases.**

I. Introduction

Keeping in mind the progress in communication and database technologies (concurrency, consistency and reliability) has increased the data processing potential. Various protocols are proposed and implemented for network reliability, concurrency, atomicity, consistency, recovery and replication. Keeping in mind these, a demand of a Comprehensive Information System for any organization/institution is also tremendously increased for the automation of manual processes and for the decision supports with quick information retrieval either using a Client-Server Database Management Systems (DBMS) or a Web-based systems. In this paper we have considered an approach of Client-Server DBMS.

In this paper we propose a method to adopt a common database concept during design and development phase of the Comprehensive Information System that will reduce data redundancy, will reduce database maintenance time, will simplify the development phase of the front-end interface, will minimize the size of the system, will maximize the database throughput and will explain how to fetch common information in various sub-domains (as stated in the paragraph four of the introduction section) of a particular domain of interests through a transaction pool. The comprehensive information system of a particular domain of interests is based on the Client-Server Database Model [2].

In this paper we are considering functionalities of the academic institution. The main domain of interests is Academic Programme. The Comprehensive Information System Service for any academic institution may provide information on annual budget, perspective budget, budget vs. actual expenditure, donors, investments, civil constructions, projects, recruitment, establishment, stores and purchase, finance and accounts, salary and wages, leave, retirement and pension, housing, medical, grants, loans, provident fund, insurance, students admission, semester examination, student grades, course information, library, trainings and placement, awards, alumnus, research and results for the decision support. Through the service, students can also submit applications for academic leave or report their return from leave, browse course schedules and course syllabi, check grades, apply for scholarships, print tuition bills and payment receipts and find many other useful information.

In this paper we consider only four sub-domains which are Recruitment & Establishment, Salary, Finance & Accounts and Students.

If we adopt a common database that stores common information in number of tables that will be used in all sub-domains according to their requirements, then database of all sub-domains will not create those common tables separately and will not maintain that information. Hence the fact, the data redundancy will reduce, no additional interface will be required to create separately in all sub-domains to maintain all these common information, and hence the size of the integrated/comprehensive information system will reduce and the development time of the comprehensive information system will also reduce.

Since every sub-domain will require common information in the comprehensive information system and for this it is not necessary to create tables and store all those information in all sub-domains, every sub-domain will fetch common information from a common database through a transaction pool. The process of establishing a database connection can be time consuming depending upon network connectivity. Connection pooling is a viable option if the network is an issue, and database server resources are

Manuscript received June 19, 2011; revised September 22, 2011

Mohammad Ghulam Ali, System Engineer, Academic Post Graduate Studies and Research, Indian Institute of Technology Kharagpur, West Bengal, INDIA, (e-mail: ali@hijli.iitkgp.ernet.in, ali_iit@yahoo.com), (Phone: 91-3222-282056, 282188, 220176).

Mohammad Ghulam Murtuza, Computer Programmer, Research Service Center, Post Graduate Department of Physics, Tilka Manjhi Bhagalpur University, Bhagalpur, Bihar, INDIA, (e-mail: mgm_rsc@yahoo.co.uk), (Phone : 91-641-2405729)

available[4]. A connection pool is a cache of database connection objects. The objects represent physical database connections that can be used by an application to connect to a database. At run time, the application requests a connection from the pool. If the pool contains a connection that can satisfy the request, it returns the connection to the application. If no connections are found, a new connection is created and returned to the application. The application uses the connection to perform some work on the database and then returns the object back to the pool. The connection is then available for the next connection request. Connection pools promote the reuse of connection objects and reduce the number of times that connection objects are created. Connection pools significantly improve performance for database-intensive applications because creating connection objects is costly both in terms of time and resources. Tasks such as network communication, reading connection strings, authentication, transaction enlistment, and memory allocation all contribute to the amount of time and resources it takes to create a connection object. In addition, because the connections are already created, the application waits less time to get the connection. Connection pools often provide properties that are used to optimize the performance of a pool. The properties control behaviors such as the minimum and maximum number of connections allowed in the pool or the amount of time a connection can remain idle before it is returned to the pool. The best configured connection pools balance quick response times with the memory spent maintaining connections in the pool. It is often necessary to try different settings until the best balance is achieved for a specific application, Applications that are database-intensive generally benefit the most from connection pools. As a policy, applications should use a connection pool whenever database usage is known to affect application performance. Benefits of Using Connection Pools are 1) reduces the number of times new connection objects are created, 2) promotes connection object reuse, quickens the process of getting a connection, 3) reduces the amount of effort required to manually manage connection objects, 4) minimizes the number of stale connections, and 5) controls the amount of resources spent on maintaining connections.[7].

