

**A GENERALIZED DATABASE SYSTEM FOR THE CENTRAL BANK OF BANGLADESH
FROM HETEROGENEOUS SYSTEM OF DIFFERENT COMMERCIAL BANKS**

by

Md. Mahbubur Rahman Alam

MASTER OF ENGINEERING IN INFORMATION AND COMMUNICATION TECHNOLOGY

Institute of Information and Communication Technology (IICT)

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

June, 2013

The project titled “A GENERALIZED DATABASE SYSTEM FOR THE CENTRAL BANK OF BANGLADESH FROM HETEROGENEOUS SYSTEM OF DIFFERENT COMMERCIAL BANKS”, submitted by Md. Mahbubur Rahman Alam, Roll No. M10073119P, Session: October/2007, has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Masters of Engineering in Information and Communication Technology on June 29, 2013.

Board of Examiners

-
1. Dr. Md. Saiful Islam Chairman
Professor and Director
Institute of Information and Communication Technology
BUET, Dhaka-1000, Bangladesh

 2. Dr. Md. Liakot Ali Member
Professor
Institute of Information and Communication Technology
BUET, Dhaka-1000, Bangladesh

 3. Dr. Mohammad Shah Alam Member
Assistant Professor
Institute of Information and Communication Technology
BUET, Dhaka-1000, Bangladesh

Candidate's Declaration

It is hereby declared that this report or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Md. Mahbubur Rahman Alam

Dedicated
To
My Parents and Family Members

Table of Contents

Title	Page No.
Board of Examiners	II
Candidate's Declaration	III
Dedication	IV
Table of Contents	V-X
List of Tables	XI
List of Figures	XII-XIV
Acknowledgement	XV
Abstract	XVI

Chapter 1: Introduction

1.1 In General	1
1.2 Objectives with Specific Aims and Possible Outcomes	2
1.3 Project Paper Layout	3

Chapter 2: Background, Present State and Implementation of the System

2.1 Overview	4
2.2 Implementation Issues	8

Chapter 3: Database Design for Central Bank, Linked-Server and Commercial Banks

3.1 Introduction	12
3.2 Normalization Issues	12
3.3 Oracle Database Design for the Production Server of the Central Bank	14
3.3.1 Entity and Attributes Exploration	14
3.3.1.1 Info_Categories	14

Table of Contents

Title	Page No.
3.3.1.2 Bank	15
3.3.1.3 Gathered_Data	15
3.3.1.4 Process_Info	16
3.3.1.5 Filtering_Data	17
3.3.2 Creating Tables of the Temporary Schema	18
3.4 Permanent Schema of Production Server: To Store Data Received from Temporary Schema	19
3.4.1 Entity and Attributes Exploration	20
3.4.1.1 Info_Categories	20
3.4.1.2 Bank	20
3.4.1.3 Process_Info	21
3.4.1.4 Filtering_Data	21
3.4.1.5 Gathered_Data	22
3.4.2 Creating Tables: Permanent Schema of Central Bank	23
3.5 Linked-Server Database (Middle-Tier)	24
3.5.1 Entity and Attributes Exploration	25
3.5.1.1 Info_Categories	25
3.5.1.2 Bank	25
3.5.1.3 Gathered_Data	26
3.5.1.4 Process_Info	27
3.5.1.5 Filtering_Data	27
3.5.2 Creation of Linked-Server Database and Tables in MS-SQL Server (Middle-Tier)	28
3.5.2.1 Creation of Database	28
3.5.2.2 Creation of Tables	29
3.6 Commercial Banks' Database	30
3.6.1 Commercial Banks' Database: Bank-6 Using MS-SQL Server	31
3.6.1.1 Entity and Attributes Exploration	31

Table of Contents

Title	Page No.
3.6.1.1.1 Customers	31
3.6.1.1.2 Account_Type	32
3.6.1.1.3 Accounts	33
3.6.1.1.4 Transactions	33
3.6.1.1.5 Vault	33
3.6.1.1.6 Vault_Transaction	34
3.6.1.2 Creation of Database and Tables in MS-SQL Server: Bank-6	34
3.6.1.2.1 Creation of Database with Log files using T-SQL	34
3.6.1.2.2 Creation of Database Tables	35
3.6.2 Commercial Banks' Database: Bank-1 Using Oracle Server	37
3.6.2.1 Creation of Database Tables	37
3.6.3 Commercial Banks' Database: Bank-4 using MS-Visual FoxPro Database	39
3.6.3.1 Creation of Database and Tables	40
3.6.4 Commercial Banks' Database: Bank-3 using MS-Access Database	42
3.6.4.1 Populating Data	43
3.6.5 Commercial Banks' Data: Bank-5 Using Text Files	43
3.6.6 Commercial Banks' Data: Bank-2 using MS-Excel Files	43
3.7 Summary	44
 Chapter 4: Designing the Middle-Tier Linked-Server	
4.1 Introduction	45
4.2 Linked-Servers	45
4.2.1 Linked Server Components	46
4.2.2 Managing a Linked Server Definition	46
4.3 OLE DB Provider for Oracle	49
4.4 OLE DB Provider for Jet	50
4.4.1 To Create a Linked Server to access an MS-Access Database	50

Table of Contents

Title	Page No.
4.4.2 To Create a Linked Server against an Excel Spreadsheet	51
4.4.3 To set up a Linked Server against a Text File	52
4.5 OLE DB Provider for ODBC (MS-Visual FoxPro)	53
4.6 OLE DB Provider for SQL Server	55
4.7 Summary	56
Chapter 5 :Data Gathering, Filtering and Processing	
5.1 Introduction	57
5.2 Distributed Query Architecture	57
5.3 OPENQUERY	59
5.4 Gathering Data from Linked Servers	60
5.4.1 Gathering data from all banks assuming all communication links are up and all linked-servers are running	60
5.4.1.1 Executing the Procedure to gather data when all servers are online	61
5.4.2 Gathering data from all banks assuming all communication links and all linked-servers are not online	61
5.4.2.1 Executing the Procedure to gather data when all servers are not online	63
5.4.3 Gathering data from a specific bank assuming the communication link and linked-server is online	63
5.4.3.1 Executing the Procedure to gather data for a specific server	65
5.4.4 Procedure to gather vault data when all servers are online	65
5.4.4.1 Executing Procedure to gather vault data when all servers are online	66
5.4.5 Procedure to gather vault data when all servers are not online	66
5.4.5.1 Executing Procedure to gather vault data when all servers are not online	67
5.4.6 Procedure to gather Vault data for a specific server	68

Table of Contents

Title	Page No.
5.4.6.1 Executing Procedure to gather Vault data for a specific server	69
5.4.7 Filter Data after Gathering from Linked Servers	69
5.4.7.1 Executing Procedure to Filter Data after Gathering from Linked Servers	71
5.7 Summary	71
Chapter 6:Transferring Data from Linked-Server to Production Server	
6.1 Introduction	72
6.2 MS-Data Transformation Services (DTS)	72
6.3 Configuring MS-Data Transformation Services (DTS)	72
6.4Filtering Data after receiving from Linked Servers	79
6.4.1Creating the Procedure	79
6.4.2Executing the Procedure	80
6.5 Loading the Processed Data from Temporary Schema to Permanent Schema of the Production Server	81
6.5.1 Creating the Procedure	81
6.5.2 Executing the Procedure	81
6.6 Summary	81
Chapter 7 :Interactive Form and Report Design for End-Users	
7.1 Introduction	82
7.2 Interactive Form Design	82
7.2.1 Vault Data	82
7.2.2Banks Information	88
7.2.3Status of Gathered Data	89
7.2.4Gathered Data	90
7.2.5Gathered Data by Category	91
7.2.6Process Wise Data	92

Table of Contents

Title	Page No.
7.2.7Running a Process to Filter and Load Data	93
7.3 Report Design	99
7.4 Summary	109
Chapter 8: Security Issues	
8.1 Introduction	110
8.2 Network Domain Configuration	110
8.3 Database Views	111
8.3.1 Creating Views for Bank-6 MS-SQL Server	111
8.3.2 Creating Views for Bank-1 Oracle Server	112
8.3.3 Creating Views for Bank-4 FoxPro Server	112
8.3.4 Creating View to See Filtered Data in Linked-Server	112
8.3.5 Creating View for Production Server of the Central Bank	113
8.4 Database User Security	113
8.4.1 MS-SQL Server	114
8.4.2 Oracle	116
8.4.3Others	117
8.5 Summary	118
Chapter 9: Conclusion	
9.1 Conclusion	119
9.2 Future Works	120
References	121
Appendix-I	122-130

List of Tables

Table No.	Table Caption	Page No.
Table 3.1	Info_Categories	15
Table 3.2	Bank	15
Table 3.3	Gathered_Data	16
Table 3.4	Process_Info	17
Table 3.5	Filtering_Data	17
Table 3.6	Info_Categories	20
Table 3.7	Bank	20
Table 3.8	Process_Info	21
Table 3.9	Filtering_Data	22
Table 3.10	Gathered_Data	23
Table 3.11	Info_Categories	25
Table 3.12	Bank	25
Table 3.13	Gathered_Data	26
Table 3.14	Process_Info	27
Table 3.15	Filtering_Data	28
Table 3.16	Customers	31
Table 3.17	Account_Type	32
Table 3.18	Accounts	32
Table 3.19	Transactions	33
Table 3.20	Vault	33
Table 3.21	Vault_Transaction	34
Table 4.1	sp_addlinkedserverParameter Values	48

List of Figures

Figure No.	Figure Caption	Page No.
Fig. 2.1	Architecture of the Project (Network Communication)	7
Fig. 2.2	Architecture of the Project (Data Communication)	11
Fig. 3.1	Snapshot of Temporary Schema of Central Bank's Server	14
Fig. 3.2	Snapshot of Permanent Schema of Central Bank's Server	19
Fig. 3.3	Snapshot of Linked-Server Database Design	24
Fig. 3.4	Snapshot of SQL Server Database Design for Bank-6	30
Fig. 3.5	Snapshot of Oracle Database Design for Bank-1	37
Fig. 3.6	Snapshot of Visual FoxPro Database Design for Bank-4	39
Fig. 3.7	Snapshot of Access Database Design for Bank-3	42
Fig. 3.8	Snapshot of Access Data for Bank-3	42
Fig. 3.9	Snapshot of Text Data for Bank-5	43
Fig. 3.10	Snapshot of Excel Data for Bank-2	44
Fig. 4.1	Snapshot of Oracle Database SQL* Net Service Provider Configuration	49
Fig. 4.2	Snapshot of ODBC System Data Source Configuration	53
Fig. 4.3	Snapshot of ODBC System Data Source Location	54
Fig. 4.4	Snapshot of Linked-Server Configuration for Visual FoxPro	54
Fig. 4.5	Snapshot of Security Configuration for Visual FoxPro	55
Fig. 4.6	Snapshot of MS-SQL Server Enterprise Manager	56
Fig 5.1	Snapshot of Output of the above command	61
Fig 5.2	Snapshot of Output of the above command	66
Fig. 6.1	Snapshot of Configuring oracle SQL-Net services	73
Fig. 6.2	Snapshot of Export Import Wizard of MS-DTS	73
Fig. 6.3	Snapshot of Data Source Wizard	74
Fig. 6.4	Snapshot of Selecting Destination Database	75
Fig. 6.5	Snapshot of Connecting Oracle Database	75
Fig. 6.6	Snapshot of Specify Table Copy or Query Wizard	76
Fig. 6.7	Snapshot of Selecting Source Tables	76
Fig. 6.8	Snapshot of Saving DTS Package	77

List of Figures

Figure No.	Figure Caption	Page No.
Fig. 6.9	Snapshot of Saving DTS Package	77
Fig. 6.10	Snapshot of Completing DTS Services	78
Fig. 6.11	Snapshot of Successful Transfer Message	78
Fig. 7.1	Snapshot of Oracle Forms Builder	83
Fig. 7.2	Snapshot of Data Block Wizard (Type)	83
Fig. 7.3	Snapshot of Data Block Wizard (Table)	84
Fig. 7.4	Snapshot of Data Block Wizard (Master-Detail)	84
Fig. 7.5	Snapshot of Layout Wizard (Data Block)	85
Fig. 7.6	Snapshot of Layout Wizard (Items)	86
Fig. 7.7	Snapshot of Layout Wizard (Style)	86
Fig. 7.8	Snapshot of Layout Wizard (Rows)	87
Fig. 7.9	Snapshot of Form Layout	87
Fig. 7.10	Snapshot of Form Runtime (Vault Data)	88
Fig. 7.11	Snapshot of Forms Layout (List of Banks)	88
Fig. 7.12	Snapshot of Form Runtime (List of Banks)	89
Fig. 7.13	Snapshot of Form Layout (Status of Gathered Data)	89
Fig. 7.14	Snapshot of Form Runtime (Status of Gathered Data)	90
Fig. 7.15	Snapshot of Form Layout (Gathered Data)	90
Fig. 7.16	Snapshot of Form Runtime (Gathered Data)	91
Fig. 7.17	Snapshot of Form Layout (Gathered Data by Category)	91
Fig. 7.18	Snapshot of Form Runtime (Gathered Data by Category)	92
Fig. 7.19	Snapshot of Form Layout (Gathered Data by Process)	92
Fig. 7.20	Snapshot of Form Runtime (Gathered Data by Process)	93
Fig. 7.21	Snapshot of Form Layout (Filter and Load Data)	94
Fig. 7.22	Snapshot of Form Runtime (Filter and Load Data)	94
Fig. 7.23	Snapshot of Procedure to Generate PID	95
Fig. 7.24	Snapshot of Procedure for Filter Data Button	96
Fig. 7.25	Snapshot of Procedure for Load Data Button	96
Fig. 7.26	Snapshot of Form Layout (Run Report)	97

List of Figures

Figure No.	Figure Caption	Page No.
Fig. 7.27	Snapshot of Form Runtime (Run Report)	98
Fig. 7.28	Snapshot of Procedure to Call Business Reports	98
Fig. 7.29	Snapshot of Procedure to Call Liquid Asset's Reports	99
Fig. 7.30	Snapshot of Report Wizard (Style Tab)	100
Fig. 7.31	Snapshot of Report Wizard (Type Tab)	100
Fig. 7.32	Snapshot of Report Wizard (Data Tab)	101
Fig. 7.33	Snapshot of Report Wizard (Group Tab)	101
Fig. 7.34	Snapshot of Report Wizard (Fields Tab)	102
Fig. 7.35	Snapshot of Report Wizard (Totals Tab)	102
Fig. 7.36	Snapshot of Report Wizard (Labels Tab)	103
Fig. 7.37	Snapshot of Report Wizard (Template Tab)	103
Fig. 7.38	Snapshot of Data Model	104
Fig. 7.39	Snapshot of Layout Model	104
Fig. 7.40	Snapshot of Report Parameter Form	105
Fig. 7.41	Snapshot of Report LOV	105
Fig. 7.42	Snapshot of Report Parameters	106
Fig. 7.43	Snapshot of Final Report (Run Time)	106
Fig. 7.44	Snapshot of Final Report (Run Time)	107
Fig. 7.45	Snapshot of SQL Query for Liquid Asset Report	108
Fig. 7.46	Snapshot of Report Parameters	108
Fig. 7.47	Snapshot of Final Report (Run Time)	109
Fig. 8.1	Snapshot of Domain: banking.com	110
Fig. 8.2	Snapshot of Creating Database User (Step-1)	114
Fig. 8.3	Snapshot of Creating Database User (Step-2)	114
Fig. 8.4	Snapshot of Creating Database User (Step-3)	115
Fig. 8.5	Snapshot of Creating Database User (Step-4)	115
Fig. 8.6	Snapshot of Sharing Folder Permissions (Step-1)	117
Fig. 8.7	Snapshot of Sharing Folder Permissions (Step-2)	118

Acknowledgement

First of all I would like to express my gratitude to Almighty Allah for giving me the opportunity to conduct this project. I would like to thank Professor Dr. Md. Saiful Islam, Director, Institute of Information and Communication Technology, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh. He has assigned me an interesting and useful topic, which has a wide range of application in real world. He has provided all sorts of support regarding the project work. Without his proper guidance, advice, continual encouragement and active participation in the process of this work, it would have not been possible.

The suggestions on the improvement of the work received from Professor Dr. Md. Liakot Ali and Assistant Professor Dr. Mohammad Shah Alam are also duly acknowledged with thanks.

Special thanks goes to Ms. Nazneen Sultana, Deputy Governor, Bangladesh Bank, Mr. GourangaChakrabarti, Executive Director, Bangladesh Bank, Mr. AbulKashem Md. Shirin, DMD, Dutch-Bangla Bank Ltd, Mr. S. M. MainuddinChowdhury, DMD, South-East Bank Ltd. and Mr. Shamsur Rahman Chowdhury, Head of IT, Jamuna Bank Ltd. for their cordial help and guidelines regarding this project.

I also like to give thanks to all of our classmates for various discussions regarding the project work.

Finally, I am grateful to all staffs and officers of IICT, especially Mr. Khairul Islam, for helping me continuously in various ways.

Abstract

Bangladesh Bank (BB), as a regulatory body and central bank of Bangladesh, collect, filter, organize and process periodical data from different commercial banks operating in Bangladesh for better policy, monitoring and management of the banking sector. There are a number of departments and authorities involved with this process. At present, in most of the cases, all the information is collected manually and stored in separate manual registers, text files and MS-Excel sheets in different departments which do not follow a uniform format. This decentralized information is complicated to aggregate for monitoring and policy purpose from a central point. By using current infrastructure it is not also possible for BB to access a heterogeneous banking system where 31 types of banking software are being used with 5 types of databases. To address this problem through a simulated environment a generalized database is designed to collect, filter, organize process and store periodical data in a unique standard format. Proposed system uses the Linked-Server concept of Microsoft SQL-Server to gather data by using distributed query technique. After filtering and processing in SQL-Server, data is exported to the production server of the central bank, which is running an Oracle database management system. Security issues are strictly handled in this regard. Finally, a set of parameterized reports have been designed that can be run from different departments by using central report server and will assist the regulatory body for proper monitoring, control and making policy decision. This system is simulated in the HUAWEI-Lab of the IICT department successfully. Now, with the proper permission of BB and cooperation of different commercial banks, this system can be implemented for the betterment of the banking sector and economy of the country.

Chapter1

Introduction

1.1 In General

Banking is the backbone of modern economy. In 21st century, modern banking totally depends on information and communication technology. For the economic development of Bangladesh it is very much important to develop the banking sector. Formulation and implementation of the national economic policy very much depends on information regarding banking. In Bangladesh, there are 47 commercial banks operating banking business all over the country. Most of the banks are computerized and providing online banking services through different delivery channels with different types of banking software (local, in-house, foreign) having different categories of databases. Most of the banks are providing online banking services through state-of-the-art data centers having disaster recover sites. The central and regulatory bank of Bangladesh, The Bangladesh Bank (BB), has been facing severe problems to have proper information regarding export, import, profit, tax, loans & advances, defaults, liquid assets, etc. from different commercial banks for quick policy making. Organizing data is also a great challenge. It is seen that several quarters/months/years required for gathering and processing data. It has been the main challenge of BB for policy formulation and implementation. Moreover, fraud prevention is not possible for manipulated data that is reported to BB manually. It is also very difficult for financial auditors to audit billions of records stored into the databases of different banks and verify it manually due to time, cost and lack of expertise. In this scenario, a generalized database system is required for BB to gather, filter, accumulate and process data for multiple uses from a heterogeneous banking system. Once the standard database is designed and data are stored with standard format, following advantages can be achieved.

- a) Quick collection, organization, filtering and processing of banking data.
- b) Easy monitoring and controlling the financial services of banks in Bangladesh.
- c) Quick and better financial audit.
- d) Fraud prevention related to false financial statements.
- e) Quick policy formulation and implementation.
- f) Monitoring and management of classified loans.
- g) Proper monitoring and management of liquid assets of banks.
- h) Data can be used for statistical analysis (correlation, regression, trend analysis, etc.)
- i) Data mining application can be used in long term.

1.2 Objectives with Specific Aims and Possible Outcomes

This project has the following objectives:

- a) To create a virtual banking environment of Bangladesh in the lab including BB with other commercial banks.
- b) To design a database for BB so that required accumulated information regarding finance, economics and statistics of all commercial banks operating in Bangladesh can be produced by BB.
- c) To write procedures for the central servers of different banks so that required data can be collected from banking database.
- d) To accumulate data from heterogeneous data sources of commercial banks and insert into a common database that is designed for BB.
- e) Finally, to filter and process data and send it to the central server of BB to generate policy report.

1.3 Project Paper Layout

This report consists of nine chapters. Contents of the chapters are as follows:

Chapter 1 of this report describes general discussion on the proposed project followed by background of the problem, objectives, methodology and project paper layout.

Chapter 2 of this report contains background, present state and implementation of the system.

Chapter 3 discusses database design and implementation issues of the project.

Chapter 4 mainly expresses on the designing of the linked-server.

Chapter 5 mainly focuses on the distributed query mechanism of the system including gathering, filtering and processing of data.

Chapter 6 includes transferring data from linked server to production server and final verification of the received data.

Chapter 7 contains form and report design for the end-users of the system.

Chapter 8 describes the network domain configuration and security issues of the system.

Finally, conclusion and recommendations for future works has been stated in Chapter 9.

The project paper ends with appendix that contains the database table scripts and other SQL commands related to the system.

Chapter 2

Background, Present State and Implementation of the System

2.1 Overview

Like any other central bank of the world, data collection, organization, filtering and processing is a large and complex work for the central bank of Bangladesh. In Bangladesh, 47 commercial banks have been operating their business with 7,772 branches. Among the banks, 4 nationalized commercial banks (NCBs), 30 private commercial banks (PCBs), 9 foreign commercial banks (FCBs) and 4 specialized banks (SBs) have been providing services through near about six crore accounts to the people in the country. Among the branches NCBs, PCBs, FCBs and SBs run 3685 (47%), 2382 (31%), 64 (1%) and 1641 (21%) branches respectively. Though almost all branches of private and foreign banks are computerized and providing online banking services, they use both centralize and distributed database for their operations [1].

Though almost all banks are computerized and providing online banking services; there are 31 banking software in operation using 5 types of databases. These banking software mainly use Oracle (42%), MS-SQL Server (35%), MS-Access (2%), MS-FoxPro (6%) and text files (15%) to store valuable banking data [2]-[5]. Banking data actually coming through different delivery channels, like Automated Teller Machine (ATM), Internet, Point of Sale Terminal (POST), Branch, Mobile Phones, etc. in to the central database server of data center of different banks by using different communication channels like VSAT, PSTN, Radio-Link, optical fiber, Internet (VPN), etc.,[Fig: 2.1].

Bangladesh Bank (BB), the central and regulatory bank of Bangladesh, has been facing tremendous problems to get proper information from different commercial banks for quick policy making. Moreover, BB has no online connectivity with different banks and still collects data using CD, DVD and formatted printed report by postal and courier services periodically. Based on this data the central bank produces several policy reports and change their audit and monitoring policy. Collecting data by postal or courier services is time consuming and costly. Organizing data is also a great challenge for BB. It also hampers policy making and implementation process of BB [6]. BB periodically collects data related to export, import, profit, tax, loans & advances, defaults, liquid asset, etc. for better economic policy. Starting from the generation of banking data through delivery channels to policy report, several critical steps are involved in this system. For each step it is necessary to accumulate a lot of information that is very much important for the BB. The data are again maintained and used by a number of authorities. The important components in this system are collection, filtering, processing and generating reports. However, it is found that several months or quarters or even years required for gathering and processing these data. Moreover, fraud prevention (intentional over/under shown amounts) is not possible for manipulated data that is reported to BB manually. For example, if a bank collects Tk. 5 crore as government taxes & duties and reports that the amount collected is only Tk. 2 crore; it is very difficult for BB to audit and verify it manually due to time, cost and human resource constraints [7]-[8]. At present, in most of the cases, all the information is stored in separate manual registers, text files and MS-Excel sheets in different departments which do not follow a uniform format. This decentralized information is complicated to aggregate for monitoring and policy purpose from a central point. By using current infrastructure it is not also possible for BB to access a heterogeneous banking system where 31 types of banking software are being used with 5 types of database [9]-[12]. But there

exists no attempt to integrate those data. No initiative has been taken yet from BB to develop a system to solve such problems.

Though required data is periodical, collecting data for BB from heterogeneous databases into a common platform, like Oracle, through online connectivity using linked database server concept would be a better solution. In this regard, a generalized database is needed for the Bangladesh Bank, for easy integration of different databases of different commercial banks operating in Bangladesh. Common procedures may be written for different types of database by using SQL and install it into the banking software of different banks. Those procedures will collect data periodically and send it to BB through online connectivity. After collecting data from all banks required policy information can be generated quickly and accurately from the server located at BB. In this regard, a simulated banking environment is created with necessary technology to solve this problem by developing the project. After successful testing of the project BB may use it to overcome the current problems regarding this issue. Finally, the data will assist the regulatory body for proper monitoring, control and making policy decision.

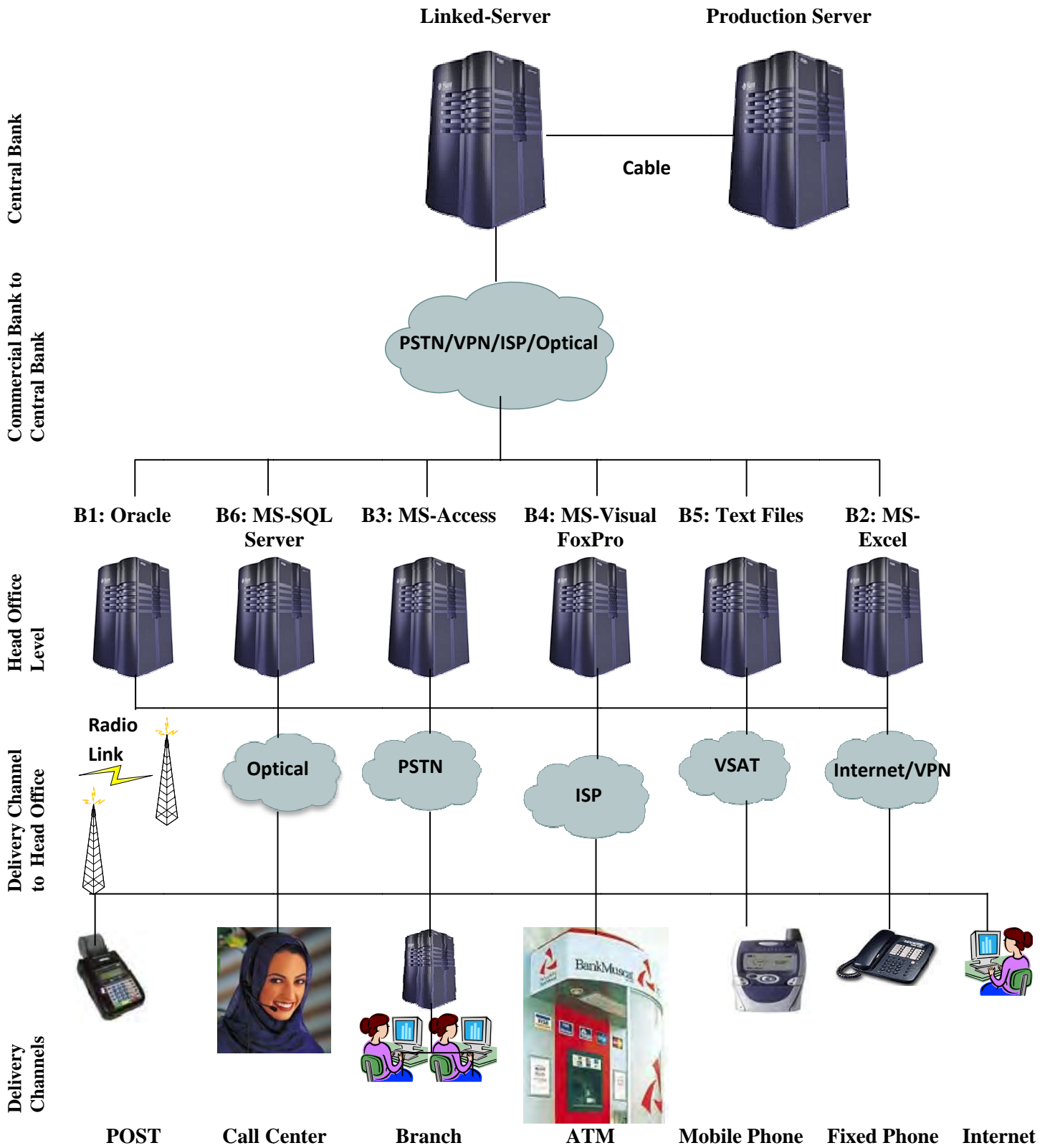


Fig. 2.1: Architecture of the Project (Network Communication)

2.2 Implementation Issues

In this project, we have considered six commercial banks; namely, B1, B2, B3, B4, B5 and B6. We also considered these banks are running under the guidance of the Central Bank. In the Central Bank there are two servers- a production server running Oracle and a Linked-Server using MS-SQL Server database. Bank-1 (B1) is a high-tech Private Commercial Bank (PCB) uses Oracle database for their banking software. MS-Excel is used by Bank-2 (B2), a Nationalized Commercial Bank (NCB), which is not a database at all. They collect data from different branches and arrange it in Excel files for reporting purpose. An old Private Commercial Bank (PCB), Bank-3 (B3), has been using MS-Access for their banking operations. Another Nationalized Commercial Bank, Bank-4 (B4), stores their data in MS-Visual FoxPro database. Bank-5 (B5) is a Specialized Bank (SB) that is using COBOL based software that stores banking information into text files. Finally, MS-SQL Server is used by a Foreign Commercial Bank (FCB), Bank-6 (B6).