Therefore, we propose to construct a separate common database that will store all those information in number of tables. This will certainly reduce data redundancy, will reduce the time to be taken in the database maintenance, will simplify the development of the front-end interface, will minimize the size of the system, will maximize the database throughput and will implement transaction pool that will save time in database connectivity. When we adopt a common database that will store information in number of common tables will be used throughout the comprehensive information system.

To use a transaction pool mechanism to fetch common information from a common database in any sub-domain, we consider in this paper PowerBuilder as a front-end development tool and Sybase Relational Database as a back-end for all sub-domains. The database for all four sub-domains and for a common database may be created in five different Sybase Database Servers or in a single Sybase database server. Here we have considered five different Sybase Database Servers with distinct network identification. In stead of PowerBuilder, we can also consider or choose other front-end development platforms [3,4,5,6,7] to develop a comprehensive information system using a transaction pool mechanism and to fetch common information from a common database and process then in the any sub-domain. Please see more about transaction pool in [8,9,10,11,12,13].

II. Prototype of The Academic Institution Domain

Through figure 1 we illustrate prototype of the Academic Institution domain, a Comprehensive Information System on distributed databases over computer LAN. Each database server has unique network name such as in this paper for a sub-domain Recruitment and Establishment database we named SYBASE2, for a sub-domain Finance and Accounts database we named SYBASE3, for a sub-domain Salary database we named SYBASE4, for a sub-domain Students database we named SYBASE5 and for a Common Database we named SYBASE1. Each unique network name of the database server has IP address with unique network listeners TCP port. On the basis of servers' interfaces information, all transaction objects are created and a transaction pool is defined.

We illustrate an example of SYBASE interfaces file which is available in the $SYBASE directory that stores following information:

```
## SERVERNAME on HOSTNAME
##      Services:
##           query   tcp    (Port Nos)
##           master  tcp    (Port Nos)

SYBASE1
     query  tcp  ether  hpaclass  5000
     master  tcp  ether  hpaclass  5000
```

where SYBASE1 is the name of the SYBASE database engine which is running on the hpaclass UNIX server host having unique server IP. We can run more than one SYBASE engine in the same host on different unique network listeners TCP port. The interfaces file also contains information about all other database servers ensuring that all Sybase products on the network interact with one another.

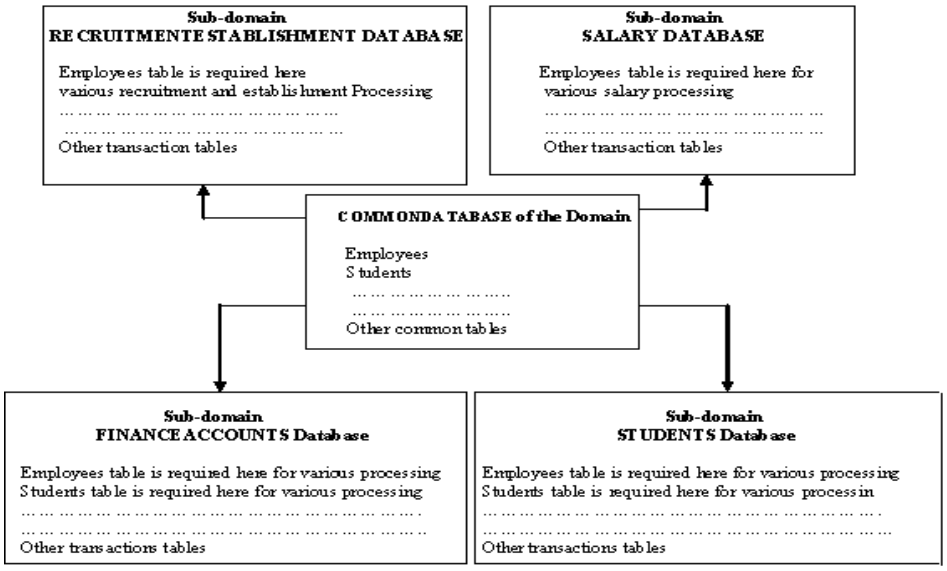

Fig. 1.Prototype of the Academic Institution Domain

III. Client-Server Database Model

We discuss few about Client-Server architecture [2] in this paper as shown in figure 2.

A. Client

Provides appropriate interfaces through a client software module to access and utilize the various server resources. Applications running on clients utilize an Application Program Interface (API) to access server databases via standard interface such as ODBC (Open Database Connectivity) standard or JDBC for Java programming access.

B. Server

Provides database query and transaction services to the clients

C. Client-Server Architectures

1) Work division: Thin client

Client implements only the graphical user interface and Server implements business logic and data management.

2) Work division: Thick client

Client implements both the graphical user interface and the business logic and Server implements data management

D. Two and three Tier Client-Server Architecture

In two-tier client/server architecture, the client communicates directly with the database server. The application or business logic either resides on the client or on the database server.

In a three-tier or multi-tier environment, the client implements the presentation logic (thin client). The business logic is implemented on an application server(s) and the data resides on database server(s).

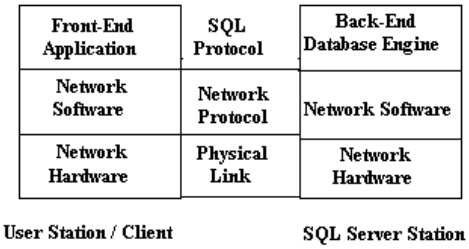

Fig. 2.Client-Server Database Model

IV. Transaction Pool

PowerBuilder is an event-driven, object-oriented front-end development tool with powerful high level Powerscript scripting language.

To optimize database processing, an application can pool database transactions. Transaction pooling [1] maximizes database throughput while controlling the number of database connections that can be open at one time. When we establish a transaction pool, an application can reuse connections made to the same data source.

When an application connects to a database without using transaction pooling, PowerBuilder physically terminates each database transaction for which a DISCONNECT statement is issued.

When transaction pooling is in effect, PowerBuilder logically terminates the database connections and commits any database changes, but does not physically remove them. Instead, the database connections are kept open in the transaction pool so that they can be reused for other database operations.

Transaction pooling can enhance the performance of an application that services a high volume of short transactions to the same data source.

Sets up a pool of database transactions for an application, SetTransPool allows us to minimize the overhead associated with database connections and also limit the total number of database connections permitted.

We create transaction objects and define transaction pool,

we write scripts using a powerful powerscript language in the open event of the comprehensive information system.

In a PowerBuilder database connection, a transaction object is a special non-visual object that functions as the communications area between PowerScript and the database as shown in figure 3. The transaction object specifies the parameters that PowerBuilder uses to connect to a database. We must establish the transaction object before we can access the database from our application.

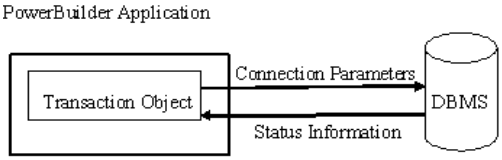

Fig. 3.Transaction object to access database

In order for a PowerBuilder application to display and manipulate data, the application must communicate with the database in which the data resides. To communicate with the database from our PowerBuilder application; we assign the appropriate values to the transaction object, connect to the database, assign the transaction object to the DataWindow control, perform the database processing and Disconnect from the database.

When we create server application connection/transaction objects the first thing our object will probably make a connection to the database. This is a time consuming process and can take between 2 - 5 seconds depending on network traffic and our DBMS.

To minimize this connection time we will setup a transaction pool. The transaction pool works by not releasing previously made connections to the database.

When we connect to the database for the first time the development tool checks to see if there is a transaction that matches our request in the pool. If not, it performs the normal processing, if it finds a match, we are pointed to the old alive transaction and we go with minimal connection time.

When we disconnect from the server the transaction is sent back to the pool. We can specify a minimum and maximum amount of connections. This will be determined by the type of processing performed by our application server.

We can also specify the timeout period for a user to wait for a transaction to become available.