First of all, we have setup network communication links to database servers of all banks. Communication may be direct or through third party, according to the wish of BB, considering financial budget and security. We assumed that data are coming through ISP, VPN, PSTN or optical fiber connectivity. Figure 2.1 describes the network communication from delivery channels to linked server of BB.

In the next step, normalization issues are considered and Oracle database is designed for the production server of the central bank (one for temporary use and other for permanent use). Relationship among entities, table structures, constraints, data types and its size are also carefully handled. Dummy data are used to test the output of the system. SQL commands for designing the databases are used. Middle-tier (linked-server) database is designed followed by the details

designing of commercial banks' database in different platforms. To retrieve data from central database servers of different banks we have written standard procedures by using T-SQL (Transact-Structured Query Language) and PL/SQL (Procedural Language/ Structured Query Language). There is a similarity among the databases and banking software of all banks as they maintain the same hierarchical relationship to manage the system. Language of the banking software or database may be different. So it becomes very easy for us to capture the data from different banking software almost in the same fashion. So, common procedures and functions are used with slight modification. Moreover, SQL is a unique common language that is accepted by all database system. As a result, we have easily communicated with all database servers of different banks to gather data.

After establishing the network connection successfully, we established database links to all database servers from the middle-tiered linked-server. In this project, MS-SQL Server is used as a linked-server because of cost, flexibility, availability and easy manageability. Microsoft has a very powerful tool, named MS-OLEDB that can be used to access, import and export data among databases easily. Except MS-Visual FoxPro, OLEDB is used for Oracle, SQL-Server, MS-Access, MS-Excel and even for Text files. By using OLEDB we directly accessed the mentioned databases without any middle interference. But, in case of Visual FoxPro, we used MS-OLEDB Provider for ODBC. ODBC is another tool, though not as secured as OLEDB, is used to access FoxPro database. Unfortunately, Microsoft did not provide any facility to OLEDB to access Visual FoxPro database directly as it is not a pure database engine and store data as a record concept. As a result, we configured ODBC for Visual FoxPro and then used MS-OLEDB Provider for ODBC to get data. Moreover, distributed query architecture, written procedures to gather, filter and process data, functionality of necessary procedures, data providers, communication and security features are also tested with due attention. Precaution regarding

smooth data gathering, in case of communication disruption, are also taken care of properly. Figure 2.2 describes the above mentioned technology.

After collecting and processing data in the linked-server (middle-tier), we exported it to the central database of BB (production server) that is connected to the linked-server via optical link. For this purpose we used MS-Data Transformation Services (MS-DTS). At first, data is received in a temporary location of the production server from linked-server and after final verification to ensure that necessary data reached successfully, data is transferred to the permanent location for official use.

After successful reception of data into the production server of the central bank from different commercial banks, GUIs are designed that opens the ways of the use of the final processed data by the ultimate end-users. We developed a set of forms and reports so that users can interact with data and take necessary hard copy print according to their needs. The end-users will also be able to pass parameters to filter the data according to the demand. Necessary procedures and functions are written according the need of the forms and reports. We configured report and form tools step-by-step as required. Tools are customized and configured efficiently so that the end-users can use it interactively. Oracle Developer 6i is used in this case.

The most important part of the project, data security is handled very carefully. Domain configuration, creation of appropriate views and user level security management are also strictly monitored. Mechanism regarding database object use, granting and revoking roles and privileges are also implemented. Proper folder permissions, for Sharing non-database files like MS-Excel and ASCII text files, are also given.

Finally, the simulated outputs are verified and we found that the proposed system developed in the lab is working properly. Ultimate output is seen by running reports before printing.

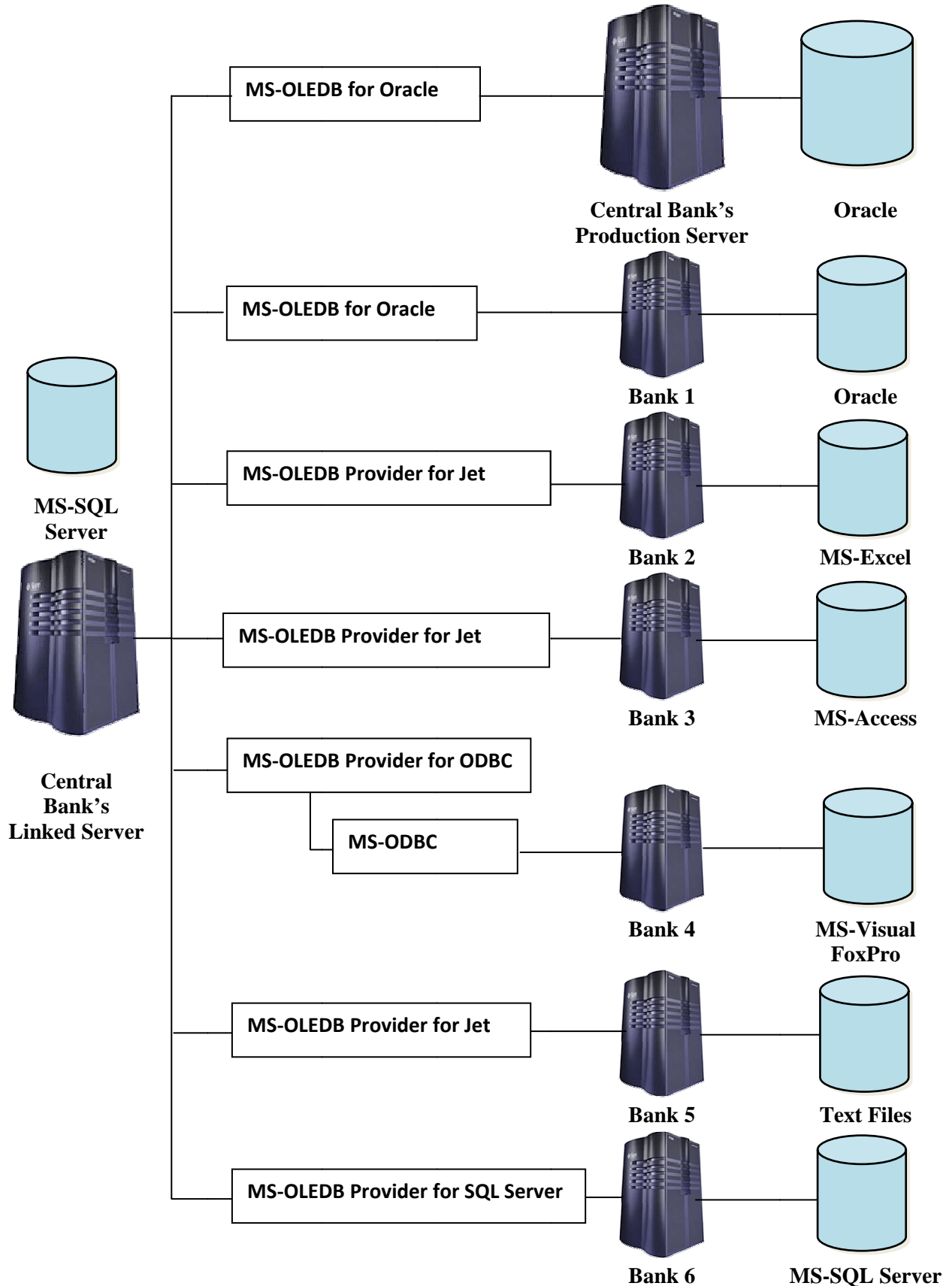


Fig. 2.2: Architecture of the Project (Data Communication)

Chapter 3

Database Design for Central Bank, Linked-Server and Commercial Banks

3.1 Introduction

There are three parts of the database design issues. First part is the designing of an oracle database for the production server of the Central Bank. In this database, we will create two schemas, one for the temporary storage of the received data from the linked-server and other one is the final destination for several uses. In the temporary schema, we will finally check the data with filtering and if we are sure that data from all banks correctly reached in the production server, we will transfer it to the permanent storage area and delete from temporary schema. A set of procedures written in PL/SQL language will be run for this purpose.

In second part, we will design a Linked-Server database in MS-SQL Server. This is the most important database in this project, because data will be gathered, filtered and processed here by linking all commercial banks' database with it.

In third part of this chapter, we will design all commercial banks' database. Here we will use Oracle, MS-SQL Server, MS-Access, MS-Visual FoxPro, MS-Excel and Text files. These databases will supply necessary information to the linked-server according to the demand of the Central Bank. So these databases are the primary data source of the project.

3.2 Normalization Issues

In database design, normalization is used to reduce data redundancy and to improve performance. Normalization also prevents update anomalies and data inconsistencies.

There are three basic steps of normalization.

- a) First Normal Form
- b) Second Normal Form
- c) Third Normal Form

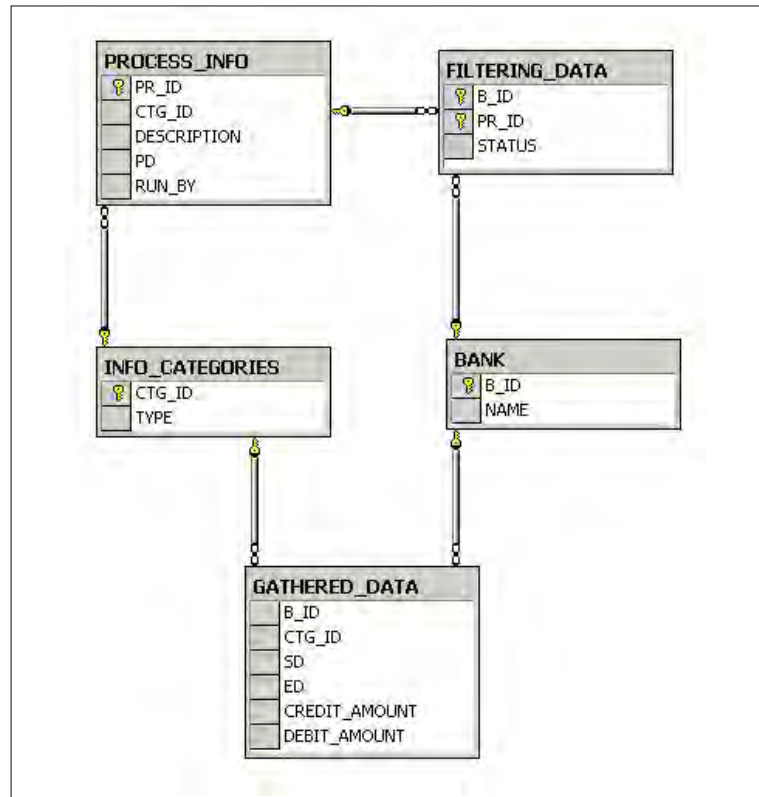
Normalization rules are applied while conducting Entity-Relationship Modeling. The normalization process results the data structure of the relational tables. First normal form is concerned with the shape of the record and the preliminary identification of keys. Second normal form and third normal form are concerned with resolving problems associated with the interrelationship of keys and dependent attributes.

In proposed system, there are eight databases; six for commercial banks, one for linked-server (middle-tier) and last one for the central bank. All databases are designed applying first, second and third normal form. The results of the normalization are relational tables that are used in all databases. The data structures of those tables are also outcome of normalization.

In case of designing the database, normalization procedures are fully maintained. Entity Relationship and relational model are also taken into account. As Oracle is an Object Oriented Relational Database Management System, relation is properly developed among entities by using primary and foreign key concept. We have also adopted different types of database constraints (not null, check, unique, etc.) where necessary.

The relational model is the primary data model for commercial data processing applications. It has attained its primary position because of its simplicity. The relational model uses a collection of tables to represent both data and relationship among those data. Each table has multiple columns, and each column has a unique name. Figure 3.1 shows the detail design and relationship of the tables in the database. Different tables with characteristics are also given below (Table: 3.1-3.5).

3.3 Oracle Database Design for the Production Server of the Central Bank



**Fig. 3.1: Temporary Schema of Central Bank's Server
(To Store Data Received from Linked-Server)**

3.3.1 Entity and Attributes Exploration

There are a number of entities involved in this part of the proposed system. Each entity has its own attributes (data elements) to describe itself. The entity and its attributes are described in the following sub sections.

3.3.1.1 Info_Categories

Different types of information will be gathered in this project. For example, tax, loan, export, import, liquid asset, etc. This entity shall categories these data for further use of the relational purpose. Table description and sample data are given below for better understanding.

Table 3.1: Info_Categories

Column Name	Description	Data Type	Size	Constraint	Reference
CTG_ID	Category Identity Number	CHAR	5	PRIMARY KEY	
TYPE	Category Name	VARCHAR	20	NOT NULL	

Sample Data

CTG_ID	TYPE
CTG01	TAX
CTG02	VAT
CTG03	IMPORT LC
CTG04	EXPORT LC
CTG05	LOAN
CTG06	LIQUID ASSET
CTG07	FDR
CTG08	CHARGES
CTG09	ADVANCES
CTG10	DEFAULTS
CTG11	INTEREST INCOME

3.3.1.2 Bank

This entity shall contain a list of banks with their identity number. Table description and sample data are given below.

Table 3.2: Bank

Column Name	Description	Data Type	Size	Constraint	Reference
B_ID	Bank's Identity Number	CHAR	2	PRIMARY KEY	
NAME	Bank' Name	VARCHAR	50	NOT NULL	

Sample Data

B_ID	NAME
B1	SONALI BANK LIMITED
B2	BANGLADESH KRISHI BANK
B3	DUTCH-BANGLA BANK LIMITED
B4	HSBC LIMITED
B5	RUPALI BANK LIMITED
B6	BASIC BANK LIMITED

3.3.1.3 Gathered_Data

Data transferred from linked-server will be received by the system and stored in this entity first. After checking whether all data are reached well or not, we will transfer this data to the permanent schema. If we found that all data are not received properly, we will request to the

linked-server to send the data again. This entity will store ultimate data. Table description and sample data are given below.

Table 3.3: Gathered_Data

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
TR_ID	Transaction Identity Number	CHAR	6	PRIMARY KEY	
B_ID	Bank's Identity Number	CHAR	2	FOREIGN KEY	Table: Bank Column: B_ID
CTG_ID	Category Identity Number	CHAR	5	FOREIGN KEY	Table: Process_Info Column: PR_ID
PR_ID	Process Identity Number	CHAR	5		
SD	Starting Date	DATE		NOT NULL	
ED	Ending Date	DATE		NOT NULL	
CD	Collection Date	DATE		NOT NULL	
CREDIT_AMOUNT	Credit Balance	NUMBER	15,2		
DEBIT_AMOUNT	Debit Balance	NUMBER	15,2		

Sample Data

TR_ID	B_ID	CTG_ID	PR_ID	SD	ED	CD	CREDIT_AMOUNT	DEBIT_AMOUNT
TR0148	B4	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12		33000000
TR0149	B5	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12	33000000	
TR0150	B6	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12		2532543
TR0151	B1	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12	251425	
TR0152	B2	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12	123456	
TR0153	B3	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12		80000000
TR0140	B2	CTG01	PR001	01-JAN-11	01-JAN-14	19-FEB-12		90634.9
TR0141	B3	CTG02	PR002	01-JAN-11	01-JAN-14	19-FEB-12	1812698	
TR0154	B4	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12	25362541	
TR0155	B5	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12		80000000
TR0156	B6	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12		80000000

3.3.1.4 Process_Info

To ensure that required data of all banks successfully reached into the temporary schema of production server from linked-server, we have to run a process. This process will check and summarize data by using batch processing technique. So it is necessary to know what categories of data handled by the process with a brief description, when the process is run and who run this. This entity will actually store data to monitor the processes run by the users. Table description and sample data are given below.

Table 3.4: Process_Info

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
PR_ID	Process Identity Number	CHAR	5	PRIMARY KEY	
CTG_ID	Category Identity Number	CHAR	5	FOREIGN KEY	Table: INFO_CATEGORIES Column: CTG_ID
DESCRIPTION	Details about the Process	VARCHAR	50	NOT NULL	
PD	Processing Date	DATE		NOT NULL	
RUN_BY	User of the Process	VARCHAR	20	NOT NULL	

Sample Data

PR_ID	CTG_ID	DESCRIPTION	PD	RUN_BY
PR001	CTG01	TAX Collection	19-FEB-12	RAIHAN
PR002	CTG02	VAT Collected	19-FEB-12	BEENA
PR003	CTG03	IMPORT LC	19-FEB-12	KAMAL
PR004	CTG04	EXPORT LC	19-FEB-12	TAPAN
PR005	CTG08	CHARGES	19-FEB-12	ALAM
PR006	CTG06	VAULT DATA	19-FEB-12	ALAM

3.3.1.5 Filtering_Data

To ensure that required data of all banks successfully reached into the temporary schema of the production server from linked-server, we have to run a process. This process will put a flag in the 'status' field of this table. Here 'Y' indicates data reached successfully and 'N' indicates failure. By observing this flags we will transfer these data to permanent schema of the production server. If we see that data for some banks are missing, in that case we will send request to the linked-server to transfer data from linked-server to the temporary schema of production server again. Table description and sample data are given below.

Table 3.5: Filtering_Data

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
PR_ID	Process Identity Number	CHAR	5	JOINTLY PRIMARY KEY	Table: Process_INFO Column: PR_ID
B_ID	Bank's Identity Number	CHAR	2		Table: BANK Column: B_ID
STATUS	Status of the Process whether executed or not	CHAR	5		

Sample Data

B_ID	PR_ID	STATUS
B1	PR002	Y
B2	PR002	Y
B3	PR002	Y
B4	PR002	N
B5	PR002	Y
B1	PR001	Y
B2	PR001	Y
B3	PR001	N
B4	PR001	Y
B5	PR001	Y
B6	PR001	N

3.3.2 Creating Tables of the Temporary Schema

The following SQL commands are used to create the necessary tables under Temporary Schema of Oracle production Server. SQL- Insert commands used to populate data in these tables are given in Appendix-I (a).

```
DROP TABLE BANK;
```

```
CREATE TABLE BANK
```

```
(B_ID CHAR(2) PRIMARY KEY,
```

```
NAME VARCHAR(50) NOT NULL);
```

```
DROP TABLE INFO_CATEGORIES;
```

```
CREATE TABLE INFO_CATEGORIES
```

```
(CTG_ID CHAR(5) PRIMARY KEY,
```

```
TYPE VARCHAR(20) NOT NULL);
```

```
DROP TABLE PROCESS_INFO;
```

```
CREATE TABLE PROCESS_INFO
```

```
(PR_ID CHAR(5) PRIMARY KEY,
```

```
CTG_ID CHAR(5) REFERENCES INFO_CATEGORIES,
```

```
DESCRIPTION VARCHAR(50) NOT NULL,
```

```
PD DATE NOT NULL,
```

```
RUN_BY VARCHAR(20) NOT NULL);
```

```
DROP TABLE FILTERING_DATA;
```

```
CREATE TABLE FILTERING_DATA
```

```

(B_ID CHAR(2) REFERENCES BANK,
PR_ID CHAR(5) REFERENCES PROCESS_INFO,
STATUS CHAR(1),
PRIMARY KEY (B_ID, PR_ID));

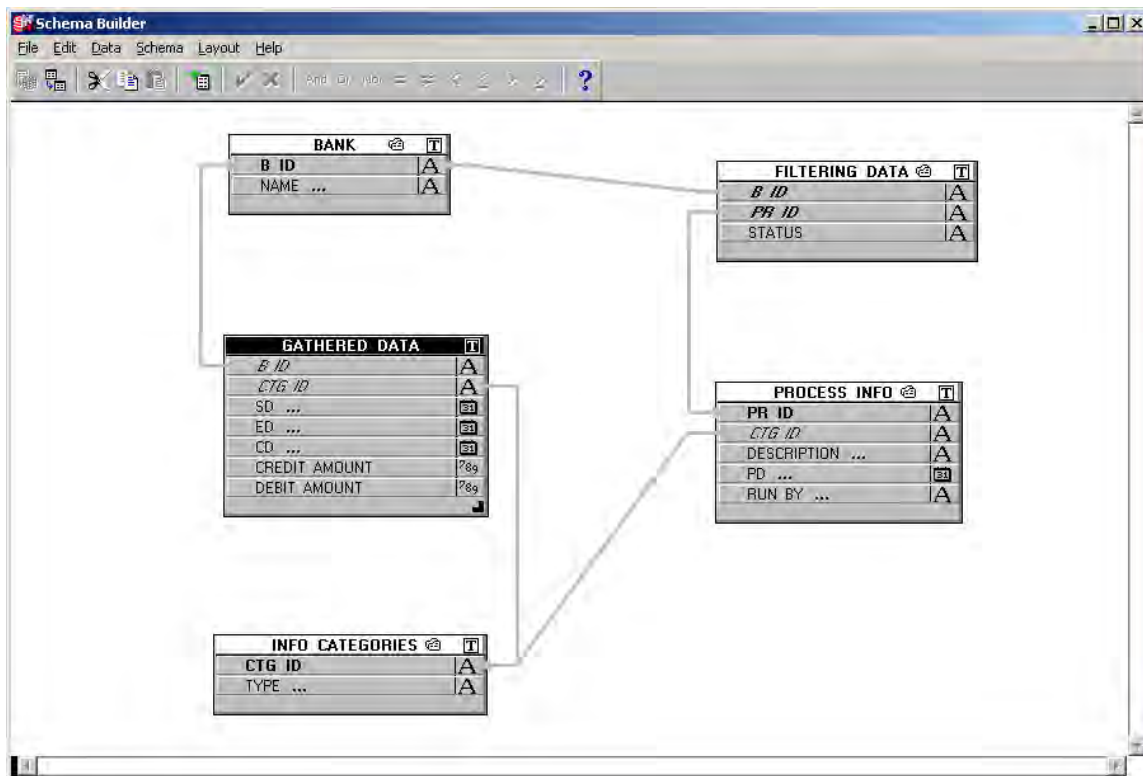
```

```

DROP TABLE GATHERED_DATA;
CREATE TABLE GATHERED_DATA
(B_ID CHAR(2) REFERENCES BANK,
CTG_ID CHAR(5) REFERENCES INFO_CATEGORIES,
SD DATE NOT NULL,
ED DATE NOT NULL,
CD DATE NOT NULL,
CREDIT_AMOUNT NUMERIC(15,2),
DEBIT_AMOUNT NUMERIC(15,2));

```

3.4 Permanent Schema of Production Server: To Store Data Received from Temporary Schema



**Fig. 3.2: Permanent Schema of Central Bank's Server
(To Store Data Received from Temporary Schema)**

3.4.1 Entity and Attributes Exploration

There are a number of entities also involved in this part of the proposed system. Each entity has its own attributes (data elements) to describe itself. The entity and its attributes are described in the following sub sections.

3.4.1.1 Info_Categories

Different types of information will be gathered in this project. For example, tax, loan, export, import, liquid asset, etc. This entity shall categories these data for further use of the relational purpose. Table description and sample data are given below.

Table 3.6: Info_Categories

Column Name	Description	Data Type	Size	Constraint	Reference
CTG_ID	Category Identity Number	CHAR	5	PRIMARY KEY	
TYPE	Category Name	VARCHAR	20	NOT NULL	

Sample Data

CTG_ID	TYPE
CTG01	TAX
CTG02	VAT
CTG03	IMPORT LC
CTG04	EXPORT LC
CTG05	LOAN
CTG06	LIQUID ASSET
CTG07	FDR
CTG08	CHARGES
CTG09	ADVANCES
CTG10	DEFAULTS
CTG11	INTEREST INCOME

3.4.1.2 Bank

This entity shall contain a list of banks with their identity number. Table description and sample data are given below.

Table 3.7: Bank

Column Name	Description	Data Type	Size	Constraint	Reference
B_ID	Bank's Identity Number	CHAR	2	PRIMARY KEY	
NAME	Bank' Name	VARCHAR	50	NOT NULL	

Sample Data

B_ID	NAME
B1	SONALI BANK LIMITED
B2	BANGLADESH KRISHI BANK
B3	DUTCH-BANGLA BANK LIMITED
B4	HSBC LIMITED
B5	RUPALI BANK LIMITED
B6	BASIC BANK LIMITED

3.4.1.3 Process_Info

To ensure that required data of all banks successfully reached into the permanent schema of the production server, we have to run a process. This process will check and summarize data by using batch processing technique. So it is necessary to know what categories of data handled by the process with a brief description, when the process is run and who run this. This entity will actually store data to monitor the processes run by the users. Table description and sample data are given below.

Table 3.8: Process_Info

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
PR_ID	Process Identity Number	CHAR	5	PRIMARY KEY	
CTG_ID	Category Identity Number	CHAR	5	FOREIGN KEY	Table: INFO_CATEGORIES Column: CTG_ID
DESCRIPTION	Details about the Process	VARCHAR	50	NOT NULL	
PD	Processing Date	DATE		NOT NULL	
RUN_BY	User of the Process	VARCHAR	20	NOT NULL	

Sample Data

PR_ID	CTG_ID	DESCRIPTION	PD	RUN_BY
PR001	CTG01	TAX Collection	19-FEB-12	RAIHAN
PR002	CTG02	VAT Collected	19-FEB-12	BEENA
PR003	CTG03	IMPORT LC	19-FEB-12	KAMAL
PR004	CTG04	EXPORT LC	19-FEB-12	TAPAN
PR005	CTG08	CHARGES	19-FEB-12	ALAM
PR006	CTG06	VAULT DATA	19-FEB-12	ALAM

3.4.1.4 Filtering_Data

To ensure that required data of all banks successfully reached into the permanent schema from the temporary schema of the production server, we have to run a process. This process will put a flag in the 'status' field of this table. Here 'Y' indicates data reached successfully and 'N' indicates failure. By observing this values we will go for final report. If we see that data for some banks are missing, in that case we will run the procedure again. If all data reaches well, we will

delete it from temporary schema permanently. Table description and sample data are given below.