To setup the transaction pool we go to the open event of our application and add following codes:

```
//Create a non-visual transaction object to connect to a
CommonDatabase Database
Transaction To_ CommonDatabase
To_ CommonDatabase = Create Transaction
To_ CommonDatabase.DBMS = "SYC Sybase System 10
CTLB"
To_CommonDatabase.database="CommonDatabase"
To_ CommonDatabase.userid = "******"
To_ CommonDatabase.dbpass = "******"
To_ CommonDatabase.logid = "******"
To_ CommonDatabase.logpass = "******"
To_ CommonDatabase.servername = "SYBASE1"
To_ CommonDatabase.dbparm = ProfileString ("pb.ini",
"CommonDatabase", "dbparm", "")

//Create a non-visual transaction object to connect to a
RecruitmentEstablishment Database
Transaction To_ RecruitmentEstablishment
To_RecruitmentEstablishment = Create Transaction
To_RecruitmentEstablishment.DBMS = "SYC Sybase
System 10 CTLB"
To_RecruitmentEstablishment.database=
"RecruitmentEstablishment"
To_RecruitmentEstablishment.userid = "******"
To_RecruitmentEstablishment.dbpass = "******"
To_RecruitmentEstablishment.logid = "******"
To_RecruitmentEstablishment.logpass = "******"
To_RecruitmentEstablishment.servername="SYBASE2"
To_RecruitmentEstablishment.dbparm= ProfileString
("pb.ini", "RecruitmentEstablishment", "dbparm", "")

//Create a non-visual transaction object to connect to a
FinanceAccounts Database
Transaction To_ FinanceAccounts
To_ FinanceAccounts = Create Transaction
To_ FinanceAccounts.DBMS = "SYC Sybase System 10
CTLB"
To_ FinanceAccounts.database = "FinanceAccounts"
To_ FinanceAccounts.userid = "******"
To_ FinanceAccounts.dbpass = "******"
To_ FinanceAccounts.logid = "******"
To_ FinanceAccounts.logpass = "******"
To_ FinanceAccounts.servername = "SYBASE3"
To_ FinanceAccounts.dbparm = ProfileString ("pb.ini",
"FinanceAccounts", "dbparm", "")

//Create a non-visual transaction object to connect to a
Salary Database
Transaction To_ Salary
To_ Salary = Create Transaction
To_ Salary.DBMS = "SYC Sybase System 10 CTLB"
To_ Salary.database = "Salary"
To_ Salary.userid = "******"
To_ Salary.dbpass = "******"
To_ Salary.logid = "******"
To_ Salary.logpass = "******"
To_ Salary.servername = "SYBASE4"
To_ Salary.dbparm = ProfileString ("pb.ini", "Salary",
"dbparm", "")

//Create a default non-visual transaction object to connect to
a Students Database
sqlca.DBMS = "SYC Sybase System 10 CTLB"
sqlca.database = "Students"
sqlca.userid = "******"
sqlca.dbpass = "******"
sqlca.logid = "******"
sqlca.logpass = "******"
sqlca.servername = "SYBASE5"
sqlca.dbparm = ProfileString ("pb.ini", "Students",
"dbparm", "")
connect using sqlca;
```

There is a file called **pb.ini** that will come with a Comprehensive Information System and that will initialize database connectivity between the client interface and the

database servers. In case of the thick client, pb.ini file resides in the client PC where client software is installed and supports client software to establish connection to the database server. In case of the thin client, pb.ini resides in the application server, and the thin client just contains published icon of the client application such as through citrix metaframe. All five databases for four different sub-domains and for a common database are created in the five different database engine servers and their connectivity information are stored in pb.ini file.

```
// set up transaction pool for a comprehensive information
system
this.SetTransPool( Min, Max, TimeOut )   or
```

where ***this*** represents to existing application.

```
myapp.SetTransPool( Min, Max, TimeOut )
```

where myapp is the name of the existing application, Min is the minimum number of transactions to be kept open in the pool , Max is the maximum number of transactions that can be open in the pool and TimeOut is the number of seconds to allow a request to wait for a connection in the transaction pool.

Example

```
this.SetTransPool(12,16,10)
```

In the above example, ***this*** represents that the transaction pool is defined in the existing application; statement specifies that up to 16 database connections will be supported in this application, and that 12 connections will be kept open once successfully connected. When the maximum number of connections has been reached, each subsequent connection request will wait for up to 10 seconds for a connection in the pool to become available. After 10 seconds, the application will return an error.