Sample Data

B_ID	PR_ID	STATUS
B1	PR002	Y
B2	PR002	Y
B3	PR002	Y
B4	PR002	N
B5	PR002	Y
B1	PR001	Y
B2	PR001	Y
B3	PR001	N
B4	PR001	Y
B5	PR001	Y
B6	PR001	N

Table 3.9: Filtering_Data

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
PR_ID	Process Identity Number	CHAR	5	JOINTLY PRIMARY KEY	Table: Process_INFO Column: PR_ID
B_ID	Bank's Identity Number	CHAR	2		Table: BANK Column: B_ID
STATUS	Status of the Process whether executed or not	CHAR	5		

3.4.1.5 Gathered_Data

Finally checked and processed data will be stored in this entity. Users will get final reports from this table. This entity will store ultimate data as follows. Table description and sample data are given below.

Sample Data

TR_ID	B_ID	CTG_ID	PR_ID	SD	ED	CD	CREDIT_AMOUNT	DEBIT_AMOUNT
TR0148	B4	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12		33000000
TR0149	B5	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12	33000000	
TR0150	B6	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12		2532543
TR0151	B1	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12	251425	
TR0152	B2	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12	123456	
TR0153	B3	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12		80000000
TR0140	B2	CTG01	PR001	01-JAN-11	01-JAN-14	19-FEB-12		90634.9
TR0141	B3	CTG02	PR002	01-JAN-11	01-JAN-14	19-FEB-12	1812698	
TR0154	B4	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12	25362541	
TR0155	B5	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12		80000000
TR0156	B6	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12		80000000

Table 3.10: Gathered_Data

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
TR_ID	Transaction Identity Number	CHAR	6	PRIMARY KEY	
B_ID	Bank's Identity Number	CHAR	2	FOREIGN KEY	Table: Bank Column: B_ID
CTG_ID	Category Identity Number	CHAR	5	FOREIGN KEY	Table: Process_Info Column: PR_ID
PR_ID	Process Identity Number	CHAR	5		
SD	Starting Date	DATE		NOT NULL	
ED	Ending Date	DATE		NOT NULL	
CD	Collection Date	DATE		NOT NULL	
CREDIT_AMOUNT	Credit Balance	NUMBER	15,2		
DEBIT_AMOUNT	Debit Balance	NUMBER	15,2		

3.4.2 Creating Tables: Permanent Schema of Central Bank

The following SQL commands are used to create the necessary tables under Permanent Schema of Oracle production Server. SQL- Insert commands used to populate data in these tables are given in Appendix-I (b).

```
DROP TABLE BANK;
```

```
CREATE TABLE BANK
```

```
(B_ID CHAR(2) PRIMARY KEY,
```

```
NAME VARCHAR(50) NOT NULL);
```

```
DROP TABLE INFO_CATEGORIES;
```

```
CREATE TABLE INFO_CATEGORIES
```

```
(CTG_ID CHAR(5) PRIMARY KEY,
```

```
TYPE VARCHAR(20) NOT NULL);
```

```
DROP TABLE PROCESS_INFO;
```

```
CREATE TABLE PROCESS_INFO
```

```
(PR_ID CHAR(5) PRIMARY KEY,
```

```
CTG_ID CHAR(5) REFERENCES INFO_CATEGORIES,
```

```
DESCRIPTION VARCHAR(50) NOT NULL,
```

```
PD DATE NOT NULL,
```

```
RUN_BY VARCHAR(20) NOT NULL);
```

```

DROP TABLE FILTERING_DATA;

CREATE TABLE FILTERING_DATA
(B_ID CHAR(2) REFERENCES BANK,
PR_ID CHAR(5) REFERENCES PROCESS_INFO,
STATUS CHAR(1),
PRIMARY KEY (B_ID, PR_ID));

```

```

DROP TABLE GATHERED_DATA;

CREATE TABLE GATHERED_DATA
(B_ID CHAR(2) REFERENCES BANK,
CTG_ID CHAR(5) REFERENCES INFO_CATEGORIES,
SD DATE NOT NULL,
ED DATE NOT NULL,
CD DATE NOT NULL,
CREDIT_AMOUNT NUMERIC(15,2),
DEBIT_AMOUNT NUMERIC(15,2));

```

3.5 Linked-Server Database (Middle-Tier)

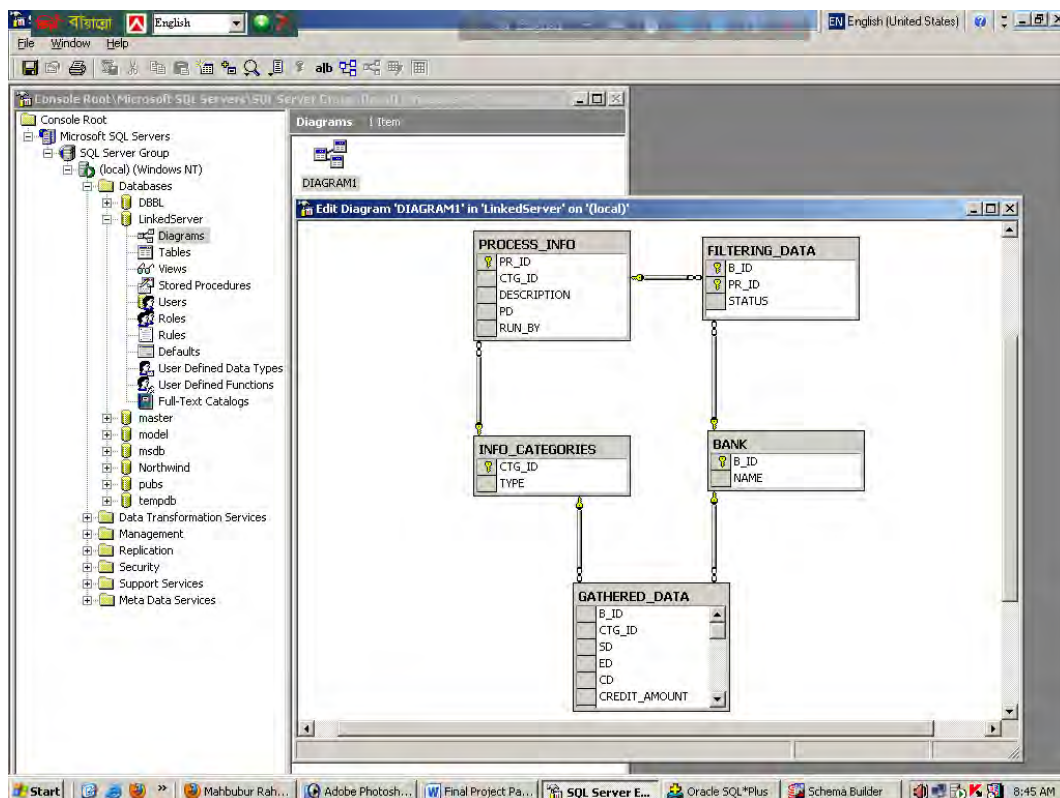


Fig. 3.3: Linked-Server Database Design

3.5.1 Entity and Attributes Exploration

There are a number of entities also involved in the most important part of the proposed system. Each entity has its own attributes (data elements) to describe itself. The entity and its attributes are described in the following sub sections.

3.5.1.1 Info_Categories

Different types of information will be gathered in this project. For example, tax, loan, export, import, liquid asset, etc. This entity shall categories these data for further use of the relational purpose. Table description and sample data are given below.

Table 3.11: Info_Categories

Column Name	Description	Data Type	Size	Constraint	Reference
CTG_ID	Category Identity Number	CHAR	5	PRIMARY KEY	
TYPE	Category Name	VARCHAR	20	NOT NULL	

Sample Data

CTG_ID	TYPE
CTG01	TAX
CTG02	VAT
CTG03	IMPORT LC
CTG04	EXPORT LC
CTG05	LOAN
CTG06	LIQUID ASSET
CTG07	FDR
CTG08	CHARGES
CTG09	ADVANCES
CTG10	DEFAULTS
CTG11	INTEREST INCOME

3.5.1.2 Bank

This entity shall contain a list of banks with their identity number. Linked-server will communicate with banks' database servers with this identity number. Table description and sample data are given below.

Table 3.12: Bank

Column Name	Description	Data Type	Size	Constraint	Reference
B_ID	Bank's Identity Number	CHAR	2	PRIMARY KEY	
NAME	Bank' Name	VARCHAR	50	NOT NULL	

Sample Data

B_ID	NAME
B1	SONALI BANK LIMITED
B2	BANGLADESH KRISHI BANK
B3	DUTCH-BANGLA BANK LIMITED
B4	HSBC LIMITED
B5	RUPALI BANK LIMITED
B6	BASIC BANK LIMITED

3.5.1.3 Gathered_Data

After successfully linked all servers of different banks, different procedure will be run that will gather summarized data according to the demand of the linked-server in this entity. Actually, this entity will be used to gather data in a central location. This data will be checked and filtered by other procedures according to the demand. If accurate and sufficient information is gathered successfully, this data will be sent to the temporary schema of the central bank's production server. Table description and sample data are given below.

Table 3.13: Gathered_Data

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
TR_ID	Transaction Identity Number	CHAR	6	PRIMARY KEY	
B_ID	Bank's Identity Number	CHAR	2	FOREIGN KEY	Table: Bank Column: B_ID
CTG_ID	Category Identity Number	CHAR	5	FOREIGN KEY	Table: Process_Info Column: PR_ID
PR_ID	Process Identity Number	CHAR	5		
SD	Starting Date	DATETIME		NOT NULL	
ED	Ending Date	DATETIME		NOT NULL	
CD	Collection Date	DATETIME		NOT NULL	
CREDIT_AMOUNT	Credit Balance	NUMERIC	15,2		
DEBIT_AMOUNT	Debit Balance	NUMERIC	15,2		

Sample Data

TR_ID	B_ID	CTG_ID	PR_ID	SD	ED	CD	CREDIT_AMOUNT	DEBIT_AMOUNT
TR0148	B4	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12		33000000
TR0149	B5	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12	33000000	
TR0150	B6	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12		2532543
TR0151	B1	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12	251425	
TR0152	B2	CTG03	PR003	01-JAN-11	01-JAN-14	19-FEB-12	123456	
TR0153	B3	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12		80000000
TR0140	B2	CTG01	PR001	01-JAN-11	01-JAN-14	19-FEB-12		90634.9
TR0141	B3	CTG02	PR002	01-JAN-11	01-JAN-14	19-FEB-12	1812698	
TR0154	B4	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12	25362541	
TR0155	B5	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12		80000000
TR0156	B6	CTG04	PR004	01-JAN-11	01-JAN-14	19-FEB-12		80000000

3.5.1.4 Process_Info

To ensure that required data of all banks successfully reached into the linked-server, we have to run a process. This process will check and summarize data by using batch processing technique. So it is necessary to know what categories of data handled by the process with a brief description, when the process is run and who run this. This entity will actually store data to monitor the processes run by the users. Table description and sample data are given below.

Table 3.14: Process_Info

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
PR_ID	Process Identity Number	CHAR	5	PRIMARY KEY	
CTG_ID	Category Identity Number	CHAR	5	FOREIGN KEY	Table: INFO_CATEGORIES Column: CTG_ID
DESCRIPTION	Details about the Process	VARCHAR	50	NOT NULL	
PD	Processing Date	DATETIME		NOT NULL	
RUN_BY	User of the Process	VARCHAR	20	NOT NULL	

Sample Data

PR_ID	CTG_ID	DESCRIPTION	PD	RUN_BY
PR001	CTG01	TAX Collection	19-FEB-12	RAIHAN
PR002	CTG02	VAT Collected	19-FEB-12	BEENA
PR003	CTG03	IMPORT LC	19-FEB-12	KAMAL
PR004	CTG04	EXPORT LC	19-FEB-12	TAPAN
PR005	CTG08	CHARGES	19-FEB-12	ALAM
PR006	CTG06	VAULT DATA	19-FEB-12	ALAM

3.5.1.5 Filtering_Data

To ensure that required data of all banks successfully reached into the linked-server from all servers of different banks, we have to run the above mentioned process. This process will put a flag in the 'status' field of this table. Here 'Y' indicates data reached successfully and 'N' indicates failure. By observing this values we will ensure whether data of all banks are gathered successfully or not. If we see that data for some banks are missing, in that case we will run the procedure again. If all data reaches well, we will export it to the temporary schema of central bank's production server. Table description and sample data are given below.

Table 3.15: Filtering_Data

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
PR_ID	Process Identity Number	CHAR	5	JOINTLY PRIMARY KEY	Table: Process_INFO Column: PR_ID
B_ID	Bank's Identity Number	CHAR	2		Table: BANK Column: B_ID
STATUS	Status of the Process whether executed or not	CHAR	5		

Sample Data

B_ID	PR_ID	STATUS
B1	PR002	Y
B2	PR002	Y
B3	PR002	Y
B4	PR002	N
B5	PR002	Y
B1	PR001	Y
B2	PR001	Y
B3	PR001	N
B4	PR001	Y
B5	PR001	Y
B6	PR001	N

3.5.2 Creation of Linked-Server Database and Tables in MS-SQL Server (Middle-Tier)**3.5.2.1 Creation of Database**

```
USE MASTER
```

```
GO
```

```
CREATE DATABASE LINKEDSERVER
```

```
ON
```

```
( NAME = 'LINKEDSERVER_DAT',
```

```
  FILENAME = 'I:\PROGRAM FILES\MICROSOFT SQL SERVER\MSSQL\DATA\LINKEDSERVER_DAT.MDF',
```

```
  SIZE = 10,
```

```
  MAXSIZE = 50,
```

```
  FILEGROWTH = 5 )
```

```
LOG ON
```

```
( NAME = 'LINKEDSERVER',
```

```
  FILENAME = 'I:\PROGRAM FILES\MICROSOFT SQL SERVER\MSSQL\DATA\LINKEDSERVER_LOG.LDF',
```

```
  SIZE = 5MB,
```

```
  MAXSIZE = 25MB,
```

```
  FILEGROWTH = 5MB )
```

```
GO
```

3.5.2.2 Creation of Tables

The following SQL commands are used to create the necessary tables in the Linked-Server which is a MS-SQL Server. SQL- Insert commands used to populate data in these tables are given in Appendix-I (c).

```
DROP TABLE BANK;
```

```
CREATE TABLE BANK
```

```
(B_ID CHAR(2) PRIMARY KEY,
```

```
NAME VARCHAR(50) NOT NULL);
```

```
DROP TABLE INFO_CATEGORIES;
```

```
CREATE TABLE INFO_CATEGORIES
```

```
(CTG_ID CHAR(5) PRIMARY KEY,
```

```
TYPE VARCHAR(20) NOT NULL);
```

```
DROP TABLE PROCESS_INFO;
```

```
CREATE TABLE PROCESS_INFO
```

```
(PR_ID CHAR(5) PRIMARY KEY,
```

```
CTG_ID CHAR(5) REFERENCES INFO_CATEGORIES,
```

```
DESCRIPTION VARCHAR(50) NOT NULL,
```

```
PD DATETIME NOT NULL,
```

```
RUN_BY VARCHAR(20) NOT NULL);
```

```
DROP TABLE FILTERING_DATA;
```

```
CREATE TABLE FILTERING_DATA
```

```
(B_ID CHAR(2) REFERENCES BANK,
```

```
PR_ID CHAR(5) REFERENCES PROCESS_INFO,
```

```
STATUS CHAR(1),
```

```
PRIMARY KEY (B_ID, PR_ID));
```

```
DROP TABLE GATHERED_DATA;
```

```
CREATE TABLE GATHERED_DATA
```

```
(B_ID CHAR(2) REFERENCES BANK,
```

```
CTG_ID CHAR(5) REFERENCES INFO_CATEGORIES,
```

```
SD DATETIME NOT NULL,
```

ED DATETIME NOT NULL,

CD DATETIME NOT NULL,

CREDIT_AMOUNT NUMERIC(15,2),

DEBIT_AMOUNT NUMERIC(15,2));

3.6 Commercial Banks' Database

This is the third part of the database design of this project. In this part, we will design all commercial banks' database. Here we will use Oracle, MS-SQL Server, MS-Access, MS-Visual FoxPro, MS-Excel and Text files. These databases and data files will supply necessary information to the linked-server according to the demand of the Central Bank. So these databases are the primary data source of the project.

3.6.1 Commercial Banks' Database: Bank-6 Using MS-SQL Server

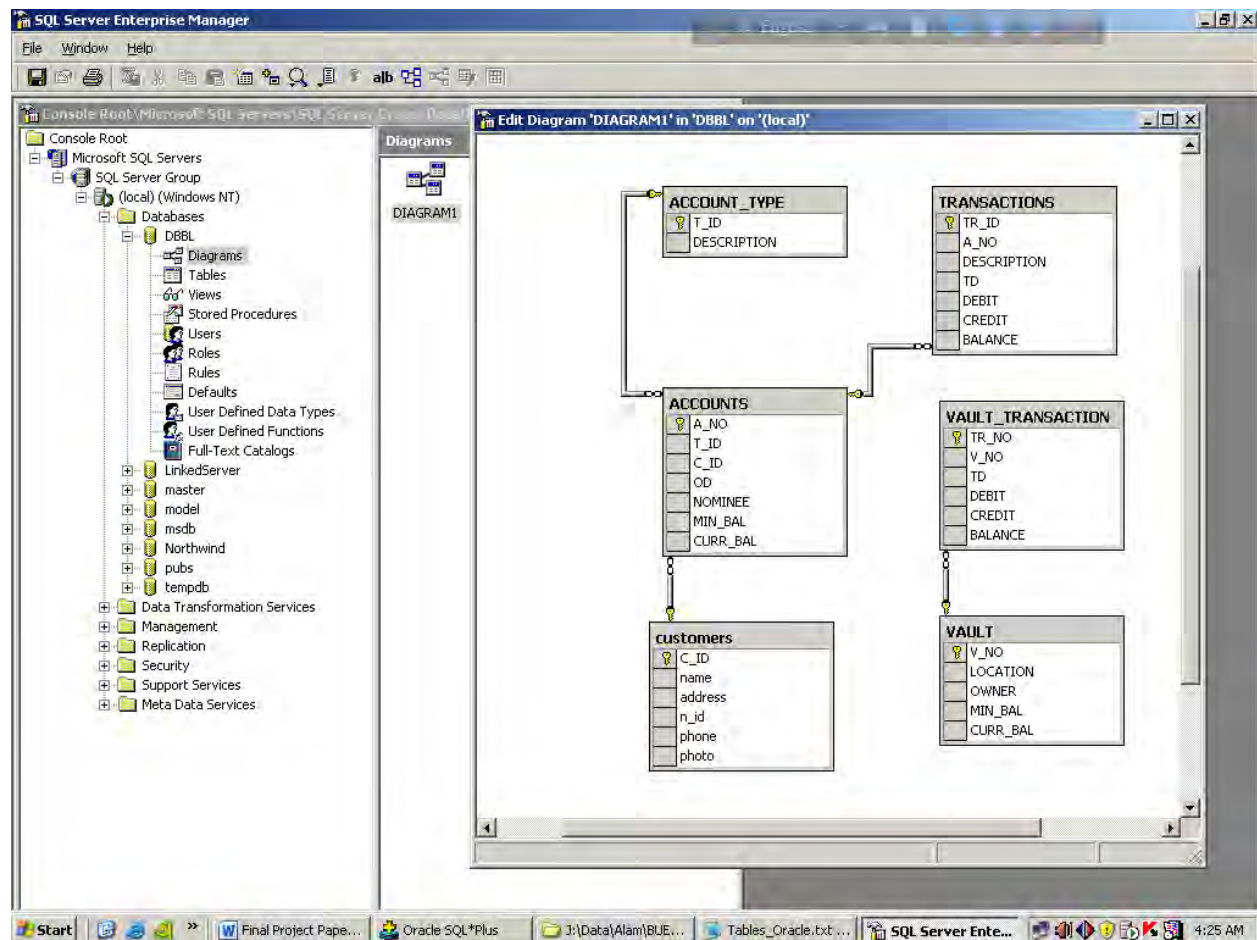


Fig. 3.4: SQL Server Database Design for Bank-6

3.6.1.1 Entity and Attributes Exploration

There are a number of entities also involved in the most important part of the proposed system. Each entity has its own attributes (data elements) to describe itself. The entity and its attributes are described in the following sub sections.

3.6.1.1.1 Customers

Different types of information regarding customers' will be gathered in this entity. Data of this entity will be used for relational purpose. Table description and sample data are given below.

Table 3.16: Customers

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
C_ID	CUSTOMER'S IDENTITY NUMBER	CHAR	4	PRIMARY KEY	
NAME	NAME OF THE CUSTOMER	VARCHAR	30	NOT NULL	
ADDRESS	PRESENT ADDRESS OF THE CUSTOMER	VARCHAR	30	FOREIGN KEY	
N_ID	NATIONAL IDENTITY NUMBER	CHAR	13	NOT NULL	
PHONE	PHONE NUMBER	VARCHAR	30		
PHOTO	PHOTOGRAPH OF THE CLIENT	LONG RAW			

Sample Data

C_ID	NAME	ADDRESS	N_ID	PHONE	PHOTO
C001	MD. HAFIZUR RAHMAN	413, MIRPUR, DHAKA	1235695823254	01775858256	
C002	MS. KANIZ RBBI	744, KAZIPARA, MIRPUR, DHAKA	4569871425632	9003088, 0174586958	
C003	MD. HELEL UDDIN	H-3, R-5, DHANMONDI, DHAKA	1885895823254	01556323244	
C004	MS. TAHMINA AKHTER	H#12, R#15, SHYAMOLI, DHAKA	2365251425632	9003031	

3.6.1.1.2 Account_Type

Different types of accounts are maintained by banks in Bangladesh. This entity will contain name of different types of accounts that will be used for relational purpose. Table description and sample data are given below.

Table 3.17: Account_Type

Column Name	Description	Data Type	Size	Constraint	Reference
T_ID	Account Type Identity Number	CHAR	2	PRIMARY KEY	
DESCRIPTION	Description of the Account Type	VARCHAR	20	NOT NULL	

Sample Data

T_ID	DESCRIPTION
T1	SAVINGS
T2	CURRENT
T3	FDR
T4	LOAN
T5	IMPORT LC
T6	EXPORT LC

3.6.1.1.3 Accounts

This entity shall contain a list of all accounts opened by the bank. Table description and sample data are given below.

Table 3.18: Accounts

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
A_NO	Account Number	CHAR	4	PRIMARY KEY	
T_ID	Account Type Identity Number	CHAR	2	FOREIGN KEY	Table: ACCOUNT_TYPE Column: T_ID
C_ID	Customer Identity Number	CHAR	4	FOREIGN KEY	Table: CUSTOMERS Column: C_ID
NOMINEE	Name of the Nominee	VARCHAR	30	NOT NULL	
OD	Account Opening Date	DATETIME		NOT NULL	
MIN_BAL	Minimum Balance of the Account	NUMERIC	10,2	CHECK(MIN_BAL>=0)	
CURR_BAL	Current Balance of the Account	NUMERIC	10,2	CHECK(CURR_BAL>=0)	
				UNIQUE(T_ID,C_ID)	

Sample Data

A_NO	T_ID	C_ID	OD	NOMINEE	MIN_BAL	CURR_BAL
A001	T1	C001	03-APR-11	SALAMA AKHTER: SISTER	500	6673
A002	T1	C002	08-APR-11	RABBANI MAHBUB: SON	500	2618.1
A003	T1	C003	13-APR-11	MOKBUL HOSSAIN: FATHER	5000	49200

3.6.1.1.4 Transactions

This is the most important entity of banking software. All transactional data are stored in this entity. We will pick data from this table mainly for the project. Table description and sample data are given below.

Table 3.19: Transactions

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
TR_ID	TRANSACTION IDENTITY NUMBER	CHAR	10	PRIMARY KEY	
A_NO	ACCOUNT NUMBER	CHAR	2	FOREIGN KEY	TABLE: ACCOUNTS COLUMN: A_NO
DESCRIPTION	DESCRIPTION OF THE ACCOUNT	VARCHAR	40		
TD	TRANSACTION DATE	DATETIME		NOT NULL	
DEBIT	DEBIT AMOUNT	NUMERIC	10,2	CHECK(DEBIT>=0)	
CREDIT	CREDIT AMOUNT	NUMERIC	10,2	CHECK(CREDIT>=0)	
BALANCE	BALANCE OF THE ACCOUNT	NUMERIC	10,2	CHECK(BALANCE>=0)	

Sample Data

TR_ID	A_NO	DESCRIPTION	TD	DEBIT	CREDIT	BALANCE
TR00000001	A001	CASH RECEIVE	13-APR-11		5000	5000
TR00000002	A001	CHEQUE	27-JUN-11	1000		400
TR00000003	A004	TRANSFER	01-AUG-11		5000	9000
TR00000004	A009	ATM	20-OCT-11	2000		7000

3.6.1.1.5 Vault

Liquid Assets management is an important issue to any central bank. The entity vault will supply the necessary information in this regard. Table description and sample data are given below.

Table 3.20: Vault

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
V_NO	VAULT NUMBER	CHAR	4	PRIMARY KEY	
LOCATION	VAULT LOCATION	VARCHAR	20	NOT NULL	
OWNER	VAULT MANAGER	VARCHAR	20	NOT NULL	
MIN_BAL	MINIMUM BALANCE	NUMERIC	10,2	CHECK(MIN_BAL>=0)	
CURR_BAL	CURRENT BALANCE	NUMERIC	10,2	CHECK(CURR_BAL>=0)	

V_NO	LOCATION	OWNER	MIN_BAL	CURR_BAL
V001	DHANMONDI BRANCH	MD. ZAMIL AHMED	25000000	30000000
V002	SUNAMGONJ BRANCH	MS. ANTARA ZERIN	20000000	40000000
V003	KHULNA BRANCH	MD. AKHTER HAMID	13000000	15000000

3.6.1.1.6 Vault_Transaction

The entity vault_transaction will hold the transaction information regarding cash-in and cash-out of a vault. Sample data and table structure are given below.

Table 3.21:Vault_Transaction

Column Name	Description	Data Type	Size	Constraint	Reference of Foreign Key
TR_NO	TRANSACTION IDENTITY NUMBER	CHAR	10	PRIMARY KEY	
VA_NO	VAULT NUMBER	CHAR	4	FOREIGN KEY	TABLE: VAULT COLUMN: V_NO
TD	TRANSACTION DATE	DATETIME		NOT NULL	
DEBIT	DEBIT AMOUNT	NUMERIC	10,2	CHECK(DEBIT>=0)	
CREDIT	CREDIT AMOUNT	NUMERIC	10,2	CHECK(CREDIT>=0)	
BALANCE	BALANCE OF THE VAULT	NUMERIC	10,2	CHECK(BALANCE>=0)	

Sample Data

TR_NO	V_NO	TD	DEBIT	CREDIT	BALANCE
VTR0000001	V001	27-JAN-12		50000000	50000000
VTR0000002	V001	28-JAN-12		30000000	80000000
VTR0000010	V002	27-JAN-12	1000000		7000000
VTR0000011	V002	29-JAN-12	1000000		6000000

3.6.1.2 Creation of Database and Tables in MS-SQL Server: Bank-6

Assuming Bank-6 using MS-SQL Server, Database and Data are produced by using the following SQL and T-SQL commands.

3.6.1.2.1 Creation of Database with Log files using T-SQL

```
USE MASTER
GO
CREATE DATABASE DBBL
ON
( NAME = DBBL_DAT,
FILENAME = 'I:\PROGRAM FILES\MICROSOFT SQL SERVER\MSSQL\DATA\DBBLDAT.MDF',
SIZE = 10,
MAXSIZE = 50,
FILEGROWTH = 5 )
LOG ON
```

```
( NAME = 'DBBL_LOG',
FILENAME = 'I:\PROGRAM FILES\MICROSOFT SQL SERVER\MSSQL\DATA\DBBLLOG.LDF',
SIZE = 5MB,
MAXSIZE = 25MB,
FILEGROWTH = 5MB )
GO
```

3.6.1.2.2 Creation of Database Tables

The following T-SQL commands are used to create the necessary tables in the central database server of Bank-6, which is a MS-SQL Server. SQL- Insert commands used to populate data in these tables are given in Appendix-I (d).

```
DROP TABLE CUSTOMERS;
```

```
CREATE TABLE CUSTOMERS
(C_ID CHAR(4) PRIMARY KEY,
NAME VARCHAR(30) NOT NULL,
ADDRESS VARCHAR(30),
N_ID CHAR(13) NOT NULL,
PHONE VARCHAR(30),
PHOTO IMAGE)
```

```
DROP TABLE ACCOUNT_TYPE;
```

```
CREATE TABLE ACCOUNT_TYPE
(T_ID CHAR(2) PRIMARY KEY,
DESCRIPTION VARCHAR(20) NOT NULL);
```

```
DROP TABLE ACCOUNTS;
```

```
CREATE TABLE ACCOUNTS
(A_NO CHAR(4) PRIMARY KEY,
T_ID CHAR(2) REFERENCES ACCOUNT_TYPE,
C_ID CHAR(4) REFERENCES CUSTOMERS,
OD DATETIME NOT NULL,
```

```
NOMINEE VARCHAR(30) NOT NULL,  
MIN_BAL NUMERIC(10,2) CHECK(MIN_BAL>=0),  
CURR_BAL NUMERIC(10,2) CHECK(CURR_BAL>=0),  
UNIQUE(T_ID,C_ID));
```

```
DROP TABLE TRANSACTIONS;
```

```
CREATE TABLE TRANSACTIONS  
(TR_ID CHAR(10) PRIMARY KEY,  
A_NO CHAR(4) REFERENCES ACCOUNTS,  
DESCRIPTION VARCHAR(20),  
TD DATETIME NOT NULL,  
DEBIT NUMERIC(10,2) CHECK(DEBIT>=0),  
CREDIT NUMERIC(10,2) CHECK(CREDIT>=0),  
BALANCE NUMERIC(10,2) CHECK(BALANCE>=0));
```

```
DROP TABLE VAULT;
```

```
CREATE TABLE VAULT  
(V_NO CHAR(4) PRIMARY KEY,  
LOCATION VARCHAR(20) NOT NULL,  
OWNER VARCHAR(20) NOT NULL,  
MIN_BAL NUMERIC(10,2) CHECK(MIN_BAL>=0),  
CURR_BAL NUMERIC(10,2) CHECK(CURR_BAL>=0));
```

```
DROP TABLE VAULT_TRANSACTION;
```

```
CREATE TABLE VAULT_TRANSACTION  
(TR_NO CHAR(10) PRIMARY KEY,  
V_NO CHAR(4) REFERENCES VAULT,  
TD DATETIME NOT NULL,  
DEBIT NUMERIC(10,2) CHECK(DEBIT>=0),  
CREDIT NUMERIC(10,2) CHECK(CREDIT>=0),  
BALANCE NUMERIC(10,2) CHECK(BALANCE>=0));
```

3.6.2 Commercial Banks' Database: Bank-1 Using Oracle Server

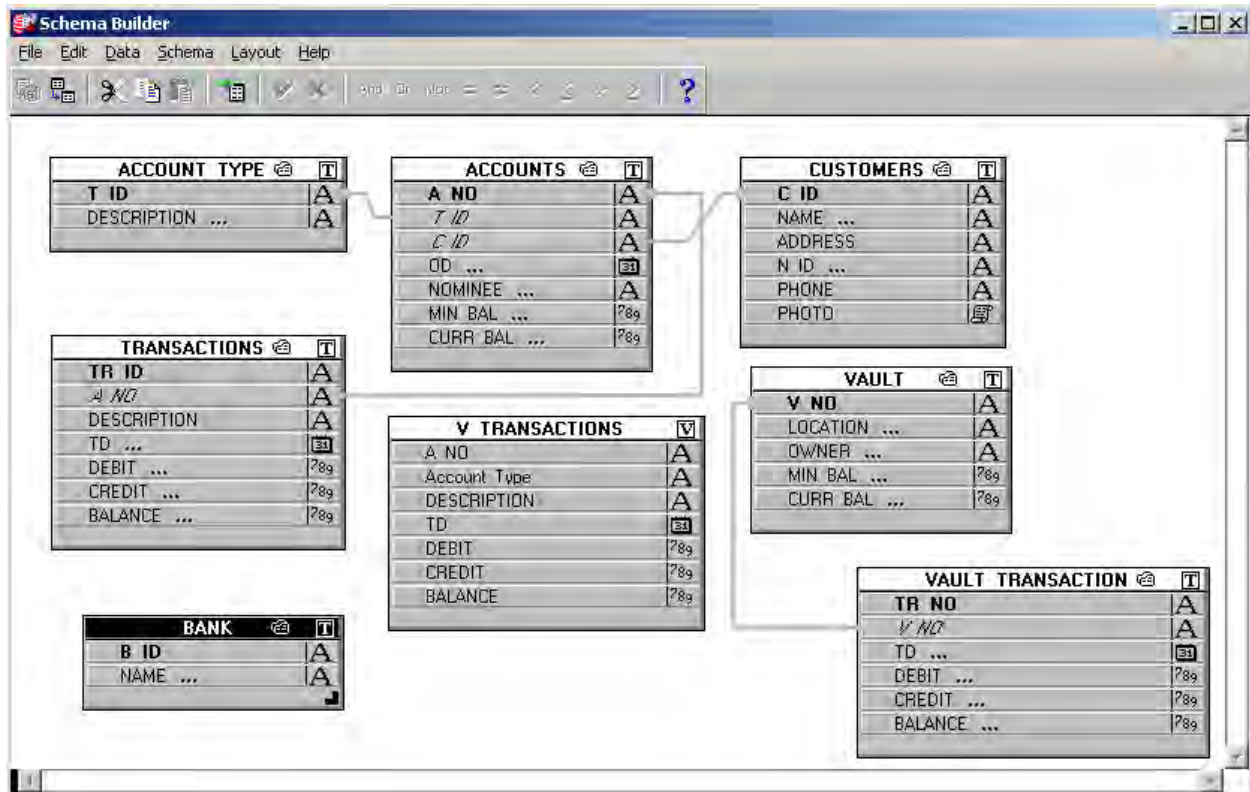


Fig. 3.5: Oracle Database Design for Bank-1

As design of Bank-1's database is same as MS-SQL Server database design used for Bank-6 and structure of entities are almost same, we skipped table descriptions here and will show the PL/SQL commands only for designing the Oracle database. PL/SQL- Insert commands used to populate data in these tables are given in Appendix-I (e).

3.6.2.1 Creation of Database Tables

```
DROP TABLE CUSTOMERS;
```

```
CREATE TABLE CUSTOMERS
```

```
(C_ID CHAR(4) PRIMARY KEY,
```

```
NAME VARCHAR2(30) NOT NULL,
```

```
ADDRESS VARCHAR2(30),
```

```
N_ID CHAR(13) NOT NULL,
```

```
PHONE VARCHAR2(30),
```

PHOTO LONG RAW);

DROP TABLE ACCOUNT_TYPE;

```
CREATE TABLE ACCOUNT_TYPE
(T_ID CHAR(2) PRIMARY KEY,
DESCRIPTION VARCHAR2(20) NOT NULL);
```

DROP TABLE ACCOUNTS;

```
CREATE TABLE ACCOUNTS
(A_NO CHAR(4) PRIMARY KEY,
T_ID CHAR(2) REFERENCES ACCOUNT_TYPE,
C_ID CHAR(4) REFERENCES CUSTOMERS,
OD DATE NOT NULL,
NOMINEE VARCHAR2(30) NOT NULL,
MIN_BAL NUMBER(10,2) CHECK(MIN_BAL>=0),
CURR_BAL NUMBER(10,2) CHECK(CURR_BAL>=0),
UNIQUE(T_ID,C_ID));
```

DROP TABLE TRANSACTIONS;

```
CREATE TABLE TRANSACTIONS
(TR_ID CHAR(10) PRIMARY KEY,
A_NO CHAR(4) REFERENCES ACCOUNTS,
DESCRIPTION VARCHAR2(20),
TD DATE NOT NULL,
DEBIT NUMBER(10,2) CHECK(DEBIT>=0),
CREDIT NUMBER(10,2) CHECK(CREDIT>=0),
BALANCE NUMBER(10,2) CHECK(BALANCE>=0));
```

DROP TABLE VAULT;

```
CREATE TABLE VAULT
(V_NO CHAR(4) PRIMARY KEY,
LOCATION VARCHAR2(20) NOT NULL,
OWNER VARCHAR2(20) NOT NULL,
```

```

MIN_BAL NUMBER(10,2) CHECK(MIN_BAL>=0),
CURR_BAL NUMBER(10,2) CHECK(CURR_BAL>=0));

DROP TABLE VAULT_TRANSACTION;

CREATE TABLE VAULT_TRANSACTION
(TR_NO CHAR(10) PRIMARY KEY,
V_NO CHAR(4) REFERENCES VAULT,
TD DATE NOT NULL,
DEBIT NUMBER(10,2) CHECK(DEBIT>=0),
CREDIT NUMBER(10,2) CHECK(CREDIT>=0),
BALANCE NUMBER(10,2) CHECK(BALANCE>=0));

```

3.6.3 Commercial Banks' Database: Bank-4 using MS-Visual FoxPro Database

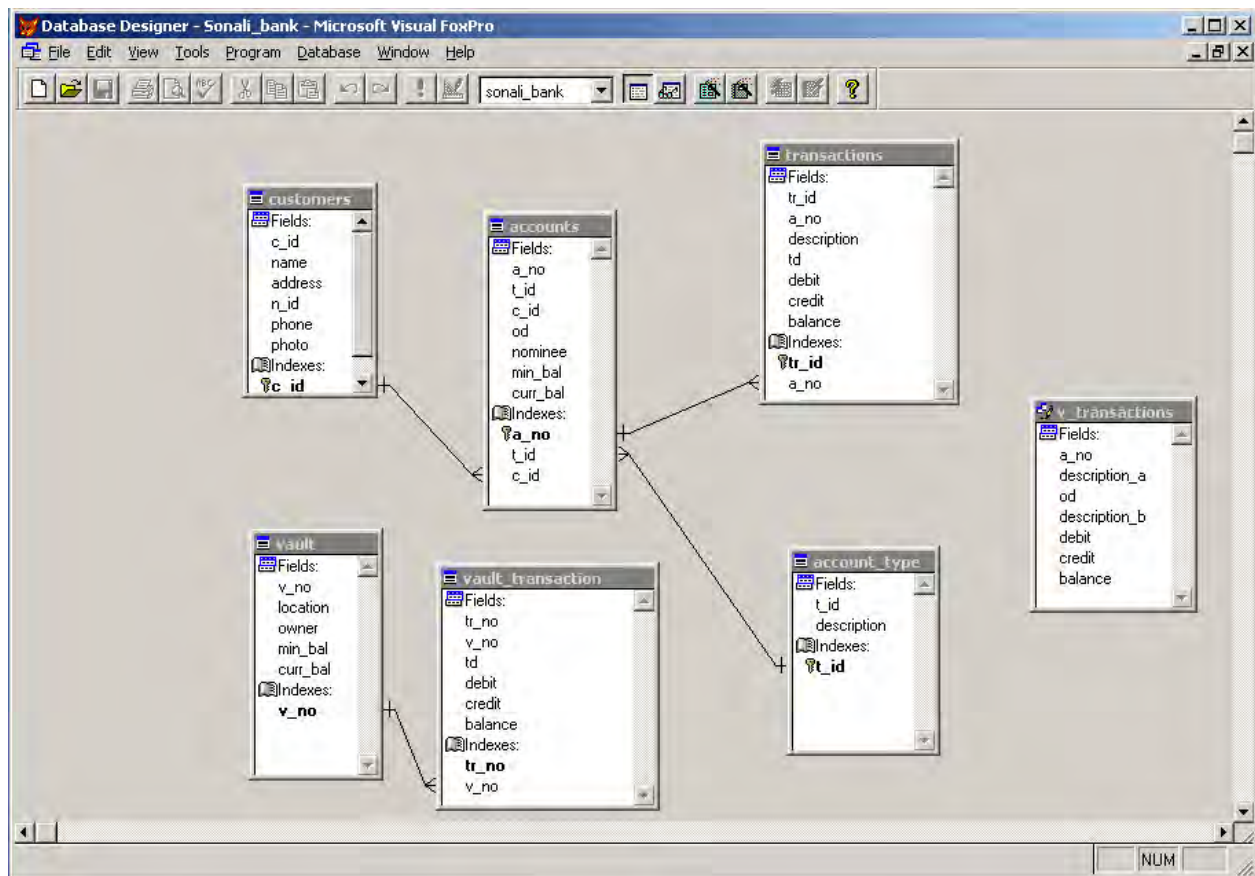


Fig. 3.6: Visual FoxPro Database Design for Bank-4

As design of Bank-4's database is same as MS-SQL Server database design used for Bank-6 and structure of entities are almost same, we skipped table descriptions here and will show the SQL commands only for designing the MS-Visual FoxPro database. SQL- Insert commands used to populate data in these tables are given in Appendix-I (f).

3.6.3.1 Creation of Database and Tables

```
CLOSE DATABASES
```

```
CLEAR
```

```
CREATE DATABASE SONALI_BANK
```

```
--SEMICOLON USED FOR LINE BREAK
```

```
DELETE TABLE CUSTOMERS
```

```
CREATE TABLE CUSTOMERS;
```

```
(C_ID CHAR(4) PRIMARY KEY, ;
```

```
NAME CHAR(30) NOT 0, ;
```

```
ADDRESS CHAR(30), ;
```

```
N_ID CHAR(13) NOT 0, ;
```

```
PHONE CHAR(30),;
```

```
PHOTO GENERAL)
```

```
DELETE TABLE ACCOUNT_TYPE
```

```
.
```

```
CREATE TABLE ACCOUNT_TYPE ;
```

```
(T_ID CHAR(2) PRIMARY KEY, ;
```

```
DESCRIPTION CHAR(20) NOT 0)
```

```
DELETE TABLE ACCOUNTS
```

```
CREATE TABLE ACCOUNTS ;
```

```
(A_NO CHAR(4) PRIMARY KEY, ;
```

```
T_ID CHAR(2) REFERENCES ACCOUNT_TYPE TAG T_ID, ;
```

```
C_ID CHAR(4) REFERENCES CUSTOMERS TAG C_ID, ;
```

```
OD DATETIME NOT 0, ;
```



```

NOMINEE CHAR(30) NOT 0, ;
MIN_BAL FLOAT CHECK(MIN_BAL>=0), ;
CURR_BAL FLOAT CHECK(CURR_BAL>=0))
--please don't put semicolon after commands

```

```

DELETE TABLE TRANSACTIONS

```

```

CREATE TABLE TRANSACTIONS ;
(TR_ID CHAR(10) PRIMARY KEY, ;
A_NO CHAR(4) REFERENCES ACCOUNTS, ;
DESCRIPTION CHAR(20), ;
TD DATETIME NOT 0, ;
DEBIT FLOAT CHECK(DEBIT>=0), ;
CREDIT FLOAT CHECK(CREDIT>=0), ;
BALANCE FLOAT CHECK(BALANCE>=0))

```

```

DELETE TABLE VAULT

```

```

CREATE TABLE VAULT ;
(V_NO CHAR(4) PRIMARY KEY, ;
LOCATION CHAR(20) NOT 0, ;
OWNER CHAR(20) NOT 0, ;
MIN_BAL FLOAT CHECK(MIN_BAL>=0), ;
CURR_BAL FLOAT CHECK(CURR_BAL>=0))

```

```

DELETE TABLE VAULT_TRANSACTION

```

```

CREATE TABLE VAULT_TRANSACTION ;
(TR_NO CHAR(10) PRIMARY KEY, ;
V_NO CHAR(4) REFERENCES VAULT, ;
TD DATETIME NOT 0, ;
DEBIT NUMBER(10,2) CHECK(DEBIT>=0), ;
CREDIT NUMBER(10,2) CHECK(CREDIT>=0), ;
BALANCE NUMBER(10,2) CHECK(BALANCE>=0))

```

3.6.4 Commercial Banks' Database: Bank-3 using MS-Access Database

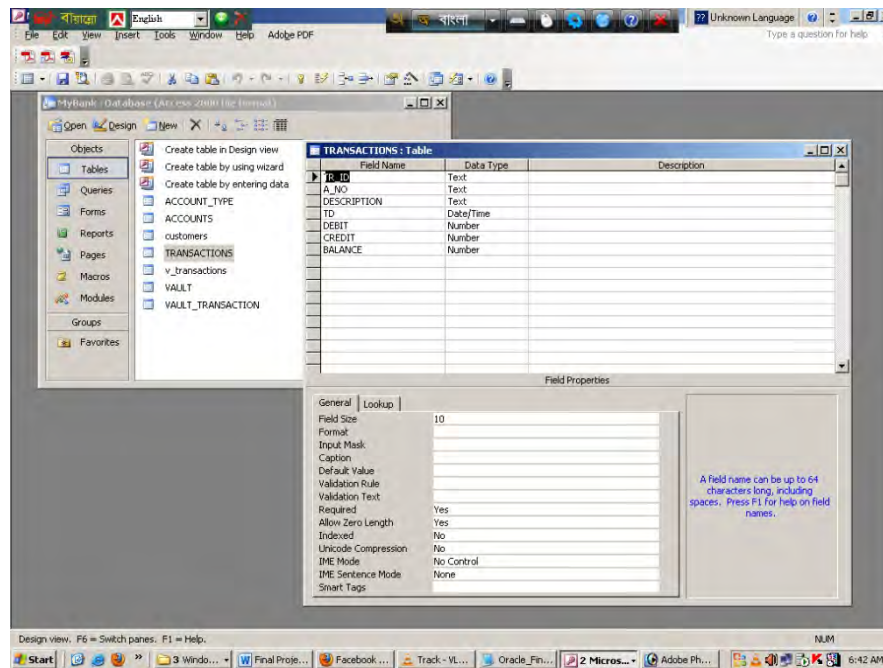


Fig. 3.7: Access Database Design for Bank-3

As design of Bank-3's database is same as MS-SQL Server database design used for Bank-6 and structure of entities are almost same, we skipped table descriptions and SQL commands for designing the MS-Access database. After creation of the database it will look like above figure (Fig. 3.7).

TR_ID	A_NO	DESCRIPTION	TD	DEBIT	CREDIT	BALANCE
TR00000001	A001	CASH	11/17/2012 AM		5000	5000
TR00000002	A001	CHEQUE	11/17/2012 AM	1000		4000
TR00000003	A001	TRANSFER	11/17/2012 AM		5000	9000
TR00000004	A001	ATM	11/17/2012 AM		7000	7000
TR00000005	A002	CASH	11/17/2012 AM		2000	2000
TR00000006	A002	CHEQUE	11/17/2012 AM	500		1500
TR00000007	A002	TRANSFER	11/17/2012 AM		1500	3000
TR00000008	A002	ATM	11/17/2012 AM	100		2900
TR00000009	A003	CASH	11/17/2012 AM		50000	50000
TR00000010	A004	CHEQUE CLEA	11/17/2012 AM		100000	100000
TR00000011	A005	CASH	11/17/2012 AM		55000	55000
TR00000012	A006	CHEQUE CLEA	11/17/2012 AM		800000	800000
TR00000013	A007	CHEQUE CLEA	11/17/2012 AM		200000	200000
TR00000014	A007	CASH WITHDR	11/17/2012 AM	200000		0
TR00000015	A007	CASH RECEIV	11/17/2012 AM		10000	10000
TR00000016	A008	CHEQUE CLEA	11/17/2012 AM		50000000	50000000
TR00000017	A008	TRANSFER	11/17/2012 AM		20000000	70000000
TR00000018	A008	CHEQUE CLEA	11/17/2012 AM	60000000		10000000
TR00000019	A009	CHEQUE CLEA	11/17/2012 AM		25000000	25000000
TR00000020	A009	CHEQUE CLEA	11/17/2012 AM		45000000	70000000
TR00000021	A009	CHEQUE CLEA	11/17/2012 AM		10000000	80000000
TR00000022	A010	CHEQUE CLEA	11/17/2012 AM		400000	400000
TR00000023	A011	TRANSFER	11/17/2012 AM		95000	95000
TR00000024	A012	TRANSFER	11/17/2012 AM		195000	195000
TR00000025	A001	VAT	12/31/2012	70		6930
TR00000026	A002	VAT	12/31/2012	29		2671
TR00000027	A003	VAT	12/31/2012	500		49500
TR00000028	A004	VAT	12/31/2012	1000		99000
TR00000029	A005	VAT	12/31/2012	650		54450
TR00000030	A006	VAT	12/31/2012	800		79200

Fig. 3.8: Access Data for Bank-3

3.6.4.1 Populating Data

After populating data it will look like the Fig.3.8.

3.6.5 Commercial Banks' Data: Bank-5 Using Text Files

As Bank-5 is using Text Files to store their data, only figure is shown below for transaction files (Fig. 3.9).

```

transactions.txt - Notepad
File Edit Format View Help
TR_ID, A_NO, DESCRIPTION, TD, DEBIT, CREDIT, BALANCE
"TR00000001", "A001", "CASH RECEIVE", "2011-04-13 04:09:25", 0, 5000.00, 5000.00
"TR00000002", "A001", "CHEQUE", "2011-06-27 04:09:25", 1000.00, 0, 4000.00
"TR00000003", "A001", "TRANSFER", "2011-08-01 04:09:25", 0, 5000.00, 9000.00
"TR00000004", "A001", "ATM", "2011-10-20 04:09:25", 2000.00, 0, 7000.00
"TR00000005", "A002", "CASH RECEIVE", "2011-04-13 04:09:25", 0, 2000.00, 2000.00
"TR00000006", "A002", "CHEQUE", "2011-07-12 04:09:25", 500.00, 0, 1500.00
"TR00000007", "A002", "TRANSFER", "2011-10-20 04:09:25", 0, 1500.00, 3000.00
"TR00000008", "A002", "ATM", "2011-12-09 04:09:25", 100.00, 0, 2900.00
"TR00000009", "A003", "CASH RECEIVE", "2011-07-12 04:09:25", 0, 50000.00, 50000.00
"TR00000010", "A004", "CHEQUE CLEARING", "2011-10-20 04:09:25", 0, 100000.00, 100000.00
"TR00000011", "A005", "CASH RECEIVE", "2011-07-12 04:09:25", 0, 55000.00, 55000.00
"TR00000012", "A006", "CHEQUE CLEARING", "2011-08-31 04:09:25", 0, 800000.00, 800000.00
"TR00000013", "A007", "CHEQUE CLEARING", "2011-09-30 04:09:25", 0, 200000.00, 200000.00
"TR00000014", "A007", "CASH WITHDRAWAL", "2011-10-20 04:09:25", 200000.00, 0, 0
"TR00000015", "A007", "CASH RECEIVE", "2011-11-09 04:09:25", 0, 10000.00, 10000.00
"TR00000016", "A008", "CHEQUE CLEARING", "2011-09-25 04:09:25", 0, 50000000.00, 50000000.00
"TR00000017", "A008", "TRANSFER", "2011-10-20 04:09:25", 0, 20000000.00, 70000000.00
"TR00000018", "A008", "CHEQUE CLEARING", "2012-01-28 04:09:25", 60000000.00, 0, 100000000.00
"TR00000019", "A009", "CHEQUE CLEARING", "2011-10-30 04:09:25", 0, 25000000.00, 25000000.00
"TR00000020", "A009", "CHEQUE CLEARING", "2011-12-09 04:09:25", 0, 45000000.00, 70000000.00
"TR00000021", "A009", "CHEQUE CLEARING", "2012-01-03 04:09:25", 0, 10000000.00, 80000000.00
"TR00000022", "A010", "CHEQUE CLEARING", "2011-11-29 04:09:25", 0, 400000.00, 400000.00
"TR00000023", "A011", "TRANSFER", "2011-12-19 04:09:25", 0, 95000.00, 95000.00
"TR00000024", "A012", "TRANSFER", "2011-12-19 04:09:25", 0, 195000.00, 195000.00
"TR00000025", "A001", "VAT", "2012-12-31 00:00:00", 70.00, 0, 6930.00
"TR00000026", "A002", "VAT", "2012-12-31 00:00:00", 29.00, 0, 2871.00
"TR00000027", "A003", "VAT", "2012-12-31 00:00:00", 500.00, 0, 49500.00
"TR00000028", "A004", "VAT", "2012-12-31 00:00:00", 1000.00, 0, 99000.00
"TR00000029", "A005", "VAT", "2012-12-31 00:00:00", 550.00, 0, 54450.00
"TR00000030", "A006", "VAT", "2012-12-31 00:00:00", 800.00, 0, 799200.00
"TR00000031", "A007", "VAT", "2012-12-31 00:00:00", 100.00, 0, 9900.00
"TR00000032", "A008", "VAT", "2012-12-31 00:00:00", 100000.00, 0, 9900000.00
"TR00000033", "A009", "VAT", "2012-12-31 00:00:00", 800000.00, 0, 79200000.00
"TR00000034", "A010", "VAT", "2012-12-31 00:00:00", 400.00, 0, 399600.00
"TR00000035", "A011", "VAT", "2012-12-31 00:00:00", 950.00, 0, 94050.00
"TR00000036", "A012", "VAT", "2012-12-31 00:00:00", 1950.00, 0, 193050.00
"TR00000037", "A001", "TAX", "2012-12-31 00:00:00", 7.00, 0, 6923.00
"TR00000038", "A002", "TAX", "2012-12-31 00:00:00", 2.90, 0, 2868.10
"TR00000039", "A003", "TAX", "2012-12-31 00:00:00", 50.00, 0, 49450.00
"TR00000040", "A004", "TAX", "2012-12-31 00:00:00", 100.00, 0, 89900.00
"TR00000041", "A005", "TAX", "2012-12-31 00:00:00", 55.00, 0, 54395.00
"TR00000042", "A006", "TAX", "2012-12-31 00:00:00", 80.00, 0, 799120.00

```

Fig. 3.9: Text Data for Bank-5

3.6.6 Commercial Banks' Data: Bank-2 using MS-Excel Files

Assuming Bank-2 Using MS-Excel Files to store their data manually, figure is shown below for transaction data only (Fig. 3.10).