In our proposed prototype we can set value as

```
this.SetTransPool(5,16,10)
```

where *Min* is 5, *Max* is 16 and Timeout is 10.

To perform operations in multiple databases at the same time, we need to use multiple transaction objects, one for each database connection. We must declare and create the additional transaction objects before referencing them.

V. How to Fetch Data From a Common Database and Process Then In The Application of Any Sub-Romain Of a Comprehensive Information System

A. Open connection object, check connectivity and close connection object

```
Connect using to_CommonDatabase;
If  To_CommonDtabase.sqlcode <> 0 then
    This.title = "Connection Failed";
Else
    This.title = "Connected";
    Disconnect using To_CommonDatabase;
End if;
```

B. Using more than one connections in any SQL statement that are used in the development of the client interface

Example 1

```
Connect using To_CommonDatabase;
Select a.RollNo, a.NameOfStudents, b.CGPA,
b.YearOfPassing
Into :RollNo, :NameOfStudents,:CGPA, :YearOfPassing
 from Students  a, DegreeAward b
Where a.RollNo = b.RollNo
using to_CommonDatabase;
disconnect using To_CommonDatabase;
```

In the above SQL statement we join two tables from two different databases in which first database is a common database and the second one database is related to the sub-domain. Where default connection object is sqlca for a Students database and to_CommonDatabase transaction object is for a Common database and connections are taking through a transaction pool. The above SQL statement will fetch data from two different databases using a transaction pool. A table DegreeAward is a transaction table of the Students Database and Students is a common table created in the Common Database.

Example 2

```
Connect using To_Salary;
Connect using To_CommonDatabase;
Select a.EmployeeCode, a.NameOfEmployee, b.BasicScale,
b.GrossSalary
Into :EmployeeCode,  :NameOfEmployee, :BasicScale, :Gr
ossSalary
from Employees  a, CurrentMonthSalary b
Where a.EmployeeCode = b.EmployeeCode
using To_CommonDatabase;
Disconnect using To_Salary;
Disconnect using To_CommonDatabase;
```

where we use two transaction objects To_Salary for a Salary database and To_CommonDatabase for a Common Database from a transaction pool. The above SQL statement will fetch data from two different databases using a transaction pool. The table CurrentMonthSalary is a transaction table of the Salary Database and Employees is a common table created in the Common Database.

Example 3

```
Connect using To_RecruitmentEstablishment;
Connect using To_CommonDatabase;
declare cursor_update_increment cursor for
Select EmployeeCode, NameOfEmployee, DateOfJoining
from Employees  where EmployeeFlag="Y"
using To_CommonDatabase;
open cursor_update_increment;
do WHILE true
fetch cursor_update_increment into
Into :EmployeeCode,  :NameOfEmployee, :DateOfJoining;
// Update Increment Transaction Table of the
RecruitmentEstablishment
FullYear = SystemDate-DateOfJoining ;
If FullYear = 365 then
      Update Increment Table
End if;
//
loop;
close cursor_update_increment;
Disconnect using To_ RecruitmentEstablishment;
Disconnect using To_CommonDatabase;
```

where we use two transaction objects To_RecruitmentEstablishment for a Recruitment and Establishment Database and To_CommonDatabase for a Common Database from a transaction pool. The above SQL statement will fetch row-wise data from a common database using transaction pool and a cursor and process then. A transaction table Increment of a Recruitment and Establishment Database will be updated with satisfying the where clause

Similarly, we can use all other common tables of a Common Database in the database of all sub domains as and when required according to the situation of the system requirements with more than one transaction connections in the development of the entire system. We can fetch data from a common database in any sub-domain's system, process then in different way according to the situation and requirements such as we can straight way fetch data into variables and then go for further processing, we can fetch row-wise data using a cursor and go for further processing, we can fetch data from a common database and then update the transaction database of any sub-domain.

VI. Conclusion

This paper will provide a method to develop a Comprehensive/Integrated Information System for any organization/institution using a Client-Server DBMS technology with a better quality and performance. This paper will provide an adequate support in an area of the Software Engineering. We can also use the same transaction pool technique in the development of the Comprehensive Information System using a different front-end development platforms. In future we shall address other issues involved in the development of the Comprehensive Information System.

Acknowledgements

We have tested this approach in our one project. This is an effective technique that we have proposed.

References

[1] SYBASE, Application Techniques, PowerBuilder 11.5 online manual.
[2] Al-Sukairi, Abdallah.: 'Advance Database Systems', King Fahd University of Petroleum & Minerals, Information & Computer Science Department.
[3] http://msdn.microsoft.com / en-us library / 8xx3tyca (v=vs.71).aspx, "Connection Pooling for the .NET Framework Data Provider for SQL Server".
[4] http://www.techrepublic.com / article/take-advantage-of-adonet-connection –pooling / 6107854, "Take advantage of ADO.NET connection pooling".
[5] http://publib.boulder.ibm.com / infocenter / db2luw/ v8/index.jsp?topic=/com.ibm.db2.udb. doc/conn/c0006170.htm, "Connection pooling".
[6] Dave Robinson, "Database Access with WebLogic Server", March 26, 2003.
[7] "Oracle® Universal Connection Pool for JDBC Developer's Guide, 11g Release 2 (11.2), July 2009.
[8] http://www.datadirect.com/docs/public/ tutorials/ Connect-For-ADO.NET/dotnet-connpool.pdf, "Connection Pooling in .NET Applications", DATADIRECT TECHNOLOGIES JANUARY 2006.
[9] Deb Erickson, Shawn Lauzon, Melissa Modjeski, "WebSphere Connection Pooling" First Edition (August 2001).
[10] Mark Matthews, "Connection pooling with MySQL Connector/J", May 29, 2003.
[11] Michael J. Shelton, "Connection Pooling in PostgreSQL". June 2004.
[12] http://pdf.coreservlets.com / first-edition / CSAJSP-Chapter18.pdf, "JDBC and Database Connection Pooling". Prentice Hall and Sun Microsystems.
[13] Giles Winstanley, "DBPool: Java Database Connection Pooling", May 2010.

Mohammad G. Ali He obtained the degree of Master Diploma in Computer Science (1991) and Master of Science in Mathematics (1993) with 1st class. He stood 1st in the Computer Science in the University. He is a Fellow (FBCS), British Computer Society, the Chartered Institute for IT, UK. He is a life member of IAENG, Hong Kong and IACSIT, Singapore. His two papers were published in the International Journal of Computer Science and Information Security, USA in the month of November 2009.

Another paper was published in the Global Journal of Computer Science and Technology, USA in the month of April, 2010. Another paper was accepted in the International Conference, IASTED, ACIT-ICT 2010, Russia. Another paper was published in International Journal of Computer Applications, Foundation of Computer Science, New York, USA in the month of September 2010. Another paper was published in the International Journal of Computer Theory and Engineering, International Association of Computer Science and Information Technology, Singapore. His one paper was accepted in the international conference, (ICMLC-2011), Singapore which was held in Feb 26-28, 2011 (The conference was sponsored by the IACSIT, IEEE, IE). He is a member of the Editorial Board of IJCA, USA and IJCTE, Singapore. He is a member of Reviewer Board of IAENG IJCS, Hong Kong. He was a Peer Reviewer of the International Conferences, ICMLC-2011, Singapore and IEEE ICCSIT 2011, China. He is a System Engineer Grade I in the Indian Institute of Technology, Kharagpur, West Bengal, India. He is associated with IT project Management, System Analysis and Design Methods, System Development Life Cycle, Programming, Implementation and Maintenance of Client-Server DBMSs and Web Applications Development. He is also associated with Database Administration, Web Server Administration, System Administration and Networking of the Institute. He has deployed many small to big projects in the Institute Network. He has been guiding undergraduate and post graduate students of the Institute in their projects. His areas of research are Parallel and Distributed Computing (Heterogeneous Distributed Databases), Software Engineering, Networking and Network Security.

Mohammad G. Murtuza He obtained the degree of Master of Science in Mathematics from the Bhagalpur University and Diploma in System Analysis & Programming (Datamatics Corporation, Kolkata). His one paper was published in the Global Journal of Computer Science and Technology, USA. Another paper was published in the International Astronomical Society, Nova Science Publisher, USA.

He has been providing statistical analysis supports to research students and faculty of the University, computerizing various works of the University, analyzing and interpreting statistical data and taking computer classes of Computer Diploma Course of the University.