TR_ID	A_NO	DESCRIPTION	TD	DEBIT	CREDIT	BALANCE
TR000000 A001		CASH RECEIVE	4/13/2011		5000	5000
TR000000 A001		CHEQUE	6/27/2011	1000		4000
TR000000 A001		TRANSFER	8/1/2011		5000	9000
TR000000 A001		ATM	10/20/2011	2000		7000
TR000000 A002		CASH RECEIVE	4/13/2011		2000	2000
TR000000 A002		CHEQUE	7/12/2011	500		1500
TR000000 A002		TRANSFER	10/20/2011		1500	3000
TR000000 A002		ATM	12/9/2011	100		2900
TR000000 A003		CASH RECEIVE	7/12/2011		50000	50000
TR000000 A004		CHEQUE CLEARING	10/20/2011		100000	100000
TR000000 A005		CASH RECEIVE	7/12/2011		55000	55000
TR000000 A006		CHEQUE CLEARING	8/31/2011		800000	800000
TR000000 A007		CHEQUE CLEARING	9/30/2011		200000	200000
TR000000 A007		CASH WITHDRAWAL	10/20/2011	200000		0
TR000000 A007		CASH RECEIVE	11/9/2011		10000	10000
TR000000 A008		CHEQUE CLEARING	3/25/2011		50000000	50000000
TR000000 A008		TRANSFER	10/20/2011		20000000	70000000
TR000000 A008		CHEQUE CLEARING	1/28/2012	60000000		10000000
TR000000 A009		CHEQUE CLEARING	10/30/2011		25000000	25000000
TR000000 A009		CHEQUE CLEARING	12/9/2011		45000000	70000000
TR000000 A009		CHEQUE CLEARING	1/3/2012		10000000	80000000
TR000000 A010		CHEQUE CLEARING	11/29/2011		400000	400000
TR000000 A011		TRANSFER	12/19/2011		95000	95000
TR000000 A012		TRANSFER	12/19/2011		195000	195000
TR000000 A001		VAT	12/31/2012	70		6930
TR000000 A002		VAT	12/31/2012	29		2871
TR000000 A003		VAT	12/31/2012	500		49500
TR000000 A004		VAT	12/31/2012	1000		99000
TR000000 A005		VAT	12/31/2012	550		54450
TR000000 A006		VAT	12/31/2012	800		799200

Fig. 3.10: Excel Data for Bank-2

3.7 Summary

In this chapter different issues regarding the design of database for the project are discussed. At first, normalization issues are discussed and Oracle database is designed for the production server of the central bank (one for temporary use and other for permanent use). Relationship among entities, constraints, data types and its size are also illustrated here. Table structures with dummy data are also presented for clear understanding. SQL commands for designing the databases are also presented in this chapter. Database design for middle-tier (linked-server) is also elaborated followed by the details designing of commercial banks' database in different platforms. In the next chapter, configuration of linked-server is presented and discussed.

Chapter 4

Designing the Middle-Tier Linked-Server

4.1 Introduction

First of all, we have setup network communication links to database servers of all banks. Communication may be direct or through third party, according to the wish of BB, considering financial budget and security. Here ISP, VPN, PSTN or optical communication can be used. Dark optical fiber can also be used, as most of the data centers are very near about BB at Motijheel. After establishing the network connection successfully, we will establish database links to all database servers from the middle-tiered linked server. In this project, MS-SQL Server is used as a linked-server; because of cost, flexibility, availability and easy manageability. Microsoft has a very powerful tool, named MS-OLEDB that can be used to access, import and export data among databases easily. Except MS-Visual FoxPro, OLEDB can be used for Oracle, SQL-Server, MS-Access, MS-Excel and even for Text files also. By using OLEDB we can directly access the mentioned databases without any middle interference. But, in case of Visual FoxPro, we can use MS-OLEDB Provider for ODBC. ODBC is another tool, though not as secured as OLEDB, can be used to access databases. But it's an old one with less security. Unfortunately, Microsoft did not provide any facility to OLEDB to access Visual FoxPro database directly as it is not a pure database engine and store data as a record concept. In that case, we will configure ODBC for Visual FoxPro and then use MS-OLEDB Provider for ODBC to get data.

4.2 Linked-Servers

A linked server configuration allows Microsoft SQL Server to execute commands against OLE DB data sources on different servers. Linked servers offer these advantages:

- Remote server access.
- The ability to issue distributed queries, updates, commands, and transactions on heterogeneous data sources across the enterprise.
- The ability to address diverse data sources similarly.

4.2.1 Linked Server Components

A linked server definition specifies an OLE DB provider and an OLE DB data source.

An OLE DB provider is a dynamic-link library (DLL) that manages and interacts with a specific data source. An OLE DB data source identifies the specific database accessible through OLE DB. Although data sources queried through linked server definitions are usually databases, OLE DB providers exist for a wide variety of files and file formats, including text files, spreadsheet data, and the results of full-text content searches.

For a data source to return data through a linked server, the OLE DB provider (DLL) for that data source must be present on the same server as SQL Server.

Typically, linked servers are used to handle distributed queries. When a client application executes a distributed query through a linked server, SQL Server breaks down the command and sends rowset requests to OLE DB. The rowset request may be in the form of executing a query against the provider or opening a base table from the provider.

4.2.2 Managing a Linked Server Definition

When setting up a linked server, we have to register the connection information and data source information with SQL Server. After registration is accomplished, that data source can always be referred to with a single logical name.

We can create or delete a linked server definition with stored procedures or through SQL Server Enterprise Manager.

With stored procedures:

- We can create a linked server definition using **sp_addlinkedserver**.
- We can delete a linked server definition using **sp_dropserver**.

With SQL Server Enterprise Manager:

- We can create a linked server definition using the SQL Server Enterprise Manager Console tree and the **Linked Servers** node (under the **Security** folder). Define the name, provider properties, server options, and security options for the linked server.

- We can edit a linked server definition by right-clicking the linked server and clicking **Properties**.
- We can delete a linked server definition by right-clicking the linked server and clicking **Delete**.

sp_addlinkedserver

sp_addlinkedserver built-in procedure creates a linked server, which allows access to distributed, heterogeneous queries against OLE DB data sources. After creating a linked server with sp_addlinkedserver, this server can then execute distributed queries. If the linked server is defined as Microsoft SQL Server, remote stored procedures can be executed.

Syntax

```
sp_addlinkedserver[ @server= ] 'server'
  [, [ @srvproduct= ] 'product_name' ]
  [, [ @provider= ] 'provider_name' ]
  [, [ @datasrc= ] 'data_source' ]
  [, [ @location= ] 'location' ]
  [, [ @provstr= ] 'provider_string' ]
  [, [ @catalog= ] 'catalog' ]
```

Arguments

[@server =] 'server' is the local name of the linked server to create. It is **sysname**, with no default.

[@srvproduct =] 'product_name' is the product name of the OLE DB data source to add as a linked server.

[@provider =] 'provider_name' is the unique programmatic identifier (PROGID) of the OLE DB provider corresponding to this data source. *provider_name* must be unique for the specified OLE DB provider installed on the current computer. The OLE DB provider is expected to be registered with the given PROGID in the registry.

[@datasrc =] 'data_source' is the name of the data source as interpreted by the OLE DB provider. When the linked server is created against the SQL Server OLE DB provider, *data_source* can be specified in the form of *servername\instancename*, which can be used to

connect to a specific instance of SQL Server running on the specified computer. *servername* is the name of the computer on which SQL Server is running, and *instancename* is the name of the specific SQL Server instance to which the user will be connected.

[**@location** =] '*location*' is the location of the database as interpreted by the OLE DB provider.

[**@provstr** =] '*provider_string*' is the OLE DB provider-specific connection string that identifies a unique data source.

[**@catalog** =] '*catalog*' is the catalog to be used when making a connection to the OLE DB provider.

The following table shows the ways that a linked server can be set up for data sources accessible through OLE DB. A linked server can be set up using more than one way for a given data source; there may be more than one row for a data source type. This table shows the **sp_addlinkedserver** parameter values to be used for setting up the linked server.

Table 4.1: sp_addlinkedserverParameter Values

Remote Data Source	OLE DB Provider	Provider Name	Data Source
SQL Server	Microsoft OLE DB Provider for SQL Server	SQLOLEDB	Servername\instancename (for specific instance)
Oracle	Microsoft OLE DB Provider for Oracle	MSDAORA	SQL*Net alias for Oracle database
Access/Jet	Microsoft OLE DB Provider for Jet	Microsoft.Jet.OLEDB.4.0	Full path name of Jet database file
ODBC	Microsoft OLE DB Provider for ODBC	MSDASQL	System DSN of ODBC data source
Microsoft Excel	Microsoft OLE DB Provider for Jet	Microsoft.Jet.OLEDB.4.0	Full path name of Excel file

4.3 OLE DB Provider for Oracle

The Microsoft OLE DB Provider for Oracle allows distributed queries to query data in Oracle databases.

To Create a Linked Server to Access an Oracle Database Instance

- We have to ensure the Oracle client software on the server running SQL Server is at the level required by the provider. The Microsoft OLE DB Provider for Oracle requires Oracle Client Software Support File version 7.3.3.4.0 or later, and SQL*Net version 2.3.3.0.4.
- We have to create an SQL*Net alias name on the server running SQL Server that points to an Oracle database instance.
- We have to execute **sp_addlinkedserver** to create the linked server, specifying **MSDAORA** as *provider_name*, and the SQL*Net alias name for the Oracle database instance as *data_source*.

Fig. 4.1: Oracle Database SQL* Net Service Provider Configuration

```

tnsnames.ora - Notepad
File Edit Format View Help

ALAM =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = b1bm-2ea9410d64)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = alam.b1bm.org)
    )
  )

Beq-local.world =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS =
        (COMMUNITY = beq.world)
        (PROTOCOL = BEQ)
        (PROGRAM = oracle73)
        (ARGS0 = oracle73ORCL)
        (ARGS = '(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))')
      )
    )
    (CONNECT_DATA = (SID = ORCL)
  )
)

Tcp-loopback.world =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS =
        (COMMUNITY = tcp.world)
        (PROTOCOL = TCP)
        (Host = 127.0.0.1)
        (Port = 1521)
      )
    )
    (CONNECT_DATA = (SID = ORCL)
  )
)

Example1.world =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS =
        (COMMUNITY = tcp.world)
        (PROTOCOL = TCP)
        (Host = Production1)
        (Port = 1521)
      )
    )
    (CONNECT_DATA = (SID = SID1)
  )
)

```

Creating a Linked-Server by Using the Microsoft OLE DB Provider for Oracle

Following T-SQL codes will create a linked server named 'ORACLEDATA' that uses the Microsoft OLE DB Provider for Oracle and assumes that the SQL*Net alias for the Oracle database is 'Alam'(Fig. 4.1).

```
--CREATING LINKED SERVERS

USE MASTER
GO
EXEC SP_DROPSERVER ORACLEDATA

EXEC SP_ADDLINKEDSERVER
    @SERVER = 'ORACLEDATA',
    @SRVPRODUCT = 'ORACLE',
    @PROVIDER = 'MSDAORA',
    @DATASRC = 'ALAM'

GO

-- @DATASRC IS THE HOST STRING IN ORACLE
--@LOCATION IS THE SERVER NAME WHERE ORACLE RUNS
--GO TO ENTERPRISE MANAGER FIND ADD LINK SERVER LONDON GO TO PROPERTIES
-- GO TO SECURITY TAB GO TO BE MADE USING THIS SECURITY CONTEXT
-- USE SCOTT AND TIGER TO CONNECT
--USE SAME SERVER FOR ORACLE AND SQL SERVER 2000
```

4.4 OLE DB Provider for Jet

The Microsoft OLE DB Provider for Jet provides an OLE DB interface to Microsoft Access databases, and allows Microsoft SQL Server distributed queries to query Access databases.

4.4.1 To Create a Linked Server to access an MS-Access Database

- We have to execute `sp_addlinkedserver` to create the linked server, specifying `Microsoft.Jet.OLEDB.4.0` as `provider_name`, and the full path name of the `Access.mdb` database file as `data_source`. The `.mdb` database file must reside on the server. `data_source` is evaluated on the server, not the client, and the path must be valid on the server.

We have created a linked server named 'ACCESS_LINK' that operates against the Access database named **DB1.mdb** in the = 'I:\BUET\DB1.MDB' directory, as follows:

```
--CREATE A LINKED SERVER FOR ACCESS DATA
USE MASTER
GO

EXEC SP_ADDLINKEDSERVER
  @SERVER = 'ACCESS_LINK',
  @PROVIDER = 'MICROSOFT.JET.OLEDB.4.0',
  @SRVPRODUCT = 'OLE DB PROVIDER FOR JET',
  @DATASRC = 'I:\BUET\DB1.MDB'
GO
```

4.4.2 To Create a Linked Server against an Excel Spreadsheet

To create a linked server definition using the Microsoft OLE DB Provider for Jet to access an Excel spreadsheet, first we have to create a named range in Excel specifying the columns and rows of the Excel worksheet to select. The name of the range can then be referenced as a table name in a distributed query.

The Microsoft OLE DB Provider for Jet 4.0 can be used to access Microsoft Excel spreadsheets. To create a linked server that accesses an Excel spreadsheet, we used the following T-SQL command.

```
--CREATE A LINKED SERVER FOR EXECL DATA FROM TRANSACTIONS

EXEC SP_ADDLINKEDSERVER 'EXCELSOURCE_VAULT',

  'JET 4.0',

  'MICROSOFT.JET.OLEDB.4.0',

  'I:\BUET\VAULT.XLS',

  NULL,

  'EXCEL 5.0'

GO

--CREATE A LINKED SERVER FOR EXECL DATA FROM VAULT
```

```
EXEC SP_ADDLINKEDSERVER 'EXCELSOURCE',
'JET 4.0',
'MICROSOFT.JET.OLEDB.4.0',
'I:\BUET\TRANSACTIONS.XLS',
NULL,
'EXCEL 5.0'
GO
```

4.4.3 To set up a Linked Server against a Text File

Microsoft OLE DB Provider for Jet can be used to access and query text files.

- To create a linked server for accessing text files directly without linking the files as tables in an Access .mdb file, we executed **sp_addlinkedserver**, as follows:

The provider is Microsoft.Jet.OLEDB.4.0 and the provider string is 'Text'. The data source is the full path name of the directory that contains the text files. A schema.ini file, which describes the structure of the text files, must exist in the same directory as the text files.

```
--CREATE A LINKED SERVER FOR TEXT DATA FROM TRANSACTIONS
USE MASTER
GO
EXEC SP_ADDLINKEDSERVER TEXTSERVER, 'JET 4.0',
'MICROSOFT.JET.OLEDB.4.0',
'I:\BUET',
NULL,
'TEXT'
GO
```

```
--CREATE A LINKED SERVER FOR TEXT DATA FROM VAULT
USE MASTER
GO
EXEC SP_ADDLINKEDSERVER TEXTSERVERVAULT, 'JET 4.0',
'MICROSOFT.JET.OLEDB.4.0',
'I:\BUET',
NULL,
'TEXT'
GO
```

4.5 OLE DB Provider for ODBC (MS-Visual FoxPro)

The OLE DB Provider for ODBC provides an OLE DB interface to ODBC data sources. Using the OLE DB Provider for ODBC, Microsoft SQL Server distributed queries can access all ODBC data.

To create a Linked Server to access an ODBC Database (MS-Visual FoxPro)

1. We have to create a System Data Source on the computer on which SQL Server is installed (Figure 4.2 and 4.3).
2. After configuring System Data Source, we have to configure the Liked-Server properties described in Figure 4.4 and 4.5.

Fig. 4.2: ODBC System Data Source Configuration

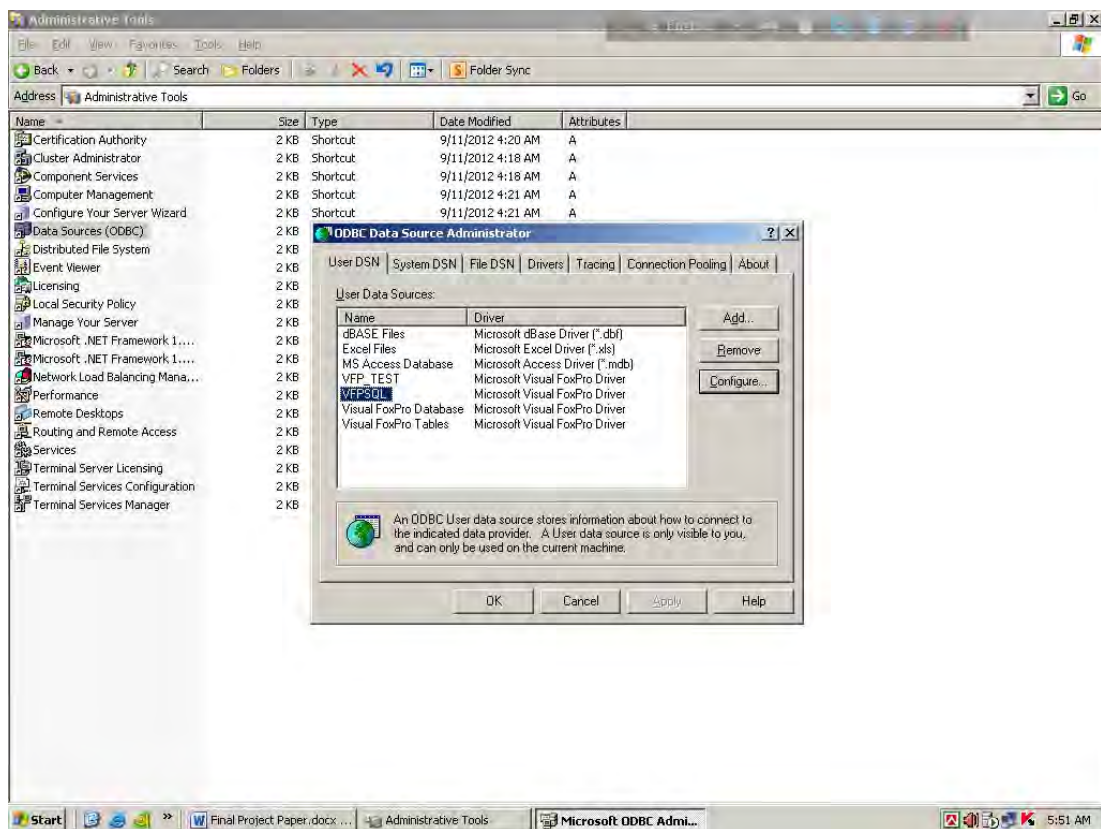


Fig. 4.3: ODBC System Data Source Location

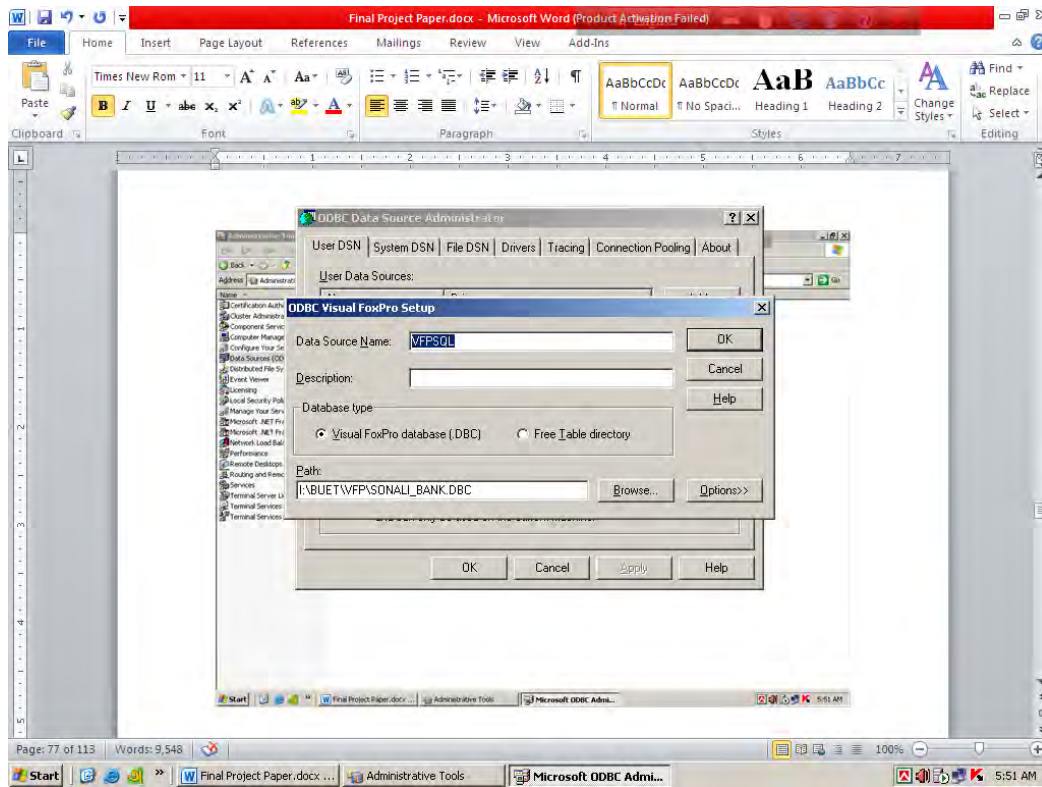


Fig. 4.4: Linked-Server Configuration for Visual FoxPro

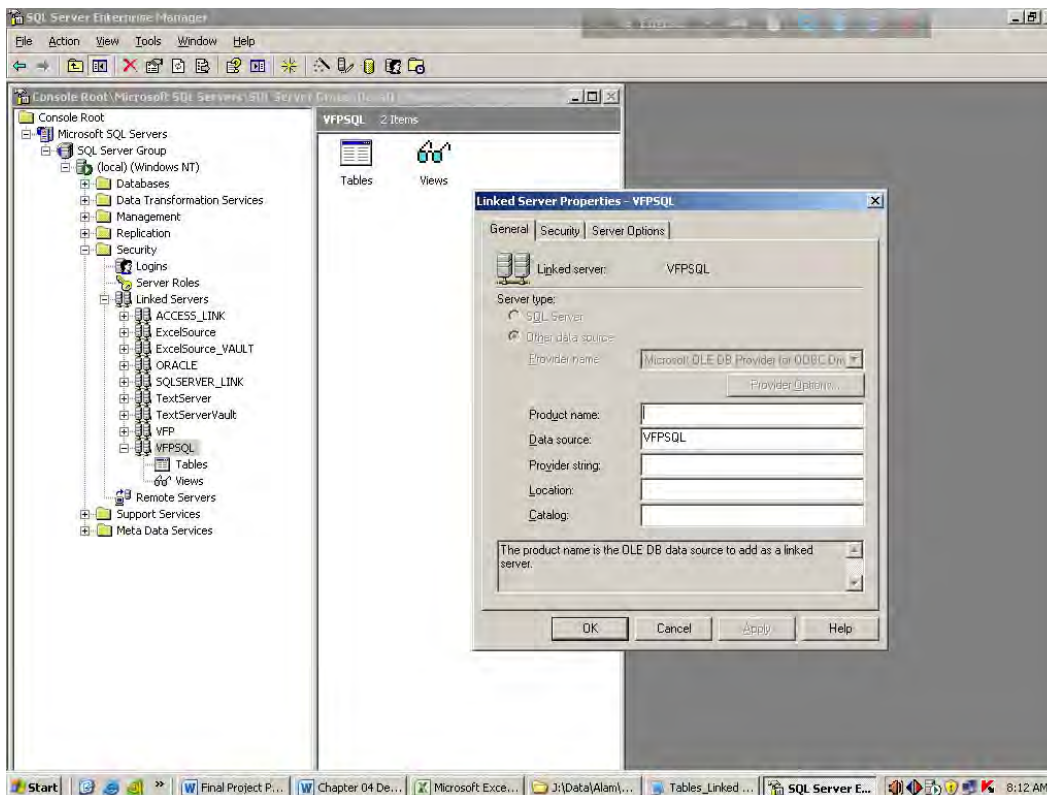
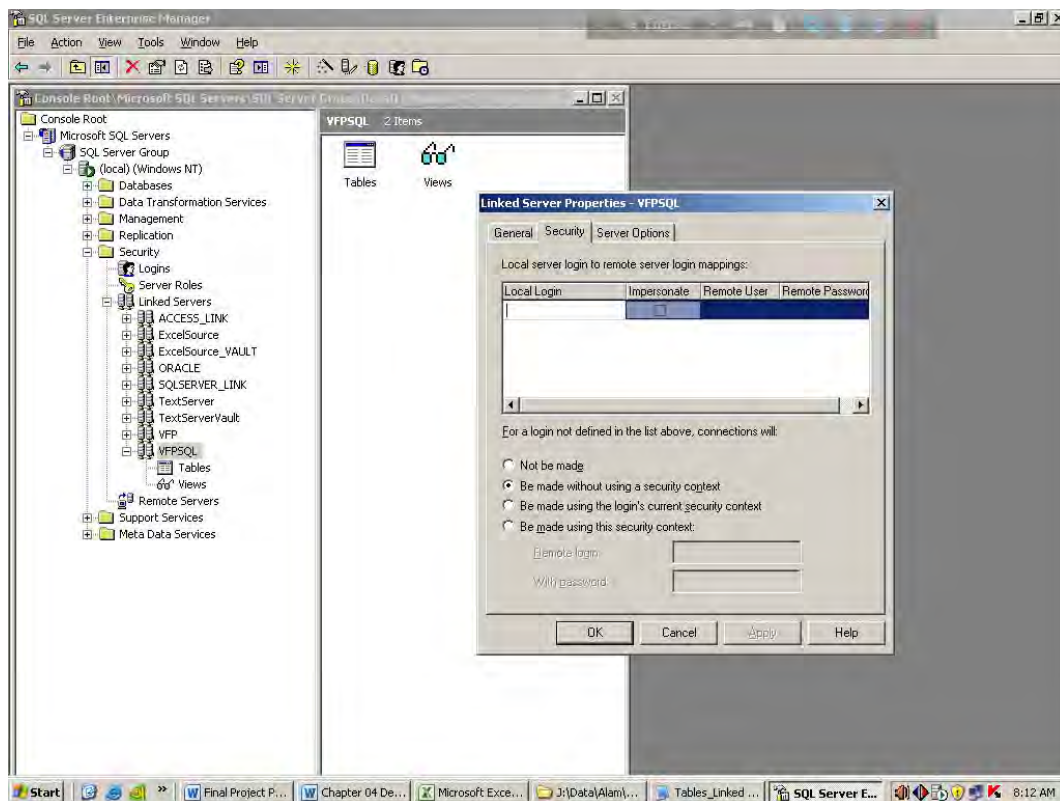


Fig. 4.5: Security Configuration for Visual FoxPro



4.6 OLE DB Provider for SQL Server

The Microsoft OLE DB Provider for SQL Server provides an OLE DB interface to Microsoft SQL Server databases. Using the OLE DB Provider for SQL Server, SQL Server distributed queries can query data in remote instances of SQL Server.

To create a Linked Server to access a SQL Server Database

- We have to execute **sp_addlinkedserver** to create the linked server, specifying **SQLOLEDB** as *provider_name*, and the network name of the server running the remote instance of SQL Server as *data_source*.

For example, to create a linked server named **SQLSERVER_LINK** that operates against the instance of SQL Server running on the server whose network name is **BANK-6**, we executed the following codes.

```
--CREATE A LINKED SERVER FOR SQL SERVER DATA
```

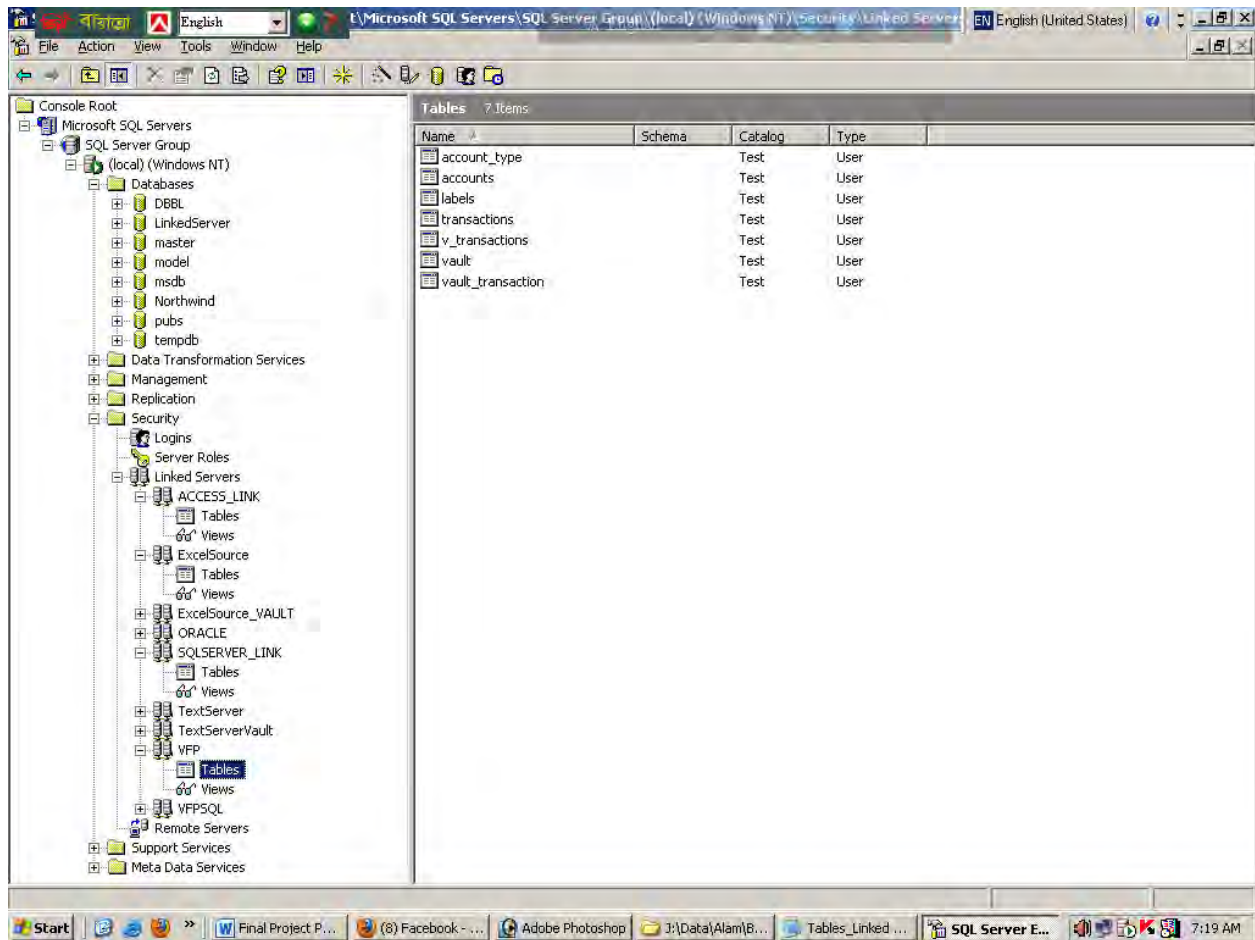
```

EXEC SP_ADDLINKEDSERVER
    @SERVER='SQLSERVER_LINK',
    @SRVPRODUCT="",
    @PROVIDER='SQLOLEDB',
    @DATASRC='BANK-6'
GO

```

After configuring all Linked Servers we will see the Enterprise Manager as shown below (Fig. 4.6).

Fig. 4.6: MS-SQL Server Enterprise Manager



4.7 Summary

This chapter mainly discusses the design and configuration of linked-server for middle-tier use. Functionality of necessary procedures and data providers are also described. Communication and security features are also tested in this chapter with due attention.

Chapter 5

Data Gathering, Filtering and Processing

5.1 Introduction

Distributed queries access data from multiple heterogeneous data sources, which can be stored on either the same or different computers. MS SQL Server supports distributed queries by using OLEDB, the Microsoft specification of an application programming interface (API) for universal data access.

Distributed queries provide SQL Server users with access to:

- Distributed data stored in multiple instances of SQL Server.
- Heterogeneous data stored in various relational and non-relational data sources accessed using an OLEDB provider.

OLE DB providers expose data in tabular objects called rowsets. SQL Server allows rowsets from OLE DB providers to be referenced in Transact-SQL statements as if they were a SQL Server table.

Tables and views in external data sources can be referenced directly in SELECT, INSERT, UPDATE, and DELETE Transact-SQL statements. Because distributed queries use OLEDB as the underlying interface, distributed queries can access traditional relational DBMS systems with SQL query processors, as well as data managed by data sources of varying capabilities and sophistication. As long as the software owning the data exposes it in a tabular rowset through an OLE DB provider, the data can be used in distributed queries. Using distributed queries in SQL Server is similar to the linked table functionality through ODBC, which is supported by Microsoft Visual FoxPro.

5.2 Distributed Query Architecture

MS SQL Server supports two methods for referencing heterogeneous OLE DB data sources in Transact-SQL statements:

- Linked server names

The system stored procedures `sp_addlinkedserver` and `sp_addlinkedsrvlogin` are used to give a server name to an OLE DB data source. Objects in these linked servers can be referenced in Transact-SQL statements using four-part names. For example, if a linked server name of `DBBL` is defined against another copy of SQL Server, the following statement references a table on that server:

```
SELECT * FROM DBBL.Northwind.dbo.Employees
```

The linked server name can also be specified in an `OPENQUERY` statement to open a rowset from the OLE DB data source. This rowset can then be referenced like a table in Transact-SQL statements.

- Ad hoc connector names

For infrequent references to a data source, the `OPENROWSET` functions are specified with the information needed to connect to the linked server. The rowset can then be referenced the same way a table is referenced in Transact-SQL statements:

```
SELECT *
FROM OPENROWSET('Microsoft.Jet.OLEDB.4.0',
                'c:\MSOffice\Access\Samples\Northwind.mdb'; 'Admin'; '' ;
                Employees)
```

SQL Server uses OLEDB to communicate between the relational engine and the storage engine. The relational engine breaks down each Transact-SQL statement into a series of operations on simple OLEDB rowsets opened by the storage engine from the base tables. This means the relational engine can also open simple OLEDB rowsets on any OLE DB data source.

The relational engine uses the OLE DB API to open the rowsets on linked servers, to fetch the rows, and to manage transactions.

For each OLEDB data source accessed as a linked server, an OLEDB provider must be present on the server running SQL Server. The set of Transact-SQL operations that can be used against a specific OLE DB data source depends on the capabilities of the OLE DB provider.

5.3 OPENQUERY

OPENQUERY executes the specified pass-through query on the given linked server, which is an OLE DB data source. The OPENQUERY function can be referenced in the FROM clause of a query as though it is a table name. The OPENQUERY function can also be referenced as the target table of an INSERT, UPDATE, or DELETE statement, subject to the capabilities of the OLE DB provider.

Syntax

```
OPENQUERY ( linked_server, 'query' )
```

Arguments

linked_server

Is an identifier representing the name of the linked server.

'*query*'

Is the query string executed in the linked server.

Remarks

OPENQUERY does not accept variables for its arguments.

Examples

This example creates a linked server named **OracleServer** against an Oracle database using the Microsoft OLE DB Provider for Oracle. Then this example uses a pass-through query against this linked server. This example assumes that an Oracle database alias called ORCLDB has been created.

```
SELECT *
FROM OPENQUERY(OracleSvr, 'SELECT name, id FROM joe.titles')
GO
```

In our project we used OPENQUERY features of MS SQL Server to run distributed queries.

5.4 Gathering Data from Linked Servers

5.4.1 Gathering data from all banks assuming all communication links are up and all linked-servers are running

The following procedure, PR_GATHER_DATA_ALL_ONLINE, will receive four parameters; starting date (@SD), ending date (@ED), data category identity (@CTG_ID) and description of the category (@CTG_DES) of the data, for example tax, vat, etc. This procedure will gather data and united them by using UNION ALL operator at first and store it to the GATHERED_DATA table. The main advantage of this procedure is that it will collect and gather data from all banks with a single click only. But the main problem is that, if any server or link goes down it will produce an error and all transactions will be rolled back.

```

CREATE PROCEDURE PR_GATHER_DATA_ALL_ONLINE
    @SD DATETIME,
    @ED DATETIME,
    @CTG_ID CHAR(5), --DATA CATEGORY ID LIKE CTG01
    @CTG_DES VARCHAR(20) --DATA CATEGORY DESCRIPTION LIKE.
AS
    DECLARE @CD DATETIME
    SELECT @CD=GETDATE()

    INSERT INTO GATHERED_DATA
    SELECT 'B1',@CTG_ID,@SD, @ED, @CD, SUM(DEBIT),SUM(CREDIT)
    FROM OPENQUERY(ORACLE, 'SELECT * FROM TRANSACTIONS') WHERE TD>=@SD AND TD<=@ED AND
    DESCRIPTION=@CTG_DES

    UNION ALL

    SELECT 'B2',@CTG_ID,@SD,@ED, @CD, SUM(DEBIT),SUM(CREDIT)
    FROM OPENQUERY(EXCELSOURCE, 'SELECT * FROM [TRANSACTIONS$]') WHERE TD>=@SD AND TD<=@ED AND
    DESCRIPTION=@CTG_DES

    UNION ALL

    SELECT 'B3',@CTG_ID,@SD,@ED, @CD, SUM(DEBIT),SUM(CREDIT)
    FROM OPENQUERY(ACCESS_LINK, 'SELECT * FROM TRANSACTIONS') WHERE TD>=@SD AND TD<=@ED AND
    DESCRIPTION=@CTG_DES

    UNION ALL

    SELECT 'B4',@CTG_ID,@SD,@ED, @CD, SUM(DEBIT),SUM(CREDIT)

```

```

FROM OPENQUERY(VFPSQL, 'SELECT * FROM TRANSACTIONS')WHERE TD>=@SD AND TD<=@ED AND
DESCRIPTION=@CTG_DES

UNION ALL

SELECT 'B5',@CTG_ID,@SD,@ED, @CD, SUM(DEBIT),SUM(CREDIT)

FROM TEXTSERVER...[TRANSACTIONS#TXT]WHERE TD>=@SD AND TD<=@ED AND DESCRIPTION=@CTG_DES

UNION ALL

SELECT 'B6',@CTG_ID,@SD,@ED, @CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(DBBL, 'SELECT * FROM TRANSACTIONS') WHERE TD>=@SD AND TD<=@ED AND
DESCRIPTION=@CTG_DES

```

5.4.1.1 Executing the Procedure to gather data when all servers are online

The following command will execute the procedure, PR_GATHER_DATA_ALL_ONLINE, and collect data from all servers from starting date '01-01-12' to ending date '01-01-14' for data category CTG01 which is CHARGES of all banks.

```
EXECUTE PR_GATHER_DATA_ALL_ONLINE '01-01-12', '01-01-14', 'CTG01', 'CHARGES'
```

Fig 5.1: Output of the above command

	B_ID	CTG_ID	SD	ED	CD	CREDIT_AMOUNT	DEBIT_AMOUNT
1	B1	CTG01	2011-01-01 00:00:00.000	2014-01-01 00:00:00.000	2013-03-31 23:02:03.263	2750.00	NULL
2	B2	CTG01	2011-01-01 00:00:00.000	2014-01-01 00:00:00.000	2013-03-31 23:02:03.263	3000.00	NULL
3	B3	CTG01	2011-01-01 00:00:00.000	2014-01-01 00:00:00.000	2013-03-31 23:02:03.263	6000.00	NULL
4	B4	CTG01	2011-01-01 00:00:00.000	2014-01-01 00:00:00.000	2013-03-31 23:02:03.263	2750.00	.00
5	B5	CTG01	2011-01-01 00:00:00.000	2014-01-01 00:00:00.000	2013-03-31 23:02:03.263	3000.00	.00
6	B6	CTG01	2011-01-01 00:00:00.000	2014-01-01 00:00:00.000	2013-03-31 23:02:03.263	3000.00	NULL

5.4.2 Gathering data from all banks assuming all communication links and all linked-servers are not online

The following procedure, PR_GATHER_DATA_ALL_NOT_ONLINE, will receive four parameters; starting date (@SD), ending date (@ED), data category identity (@CTG_ID) and description of the category (@CTG_DES) of the data, for example tax, vat, etc. This procedure will gather and insert data in to the GATHERED_DATA table bank by bank. The main advantage of this procedure is that it will collect data bank by bank with a single click only. If any server or link goes down it will produce an error and rest of the transactions will be terminated but data that are already received by the linked-server will not be rolled back.

```
CREATE PROCEDURE PR_GATHER_DATA_ALL_NOT_ONLINE
```

```

@SD DATETIME,
@ED DATETIME,
@CTG_ID CHAR(5), --DATA CATEGORY ID LIKE CTG01
@CTG_DES VARCHAR(20) --DATA CATEGORY DESCRIPTION LIKE.

```

AS

```

DECLARE @CD DATETIME

SELECT @CD=GETDATE()

INSERT INTO GATHERED_DATA

SELECT 'B1',@CTG_ID,@SD, @ED, @CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(ORACLE, 'SELECT * FROM TRANSACTIONS') WHERE TD>=@SD AND TD<=@ED AND
DESCRIPTION=@CTG_DES

INSERT INTO GATHERED_DATA

SELECT 'B2',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(EXCELSOURCE, 'SELECT * FROM [TRANSACTIONS$]') WHERE TD>=@SD AND TD<=@ED AND
DESCRIPTION=@CTG_DES

INSERT INTO GATHERED_DATA

SELECT 'B3',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(Access_Link, 'SELECT * FROM TRANSACTIONS') WHERE TD>=@SD AND TD<=@ED AND
DESCRIPTION=@CTG_DES

INSERT INTO GATHERED_DATA

SELECT 'B4',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(VFPSQL, 'SELECT * FROM TRANSACTIONS')WHERE TD>=@SD AND TD<=@ED AND
DESCRIPTION=@CTG_DES

INSERT INTO GATHERED_DATA

SELECT 'B5',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM TEXTSERVER...[TRANSACTIONS#TXT]WHERE TD>=@SD AND TD<=@ED AND DESCRIPTION=@CTG_DES

INSERT INTO GATHERED_DATA

SELECT 'B6',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(DBBL, 'SELECT * FROM TRANSACTIONS')..TRANSACTIONS WHERE TD>=@SD AND TD<=@ED
AND DESCRIPTION=@CTG_DES

```

5.4.2.1 Executing the Procedure to gather data when all servers are not online

The following command will execute the procedure, PR_GATHER_DATA_ALL_NOT_ONLINE, and collect data from all servers from starting date '01-01-12' to ending date '01-01-14' for data category CTG01 which is CHARGES of all available banks.

```
EXECUTE PR_GATHER_DATA_ALL_NOT_ONLINE '01-01-12', '01-01-14', 'CTG01', 'CHARGES'
```

5.4.3 Gathering data from a specific bank assuming the communication link and linked-server is online

The following procedure, PR_GATHER_DATA_SPECIFIC_SERVER, will receive four parameters; starting date (@SD), ending date (@ED), bank identity number (@BANK_ID), data category identity (@CTG_ID) and description of the category (@CTG_DES) of the data, for example tax, vat, etc. This procedure will gather and insert data in to the GATHERED_DATA table of the required bank.

```
CREATE PROCEDURE PR_GATHER_DATA_SPECIFIC_SERVER
    @SD DATETIME,--STARTING DATE
    @ED DATETIME,--ENDING DATE
    @CTG_ID CHAR(5), --DATA CATEGORY ID LIKE CTG01
    @CTG_DES VARCHAR(20), --DATA CATEGORY DESCRIPTION LIKE.
    @BANK_ID CHAR(2)--BANK ID
AS
    DECLARE @CD DATETIME
    SELECT @CD=GETDATE()

    IF @BANK_ID='B1'
    BEGIN
        INSERT INTO GATHERED_DATA
        SELECT 'B1',@CTG_ID,@SD, @ED, @CD, SUM(DEBIT),SUM(CREDIT)
        FROM OPENQUERY(ORACLE, 'SELECT * FROM TRANSACTIONS') WHERE TD>=@SD AND TD<=@ED AND
        DESCRIPTION=@CTG_DES

        PRINT 'DATA OF B1 GATHERED'
    END
    ELSE IF @BANK_ID='B2'
    BEGIN
```

```

INSERT INTO GATHERED_DATA

SELECT 'B2',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(EXCELSOURCE, 'SELECT * FROM [TRANSACTIONS$]') WHERE TD>=@SD AND TD<=@ED AND
DESCRIPTION=@CTG_DES

PRINT 'DATA OF B2 GATHERED'

END

ELSE IF @BANK_ID='B3'

BEGIN

INSERT INTO GATHERED_DATA

SELECT 'B3',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(Access_Link, 'SELECT * FROM TRANSACTIONS') WHERE TD>=@SD AND TD<=@ED AND
DESCRIPTION=@CTG_DES

PRINT 'DATA OF B3 GATHERED'

END

ELSE IF @BANK_ID='B4'

BEGIN

INSERT INTO GATHERED_DATA

SELECT 'B4',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(VFSQL, 'SELECT * FROM TRANSACTIONS')WHERE TD>=@SD AND TD<=@ED AND
DESCRIPTION=@CTG_DES

PRINT 'DATA OF B4 GATHERED'

END

ELSE IF @BANK_ID='B5'

BEGIN

INSERT INTO GATHERED_DATA

SELECT 'B5',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM TEXTSERVER...[TRANSACTIONS#TXT]WHERE TD>=@SD AND TD<=@ED AND DESCRIPTION=@CTG_DES

PRINT 'DATA OF B5 GATHERED'

END

ELSE IF @BANK_ID='B6'

BEGIN

INSERT INTO GATHERED_DATA

SELECT 'B6',@CTG_ID,@SD,@ED,@CD, SUM(DEBIT),SUM(CREDIT)

FROM OPENQUERY(DBBL, 'SELECT * FROM TRANSACTIONS')..TRANSACTIONS WHERE TD>=@SD AND TD<=@ED
AND DESCRIPTION=@CTG_DES

END

ELSE PRINT 'INVALID BANK CODE SELECTED'

```


5.4.3.1 Executing the Procedure to gather data for a specific server

The following command will execute the procedure, PR_GATHER_DATA_SPECIFIC_SERVER, and collect data from the specific server (B1) during the period '01-01-12' to '01-01-14' for data category CTG01 which is CHARGES of that bank.

```
EXECUTE PR_GATHER_DATA_SPECIFIC_SERVER '01-01-12', '01-01-14', 'CTG01', 'CHARGES', 'B1'
```

5.4.4 Procedure to gather vault data when all servers are online

Since vault management is a separate management, we need separate procedures to handle the data in this case. The following procedure, PR_GATHER_VAULT_DATA_ALL_ONLINE, will receive one parameter: data category identity (@CTG_ID). This procedure will gather data and joined them by using UNION ALL operator and store it to the GATHERED_DATA table. The main advantage of this procedure is that it will collect and gather data from all banks with a single click only. But the main problem is that, if any server or link goes down it will produce an error and all transactions will be rolled back.

```
CREATE PROCEDURE PR_GATHER_VAULT_DATA_ALL_ONLINE
    @CTG_ID CHAR(5) --DATA CATEGORY ID LIKE CTG01
AS
    DECLARE @CD DATETIME
    SELECT @CD=GETDATE()

    INSERT INTO GATHERED_DATA
    SELECT 'B1',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
    FROM OPENQUERY(ORACLE, 'SELECT * FROM VAULT')

    UNION ALL

    SELECT 'B2',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
    FROM OPENQUERY(EXCELSOURCE_VAULT, 'SELECT * FROM [VAULT$]')

    UNION ALL

    SELECT 'B3',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
    FROM OPENQUERY(ACCESS_LINK, 'SELECT * FROM VAULT')

    UNION ALL

    SELECT 'B4',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
```

```

FROM OPENQUERY(VFPSQL, 'SELECT * FROM VAULT')

UNION ALL

SELECT 'B5',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
FROM TEXTSERVERVAULT...[VAULT#TXT]

UNION ALL

SELECT 'B6',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
FROM OPENQUERY(DBBL, 'SELECT * FROM VAULT')

```

5.4.4.1 Executing Procedure to gather vault data when all servers are online

The following command will execute the procedure, PR_GATHER_VAULT_DATA_ALL_ONLINE, and collect data from all servers for data category CTG09 which is LIQUID ASSEST of all available banks. By default it will gather data of the current date.

```
EXECUTE PR_GATHER_VAULT_DATA_ALL_ONLINE 'CTG09'
```

Fig 5.2: Output of the above command

	B_ID	CTG_ID	SD	ED	CD	CREDIT_AMOUNT	DEBIT_AMOUNT
1	B1	CTG06	2012-02-19 05:03:28.090	2012-02-19 05:03:28.090	2012-02-19 05:03:28.077	58000000.00	85000000.00
2	B2	CTG06	2012-02-19 05:03:28.090	2012-02-19 05:03:28.090	2012-02-19 05:03:28.077	87000000.00	140000000.00
3	B3	CTG06	2012-02-19 05:03:28.090	2012-02-19 05:03:28.090	2012-02-19 05:03:28.077	100000000.00	200000000.00
4	B4	CTG06	2012-02-19 05:03:28.090	2012-02-19 05:03:28.090	2012-02-19 05:03:28.077	55000000.00	85000000.00
5	B5	CTG06	2012-02-19 05:03:28.090	2012-02-19 05:03:28.090	2012-02-19 05:03:28.077	87000000.00	160000000.00
6	B6	CTG06	2012-02-19 05:03:28.090	2012-02-19 05:03:28.090	2012-02-19 05:03:28.077	50000000.00	100000000.00

5.4.5 Procedure to gather vault data when all servers are not online

The following procedure, PR_GATHER_VAULT_DATA_ALL_NOT_ONLINE, will receive one parameter, data category identity (@CTG_ID). This procedure will gather and insert data into the GATHERED_DATA table bank by bank. The main advantage of this procedure is that it will collect data bank by bank with a single click only. If any server or link goes down, it will produce an error and rest of the transactions will be terminated but data that are already received by the linked-server will not be rolled back.

```

CREATE PROCEDURE PR_GATHER_VAULT_DATA_ALL_NOT_ONLINE
    @CTG_ID CHAR(5) --DATA CATEGORY ID LIKE CTG01
AS
    DECLARE @CD DATETIME
    SELECT @CD=GETDATE()

```

```

INSERT INTO GATHERED_DATA
SELECT 'B1',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
FROM OPENQUERY(ORACLE, 'SELECT * FROM VAULT')

```

```

INSERT INTO GATHERED_DATA
SELECT 'B2',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
FROM OPENQUERY(EXCELSOURCE_VAULT, 'SELECT * FROM [VAULT$]')

```

```

INSERT INTO GATHERED_DATA
SELECT 'B3',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
FROM OPENQUERY(Access_Link, 'SELECT * FROM VAULT')

```

```

INSERT INTO GATHERED_DATA
SELECT 'B4',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
FROM OPENQUERY(VFPSQL, 'SELECT * FROM VAULT')

```

```

INSERT INTO GATHERED_DATA
SELECT 'B5',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
FROM TEXTSERVERVAULT...[VAULT#TXT]

```

```

INSERT INTO GATHERED_DATA
SELECT 'B6',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
FROM OPENQUERY(DBBL, 'SELECT * FROM VAULT')

```

5.4.5.1 Executing Procedure to gather vault data when all servers are not online

The following command will execute the procedure, PR_GATHER_VAULT_DATA_ALL_NOT_ONLINE, and collect data from all servers for data category CTG09 which is LIQUID ASSEST of all available banks. By default it will gather data of the current date.

```
EXECUTE PR_GATHER_VAULT_DATA_ALL_NOT_ONLINE 'CTG09'
```

5.4.6 Procedure to gather Vault data for a specific server

The following procedure, PR_GATHER_VAULT_DATA_SPECIFIC_SERVER, will receive two parameter data category identity (@CTG_ID) and bank identity number (@BANK_ID). This procedure will gather and insert data into the GATHERED_DATA table of the required bank.

```

CREATE PROCEDURE PR_GATHER_VAULT_DATA_SPECIFIC_SERVER
    @CTG_ID CHAR(5), --DATA CATEGORY ID LIKE CTG01
    @BANK_ID CHAR(2) --BANK ID
AS
    DECLARE @CD DATETIME
    SELECT @CD=GETDATE()

IF @BANK_ID='B1'
    BEGIN
        INSERT INTO GATHERED_DATA
        SELECT 'B1',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
        FROM OPENQUERY(ORACLE, 'SELECT * FROM VAULT')
        PRINT 'DATA OF B1 GATHERED'
    END
ELSE IF @BANK_ID='B2'
    BEGIN
        INSERT INTO GATHERED_DATA
        SELECT 'B2',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
        FROM OPENQUERY(EXCELSOURCE_VAULT, 'SELECT * FROM [VAULT$]')
        PRINT 'DATA OF B2 GATHERED'
    END
ELSE IF @BANK_ID='B3'
    BEGIN
        INSERT INTO GATHERED_DATA
        SELECT 'B3',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)
        FROM OPENQUERY(ACCESS_LINK, 'SELECT * FROM VAULT')
        PRINT 'DATA OF B3 GATHERED'
    END
ELSE IF @BANK_ID='B4'

```

```

BEGIN

    INSERT INTO GATHERED_DATA

    SELECT 'B4',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)

    FROM OPENQUERY(VFPSQL, 'SELECT * FROM VAULT')

    PRINT 'DATA OF B4 GATHERED'

END

ELSE IF @BANK_ID='B5'

    BEGIN

        INSERT INTO GATHERED_DATA

        SELECT 'B5',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)

        FROM TEXTSERVERVAULT...[VAULT#TXT]

        PRINT 'DATA OF B5 GATHERED'

    END

ELSE IF @BANK_ID='B6'

    BEGIN

        INSERT INTO GATHERED_DATA

        SELECT 'B6',@CTG_ID,GETDATE(), GETDATE(), @CD, SUM(MIN_BAL),SUM(CURR_BAL)

        FROM DBBL..VAULT

        PRINT 'DATA OF B6 GATHERED'

    END

ELSE

    PRINT 'INVALID BANK CODE SELECTED'

```

5.4.6.1 Executing Procedure to gather Vault data for a specific server

The following command will execute the procedure, PR_GATHER_VAULT_DATA_SPECIFIC_SERVER, and collect data from all servers for data category CTG09 which is LIQUID ASSEST of that bank (B1). By default it will gather data of the current date.

```
EXECUTE PR_GATHER_VAULT_DATA_SPECIFIC_SERVER 'CTG09', 'B1'
```

5.4.7 Filter Data after Gathering from Linked Servers

After gathering data, it is important to check whether all required data are successfully reached into the linked server database or not. Received data will be stored into the GATHERED_DATA

table at first. Then the following procedure, FILTER_DATA, will be run. This procedure will receive four parameters; process identity number (@PR_ID), data category identity number(@CTG_ID), description and objective of the process (@DESCRIPTION) and who run the procedure(@RUN_BY). This procedure will insert process information into PROCESS_INFO table and put a flag into the FILTERING_DATA table to show that data of the required process reached successfully. If flag for any bank is found as 'N', it will indicate that data of that bank is not reached successfully due to unavailability of the server or link-down problem. In that case we have to run the procedure named PR_GATHER_DATA_SPECIFIC_SERVER to gather the missing information. When we will be satisfied that all data are gathered successfully, we will transfer this data to the temporary location of the production server. Most powerful data processing technique, i.e., cursor, is used to do the job successfully.

```

CREATE PROCEDURE FILTER_DATA
    @PR_ID CHAR(5),
    @CTG_ID CHAR(5),
    @DESCRIPTION VARCHAR(50),
    @RUN_BY VARCHAR(20)
AS
    INSERT INTO PROCESS_INFO VALUES(@PR_ID,@CTG_ID,@DESCRIPTION,GETDATE(),@RUN_BY)
DECLARE
    @TEMP_B_ID CHAR(2),
    @B_ID CHAR(2),
    @NAME VARCHAR(50)
DECLARE FILTER_DATA_CURSOR CURSOR
FOR
    SELECT * FROM BANK
OPEN FILTER_DATA_CURSOR
    FETCH NEXT FROM FILTER_DATA_CURSOR INTO @B_ID, @NAME
WHILE @@FETCH_STATUS=0
BEGIN
    SELECT @TEMP_B_ID =(SELECT B_ID FROM GATHERED_DATA WHERE CTG_ID=@CTG_ID
        AND B_ID=@B_ID)

```

```

IF @TEMP_B_ID IS NOT NULL
    BEGIN
        INSERT INTO FILTERING_DATA VALUES(@B_ID,@PR_ID,'Y')
        PRINT @B_ID+' '+@CTG_ID
    END
ELSE
    BEGIN
        INSERT INTO FILTERING_DATA VALUES(@B_ID,@PR_ID,'N')
    END
FETCH NEXT FROM FILTER_DATA_CURSOR INTO @B_ID, @NAME
END
CLOSE FILTER_DATA_CURSOR
DEALLOCATE FILTER_DATA_CURSOR

```

5.4.7.1 Executing Procedure to Filter Data after Gathering from Linked Servers

The following command will execute the procedure, FILTER_DATA for data category CTG04 which is IMPORT of different banks. This procedure will work after the request of the process number 'PR004' which is run by 'KAMAL'.

```
EXECUTE FILTER_DATA 'PR004','CTG02','IMPORT','KAMAL'
```

5.5 Summary

The above simulation and experimental results verify that the proposed system developed in the lab is working properly. Moreover, distributed query architecture, written procedures to gather, filter and process data are also discussed clearly. Precaution regarding smooth data gathering in case of communication disruption are also taken care of properly.

Chapter 6

Transferring Data from Linked-Server to Production Server

6.1 Introduction

After gathering and filtering data, our next step is to send data to the production server of the central bank. In this regard, at first we will send the data to the temporary location (schema) of the production server. In the temporary schema we will again verify the data to see that all necessary data has been received by the production server successfully. Because, there may be several issues that may hamper this transportation process. It might be security issues, network responsibility or any other software problems that may hamper smooth and accurate transportation of data. To do this job, we have selected MS-Data Transformation Services (DTS). It's a fantastic built-in service of MS-SQL Server to export and import data among heterogeneous databases and data files, like text and spread sheets.

6.2 MS-Data Transformation Services (DTS)

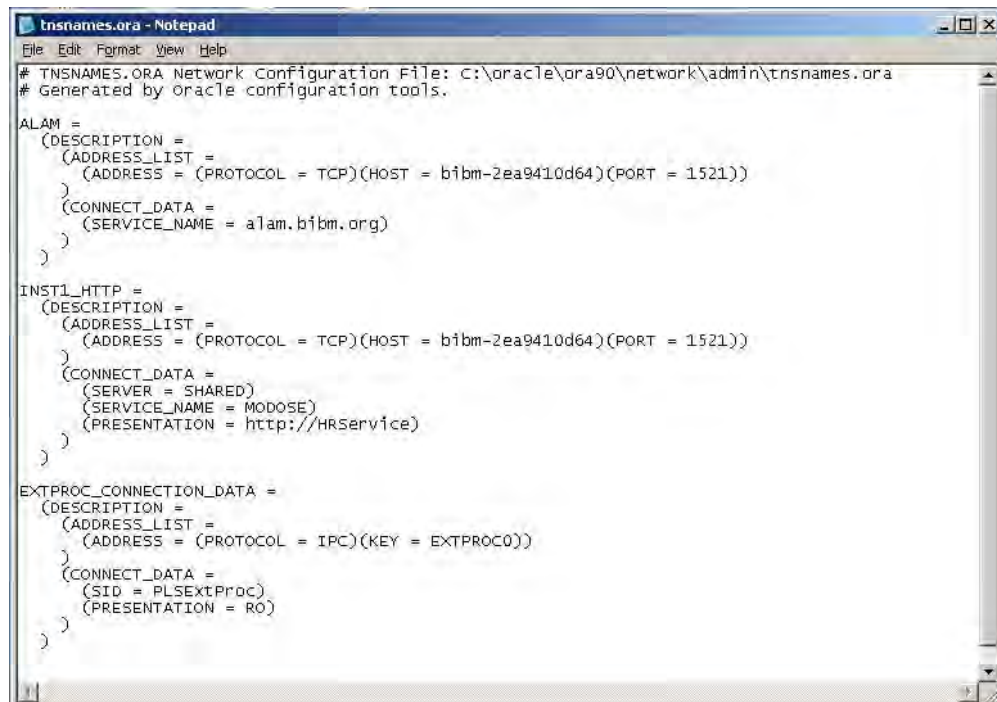
We can use Data Transformation Services (DTS) to copy data between a source and destination and optionally apply column-level transformations to the data. The Transform Data task is the most basic implementation of the data pump engine in Data Transformation Services (DTS).

The Transform Data task is optimized for insert-based copying and transforming of column-level data between commercial databases, spreadsheets, and text files. We can use the task to copy and transform data between any supported OLE DB connections. Because, the task handles such a wide variety of data sources and transformation scenarios, we will frequently use one or more instances of it when creating packages that consolidate data from disparate sources.

6.3 Configuring MS-Data Transformation Services (DTS)

Here we will transfer data from MS SQL Server to Oracle database server by using MS OLEDB. Oracle will receive data by using oracle SQL-Net services. We must configure tnsnames.ora file in this regard as follows (Fig. 6.1).

Fig. 6.1: Configuring oracle SQL-Net services



```

tnsnames.ora - Notepad
File Edit Format View Help
# TNSNAMES.ORA Network Configuration File: C:\oracle\ora90\network\admin\tnsnames.ora
# Generated by oracle configuration tools.

ALAM =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = bibm-2ea9410d64)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = alam.bibm.org)
    )
  )

INST1_HTTP =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = bibm-2ea9410d64)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = SHARED)
      (SERVICE_NAME = MODOSE)
      (PRESENTATION = http://HRService)
    )
  )

EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
    )
    (CONNECT_DATA =
      (SID = PLSEXTPrdc)
      (PRESENTATION = RO)
    )
  )

```

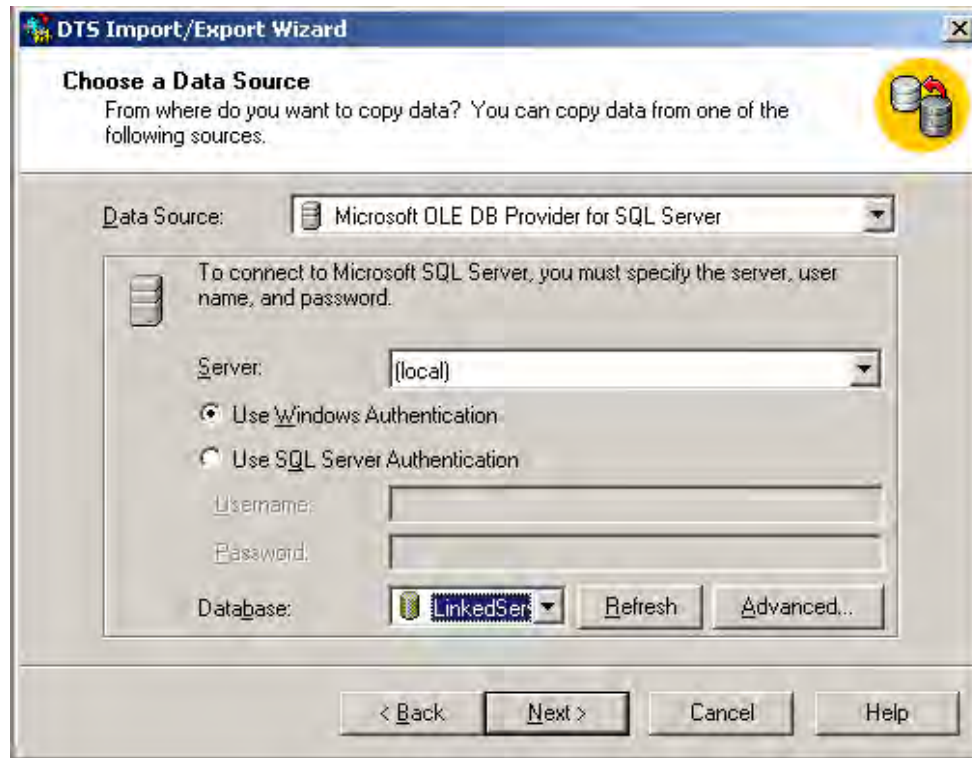
After configuring oracle SQL-Net services, we will open Import and Export Data utility wizard from Microsoft SQL Server of Windows Program menu. We will see the first dialog box as below (Fig. 6.2).

Fig. 6.2: Export Import Wizard of MS-DTS

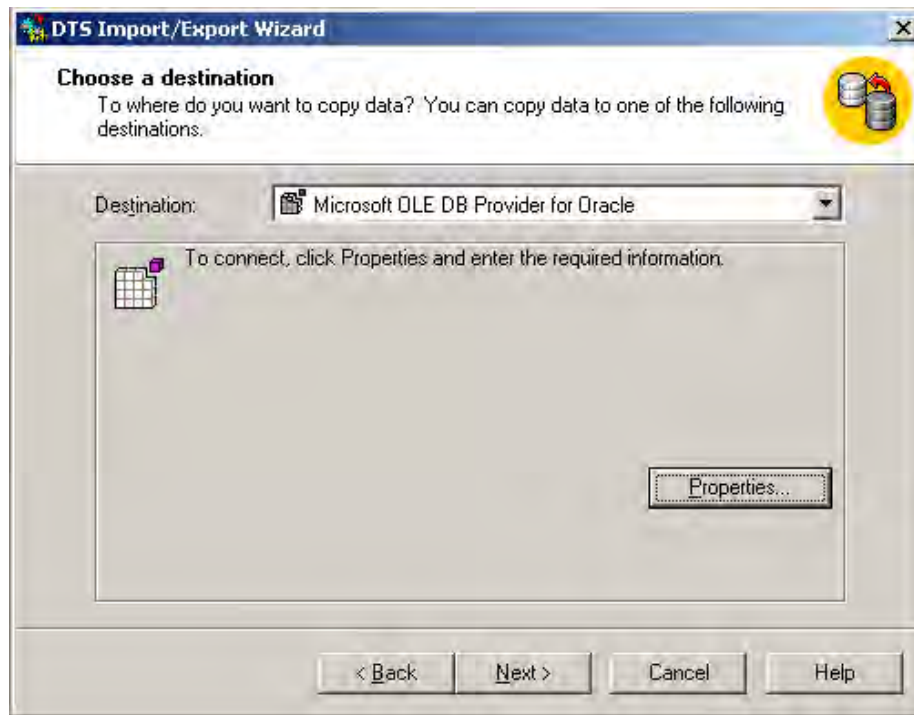
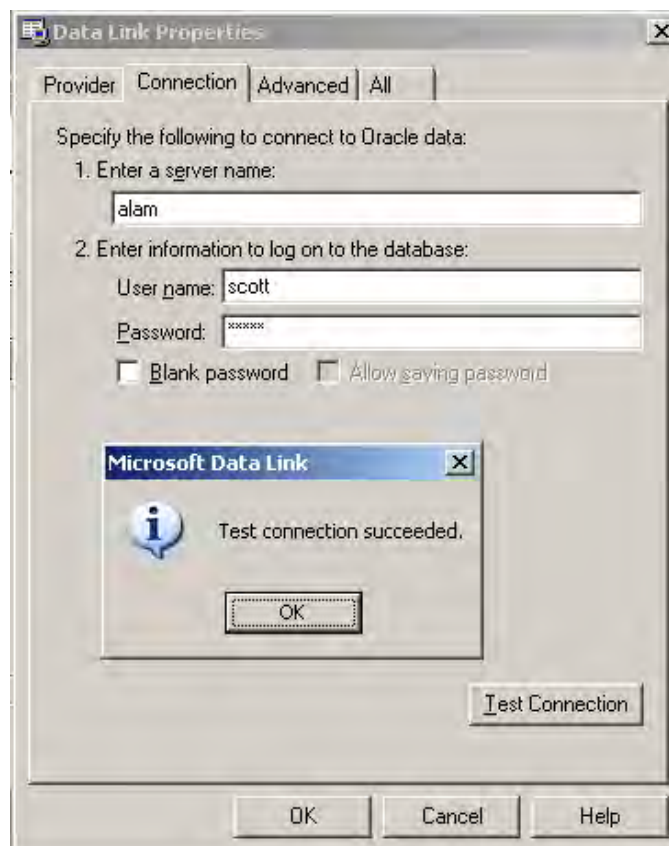


Then we will select next button to get the data source wizard (Fig. 6.3). After selecting Next button the data source wizard will appear as follows.

Fig. 6.3: Data Source Wizard



In this box we will select a data source, i.e., Microsoft OLEDB Provider for SQL Server, because we are using MS-SQL Server as a linked server. In the next dialog box (Fig. 6.4) we will select the destination database. It is an Oracle database for our project. So we will select Microsoft OLEDB Provider for Oracle. Then 'Properties' button will be clicked to configure the database connection properties for Oracle. Here we will give the Oracle host string, user name and password to connect (Fig. 6.5). We will also test the connectivity by pressing the 'Test Connection' button (Fig. 6.5). Then we will press the Next button to go to the next phase (Fig. 6.6).

Fig. 6.4: Selecting Destination Database**Fig. 6.5: Connecting Oracle Database**

'Specify Table Copy or Query' box will be appeared (Fig. 6.6). We will select copy tables and views radio button and press 'Next' button. From the 'Select Source Tables and Views' box we will select only GATHERED_DATA table as we will transfer data only for this table (Fig. 6.7).

Fig. 6.6: Specify Table Copy or Query Wizard

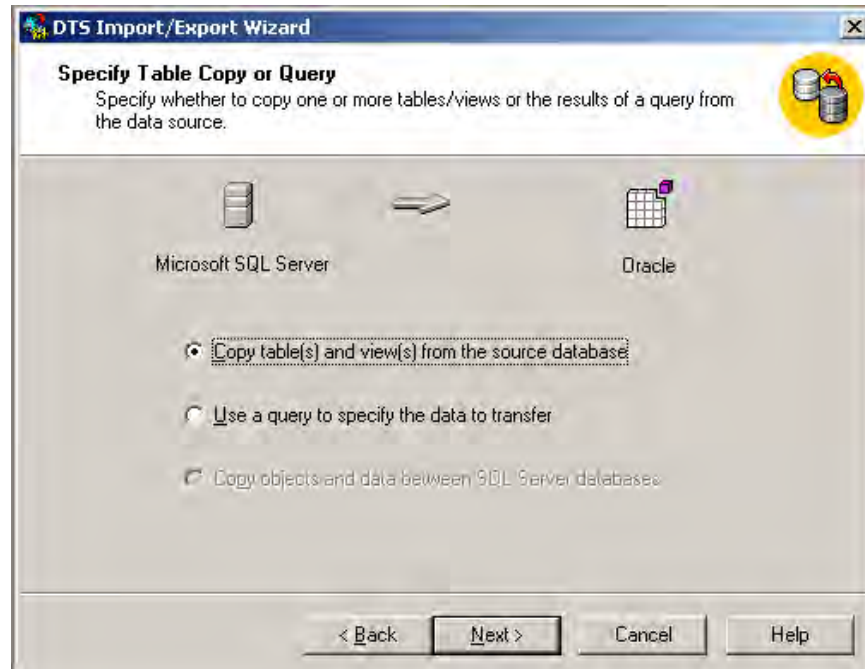
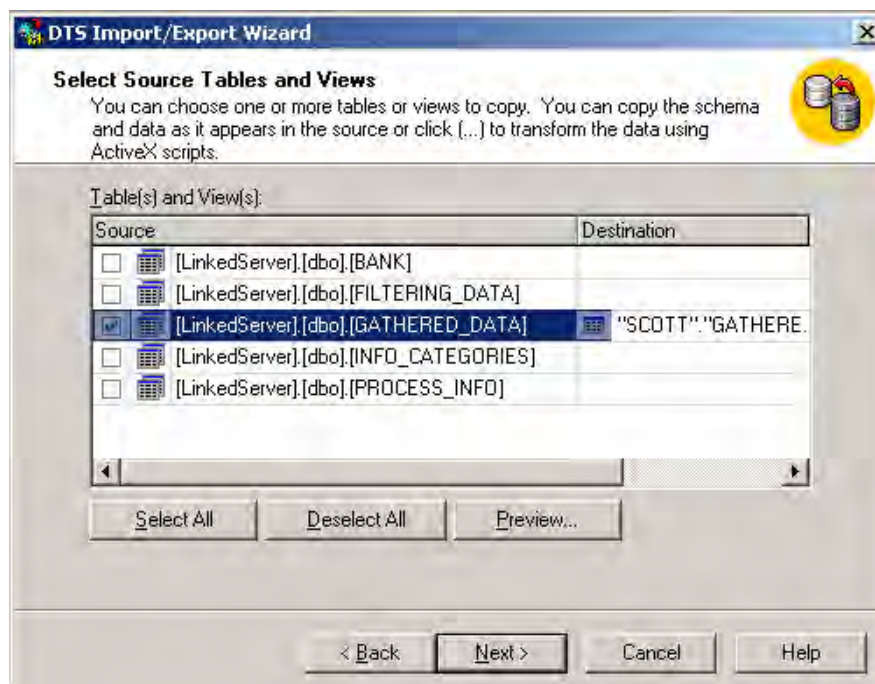


Fig. 6.7: Selecting Source Tables

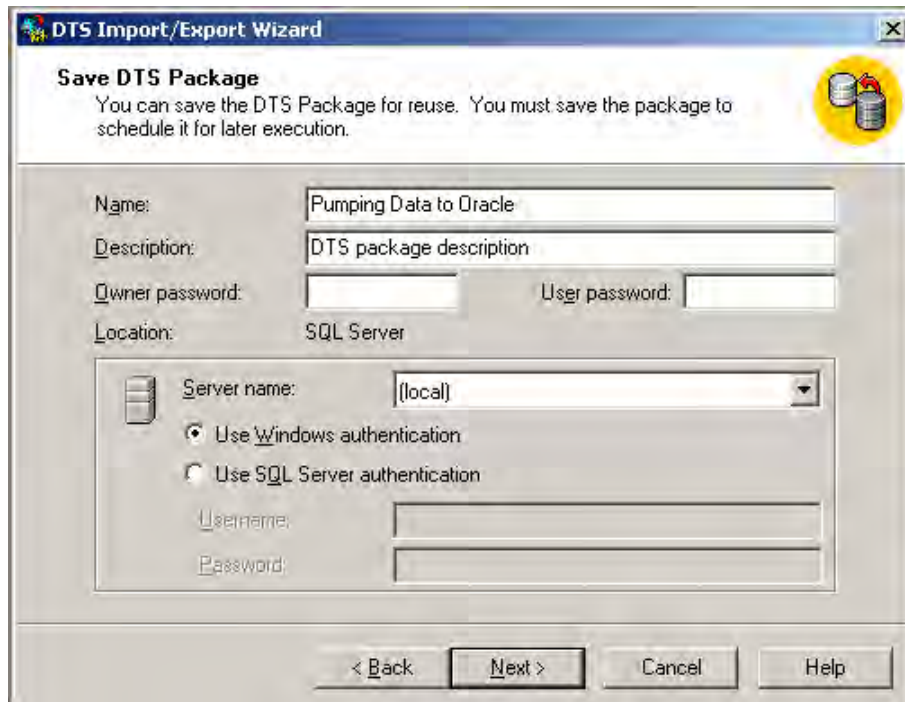


Now, we will save the DTS package as follows, because we will use it to transfer data all times (Fig. 6.8 and 6.9).

Fig. 6.8: Saving DTS Package



Fig. 6.9: Saving DTS Package



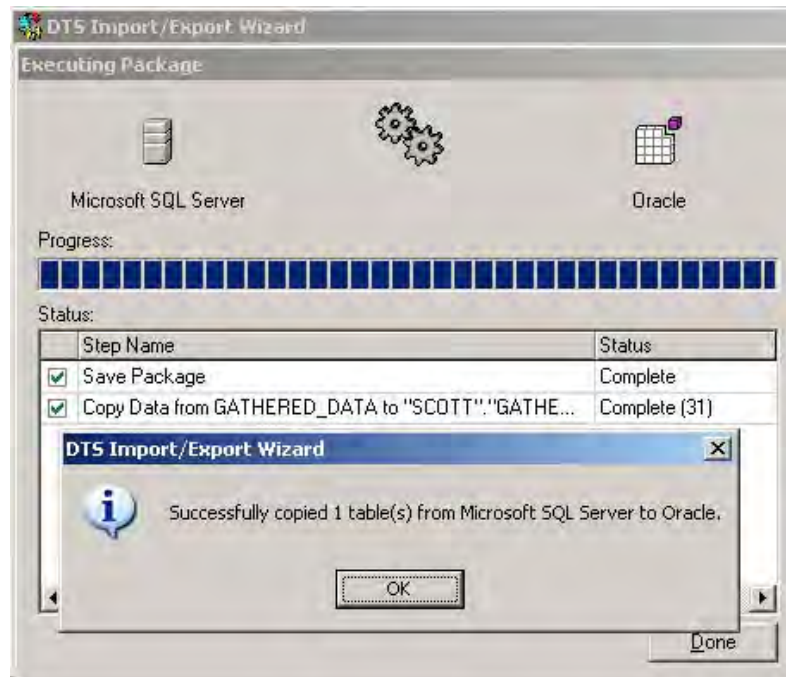
Finally, we will press the 'Finish' button to complete the job (Fig. 6.10). Data will be transferred successfully by these processes.

Fig. 6.10: Completing DTS Services



After successful transfer of data we will be notified by a message as follows (Fig. 6.11).

Fig. 6.11: Successful Transfer Message



6.4 Filtering Data after receiving from Linked Servers

After receiving data from linked server, it is important to check again whether all required data are successfully reached into the temporary schema of the production server database or not. Received data will be stored into the GATHERED_DATA table at first. Then the following procedure, FILTER_DATA, will be run. This procedure will receive four parameters; process identity number (V_PR_ID), data category identity number(V_CTG_ID), description and objective of the process (V_DESCRIPTION) and who run the procedure(V_RUN_BY). This procedure will insert process information into PROCESS_INFO table and put a flag into the FILTERING_DATA table to show that data of the required process reached successfully. If flag for any bank is found as 'N', it will indicate that data of that bank is not reached successfully due to unavailability of the server or link-down problem. If we see that all required data are not reached successfully, linked-server must send the data set again by using export data utility described in section 6.3.

6.4.1 Creating the Procedure

```
CREATE OR REPLACE PROCEDURE FILTER_DATA (V_PR_ID IN CHAR,V_CTG_ID IN CHAR,V_DESCRIPTION IN
VARCHAR,V_RUN_BY IN VARCHAR)
IS
BEGIN
    INSERT INTO PROCESS_INFO VALUES(V_PR_ID,V_CTG_ID,V_DESCRIPTION,SYSDATE,V_RUN_BY);
    COMMIT;--OTHERWISE FOREIGN KEY WILL PRODUCE ERROR NO PK FOUND
    DECLARE
        CURSOR FILTER_DATA_CURSOR IS SELECT B_ID FROM BANK;
        CURSOR GATHERED_DATA_CURSOR( V_CTG_ID CHAR) IS SELECT B_ID FROM SCOTT.GATHERED_DATA
        WHERE CTG_ID=V_CTG_ID;
        V_B_ID CHAR(2);
        TEMP_B_ID CHAR(2);
        TEMP_DUAL CHAR(8);
    BEGIN
        OPEN FILTER_DATA_CURSOR;
        IF FILTER_DATA_CURSOR%ISOPEN THEN
            LOOP
                FETCH FILTER_DATA_CURSOR INTO V_B_ID;
```

```
        EXIT WHEN FILTER_DATA_CURSOR%NOTFOUND;
            INSERT INTO FILTERING_DATA VALUES(V_B_ID, V_PR_ID,'N');
    END LOOP;
    CLOSE FILTER_DATA_CURSOR;
    COMMIT;
ELSE
    DBMS_OUTPUT.PUT_LINE('UNABLE TO OPEN CURSOR');
END IF;

OPEN GATHERED_DATA_CURSOR(V_CTG_ID);
IF GATHERED_DATA_CURSOR%ISOPEN THEN
LOOP
    FETCH GATHERED_DATA_CURSOR INTO V_B_ID;
    EXIT WHEN GATHERED_DATA_CURSOR%NOTFOUND;
        UPDATE FILTERING_DATA SET STATUS='Y' WHERE B_ID=V_B_ID;
    END LOOP;

ELSE
    DBMS_OUTPUT.PUT_LINE('UNABLE TO OPEN CURSOR');
END IF;
COMMIT;
END;
EXCEPTION

WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('NO DATA FOUND');
END;
```

6.4.2 Executing the Procedure

To run the above procedure we will execute the following command.

```
EXECUTE FILTER_DATA ('PR005','CTG01','IMPORT','KAMAL');
```


6.5 Loading the Processed Data from Temporary Schema to Permanent Schema of the Production Server

When we will be satisfied that all data are gathered successfully by executing the above procedure, we will transfer this data to the permanent location of the production server for future use and delete the data set from the GATHERED_DATA table of temporary schema. Following procedure, FINALLY_LOAD_DATA, will do this job.

6.5.1 Creating the Procedure

```
CREATE OR REPLACE PROCEDURE FINALLY_LOAD_DATA
IS
BEGIN
    INSERT INTO GATHERED_DATA (B_ID,CTG_ID,SD,ED,CREDIT_AMOUNT,DEBIT_AMOUNT)
    SELECT * FROM SCOTT.GATHERED_DATA;
    DELETE FROM SCOTT.GATHERED_DATA;
    COMMIT;
END;
```

6.5.2 Executing the Procedure

To run the above procedure, FINALLY_LOAD_DATA, we will execute the following command.

```
EXECUTE FINALLY_LOAD_DATA ;
```

6.6 Summary

This chapter mainly includes the final destination of the data after proper filtering and process in the linked-server. Data received from linked-server is also verified here to ensure that necessary data reached successfully. Required procedures that are developed for this purpose are also tested here followed by execution.

Chapter 7

Interactive Form and Report Design for End-Users

7.1 Introduction

So far what we did for this project is to collect data from heterogeneous data sources by using linked-server concept, check and filter data after gathering from different servers, process it and transfer data to the ultimate destination. What is done is a technical job and must be used by technical persons. But when data will be reached to the ultimate users, we need some common platforms so that general users of different departments can easily access, view and use the data and take official prints to submit it to the policy makers, like deputy governors and governor of the central bank, which was our main objective of the project. To do this we need some forms and reports. Here we will use Oracle Forms Developer and Report Developer of 'Oracle Developer 6i' utility. The following sections will describe the whole process step by step.

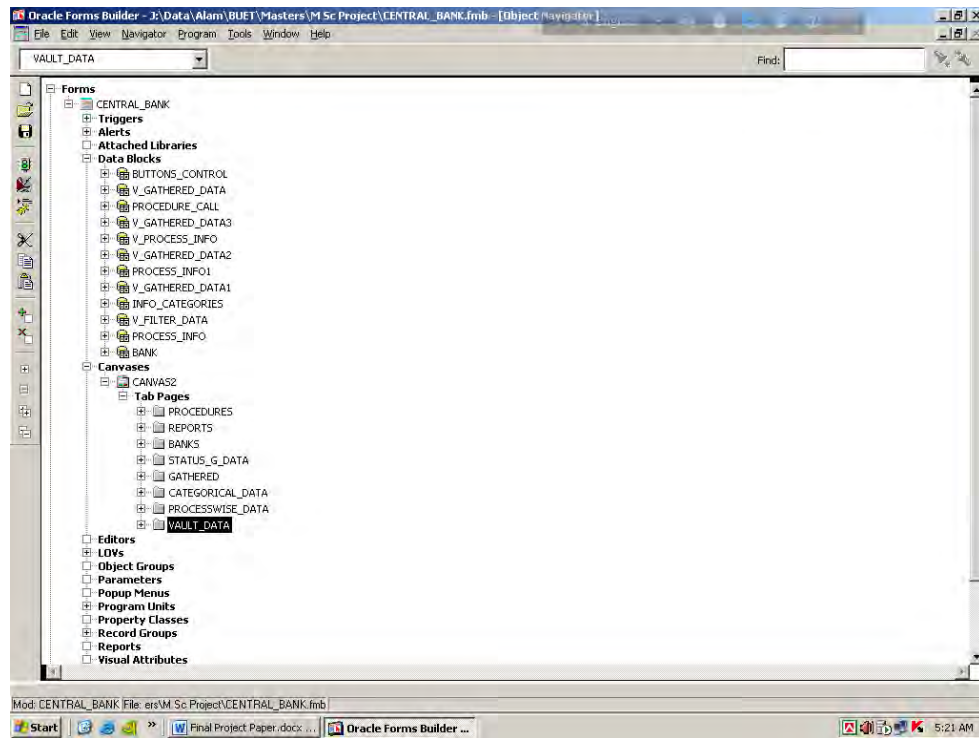
7.2 Interactive Form Design

For this project we need eight forms that will be integrated to a Tab Canvas for easy use. Out of these eight tabs, six for data viewing, one for procedure runs and other for report running. The steps for designing all the forms are same, except the two for report and procedure run. So we will discuss the procedures to design only one form, avoiding others, as it will be redundant description.

7.2.1 Vault Data

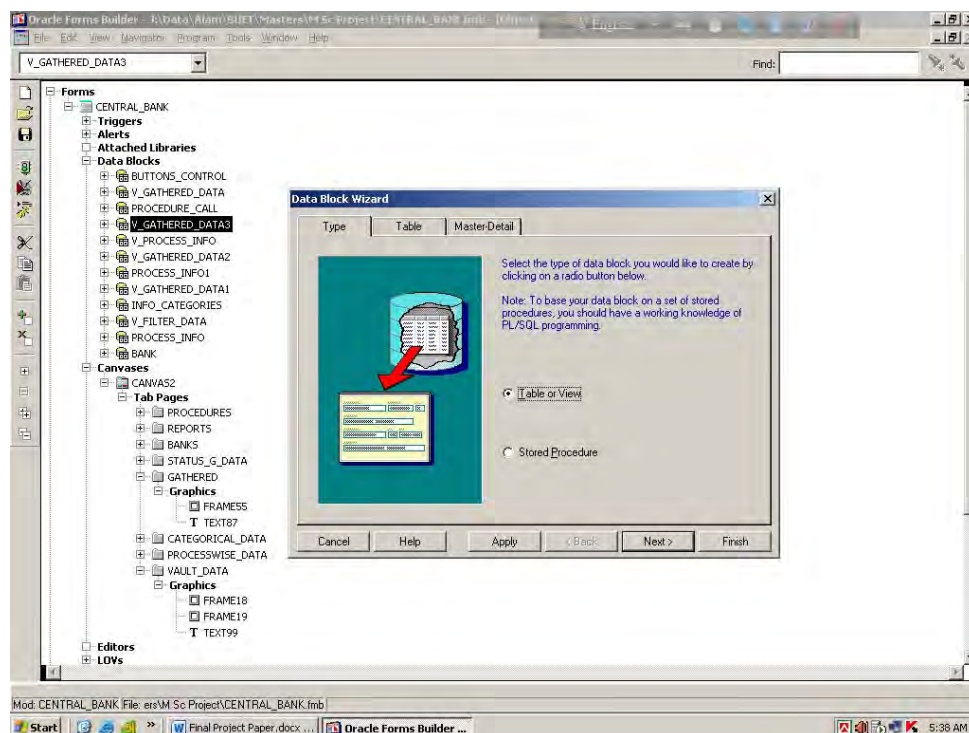
At first, we will run Oracle Developer Form Designer. It will be looked like the following one. We will design the first form to show the Vault Data.

Fig. 7.1: Oracle Forms Builder



From the Form-Tree we will select Data Block option and click the green plus (+) sign at the left icon bar (Fig. 7.1). Data block wizard will appear as follows (Fig. 7.2).

Fig. 7.2: Data Block Wizard (Type)



From the above wizard (Fig. 7.2), we will select 'Table or View' option to select a table or view whose data will be shown in the form. After selecting the option we will select V_GATHERED_DATA view and all of its fields for our project (Fig. 7.3).

Fig. 7.3: Data Block Wizard (Table)

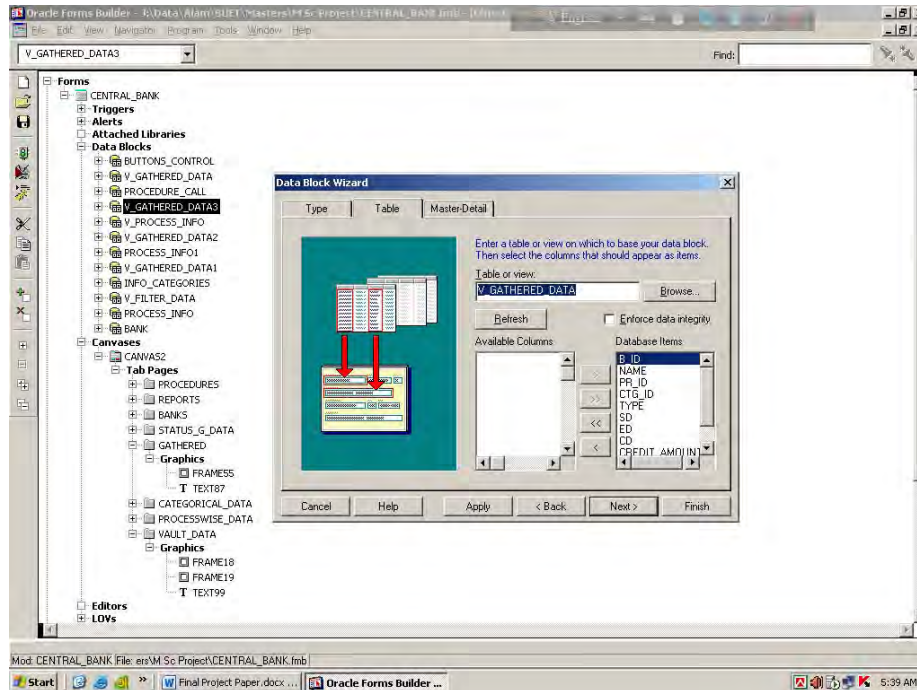
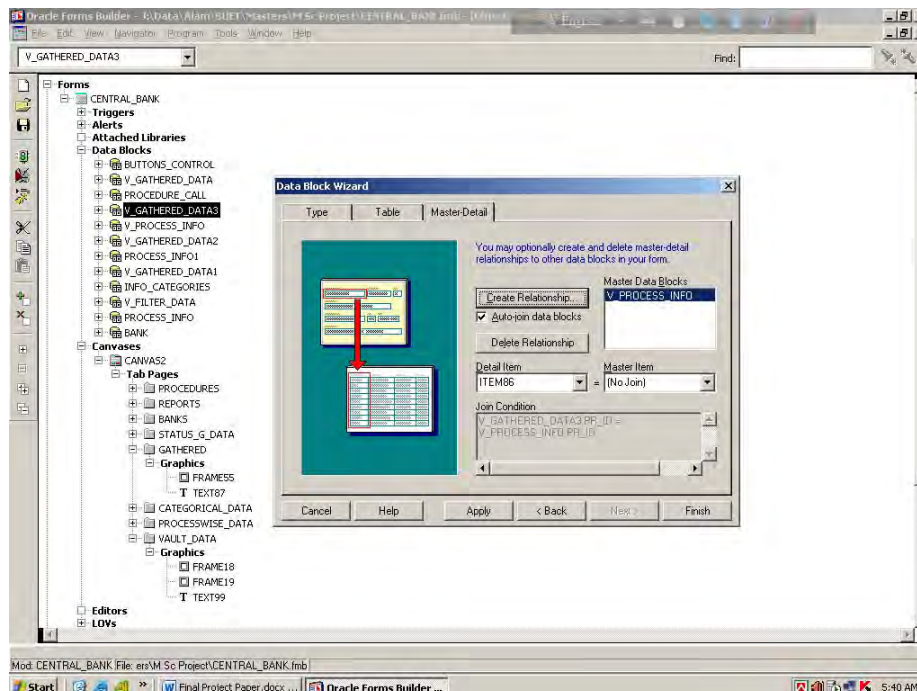
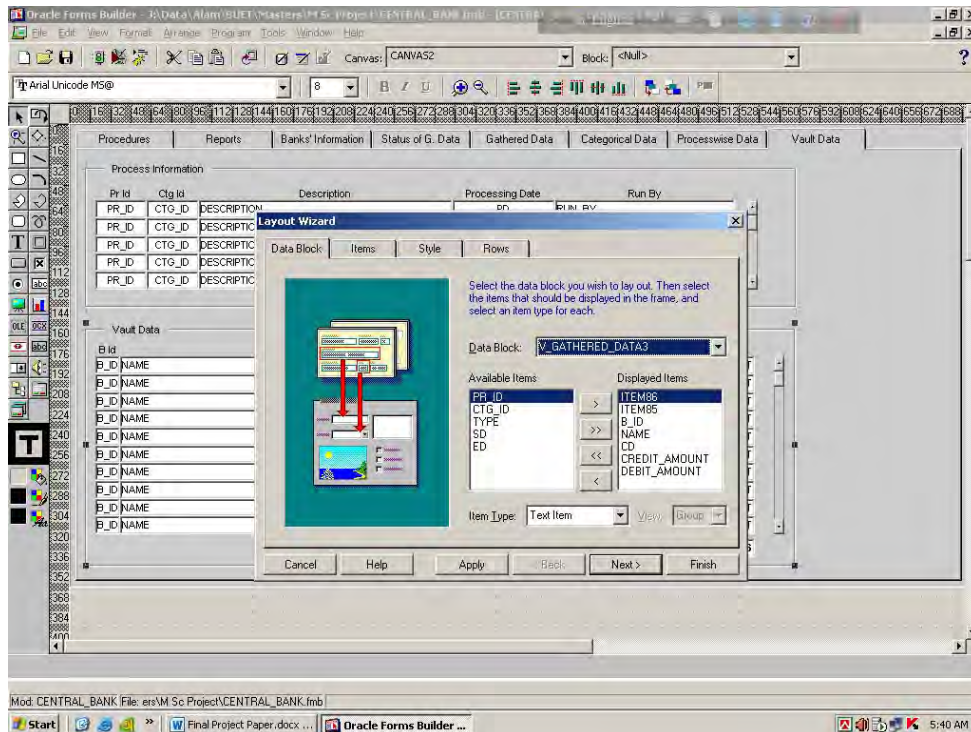


Fig. 7.4: Data Block Wizard (Master-Detail)



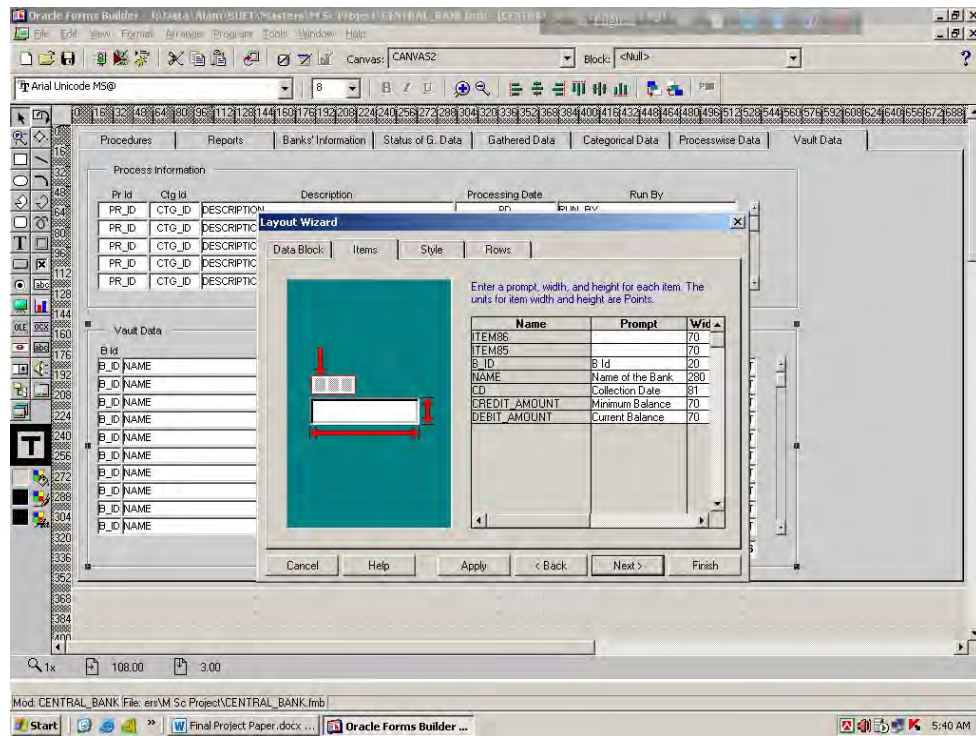
As this view is related with V_PROCESS_INFO view, we will make a master-detail relationship here by the primary key of V_PROCESS_INFO view with the foreign key of V_GATHERED_DATA view (Fig. 7.4). After creating the data blocks we will design the layouts of the blocks by the layout wizard. First step of the layout wizard will be look like the following one (Fig. 7.5). And we will select the fields we want to see on the form.

Fig. 7.5: Layout Wizard (Data Block)



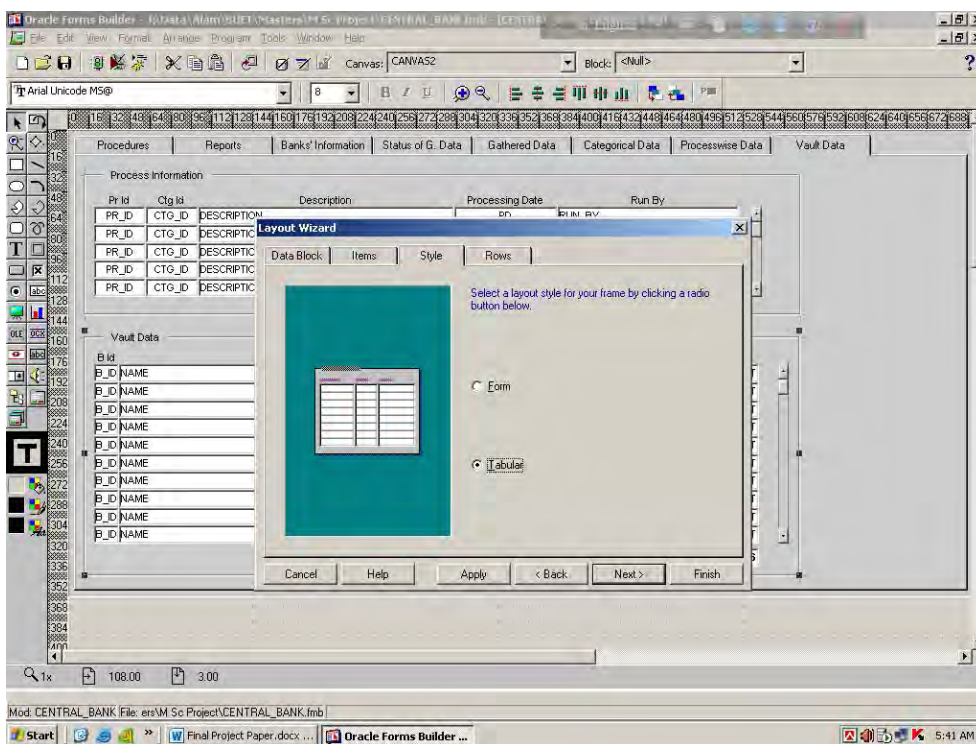
From the following option (Fig. 7.6) we will customize the name and width of the text boxes as required.

Fig. 7.6: Layout Wizard (Items)



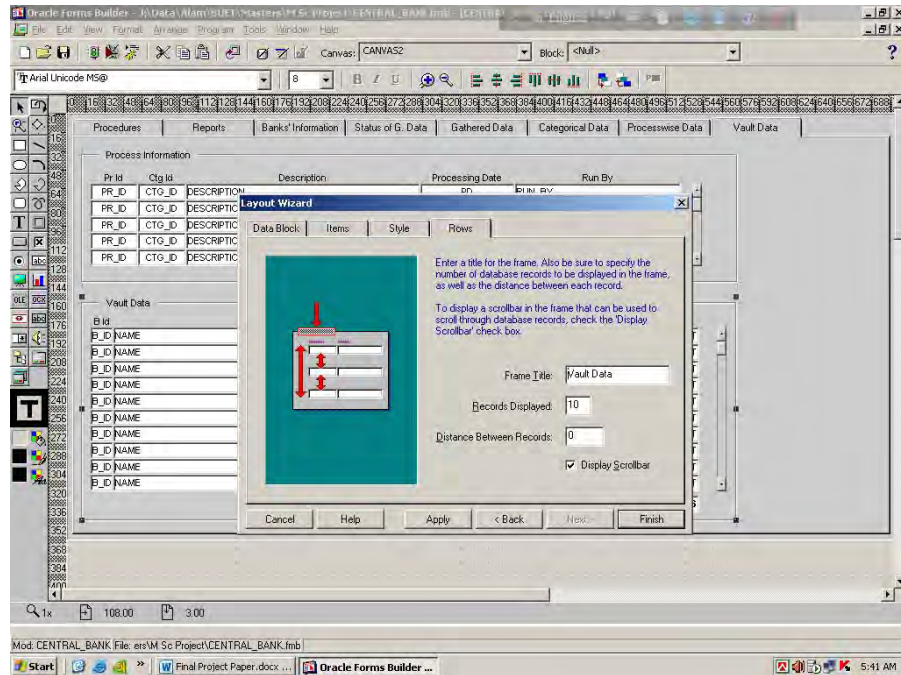
Next, we will select the style of the form. Here we need to design it as tabular format. 'Tabular' option is selected here (Fig. 7.7).

Fig. 7.7: Layout Wizard (Style)



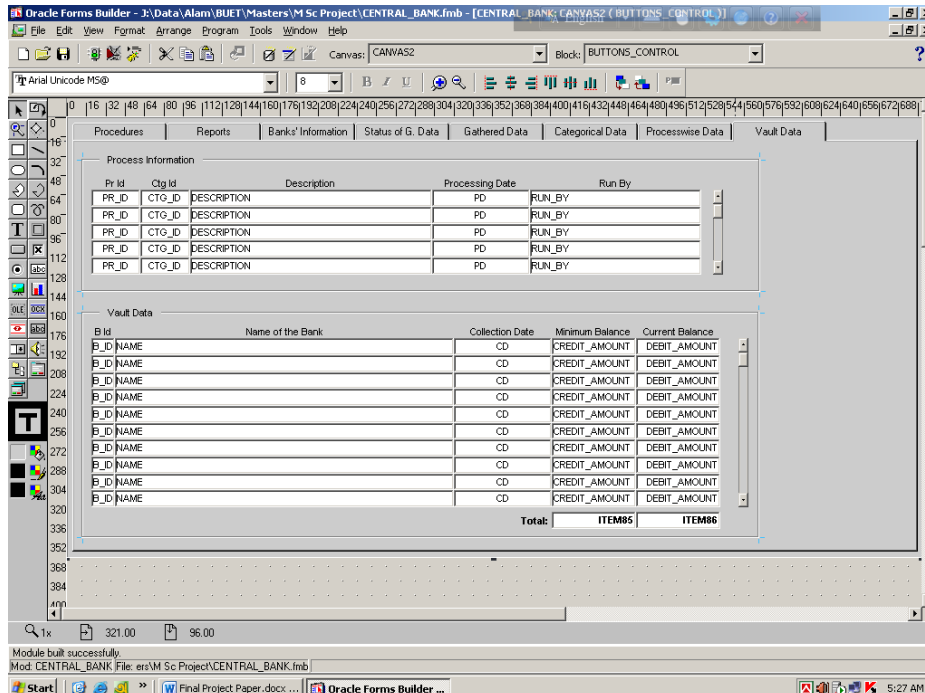
Finally, we wanted to see ten records at a time. So we have put the value 10 for the box ‘record displayed’. We also put a frame title to show the data heading and selected scroll bar option so that we can scroll our records as we wish (Fig. 7.8).

Fig. 7.8: Layout Wizard (Rows)



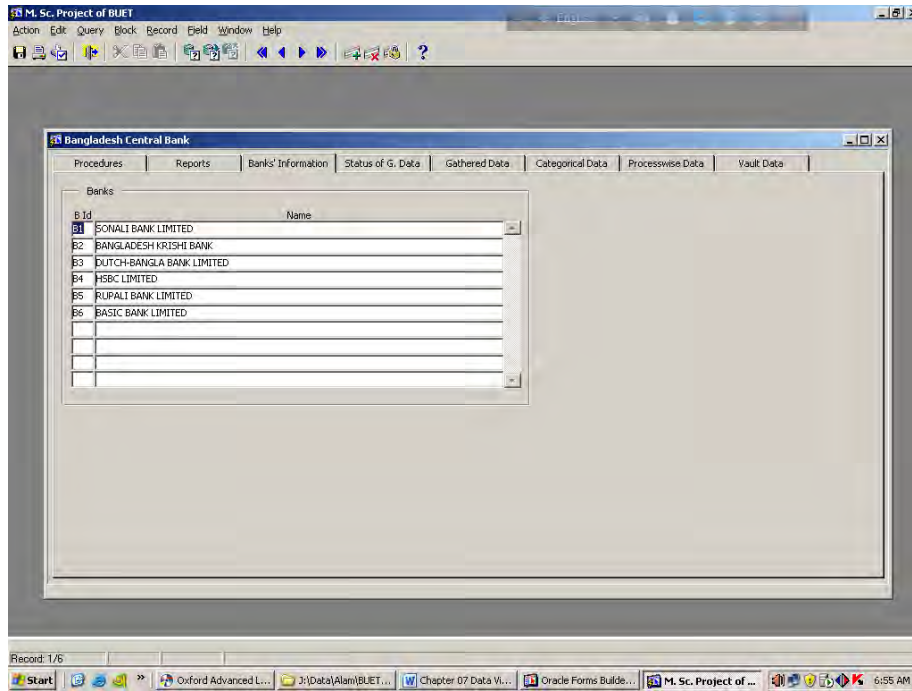
After pressing the ‘Finish’ button the form will be shown like the following one (Fig. 7.9).

Fig. 7.9: Form Layout



After running the form, list of all banks will be populated as Fig. 7.12 shown below.

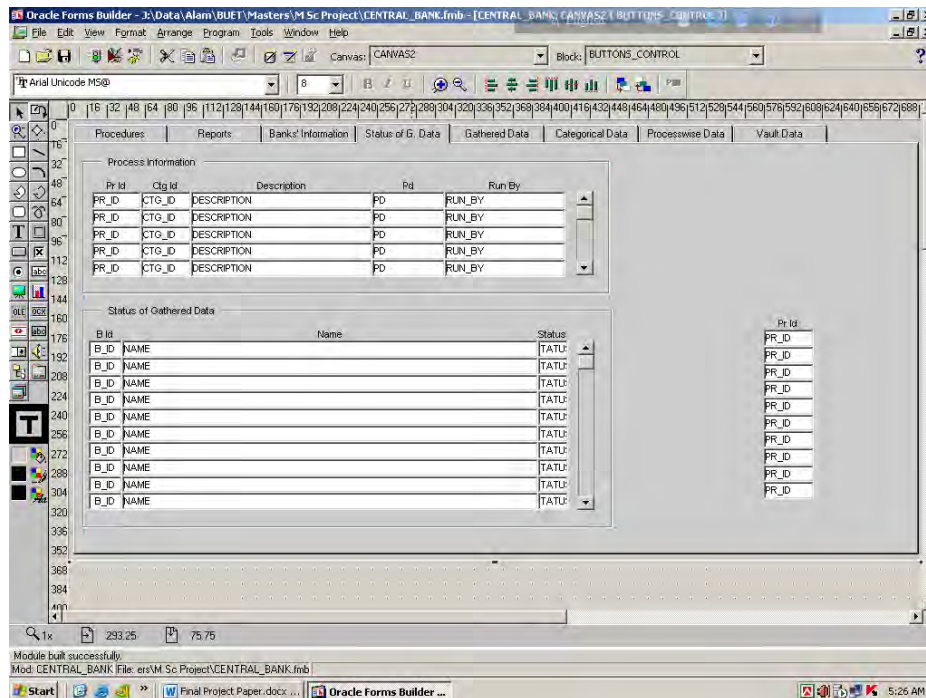
Fig. 7.12: Form Runtime (List of Banks)



7.2.3 Status of Gathered Data

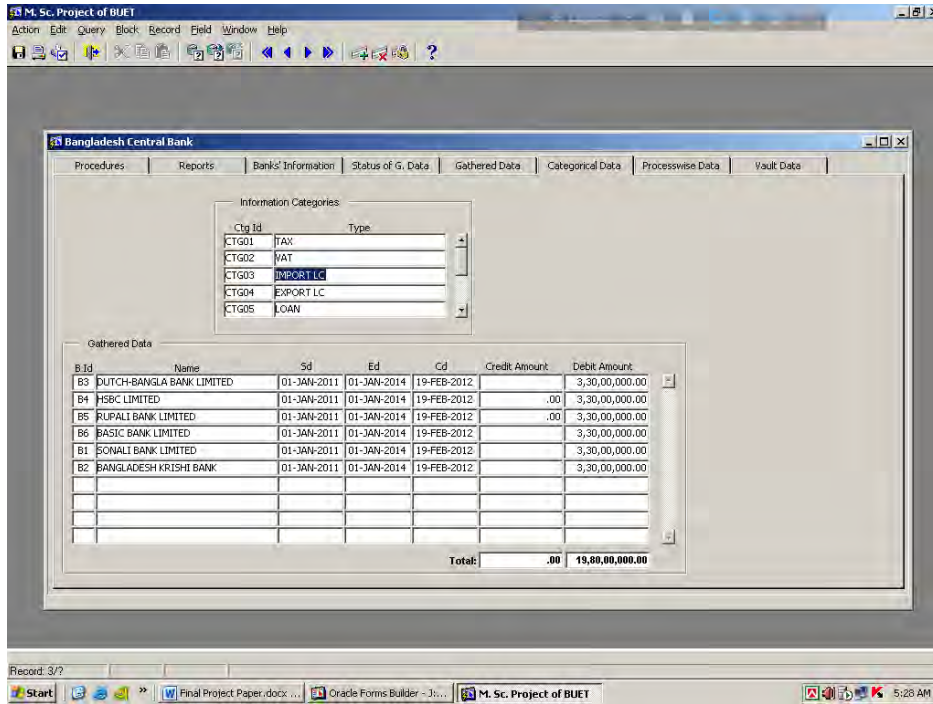
This form will show process wise gathered data. By this form we can be ensured that required data of all banks are gathered accurately for future use (Fig. 7.13).

Fig. 7.13: Form Layout (Status of Gathered Data)



After running the form, gathered data by category will be populated as Fig. 7.18 shown below.

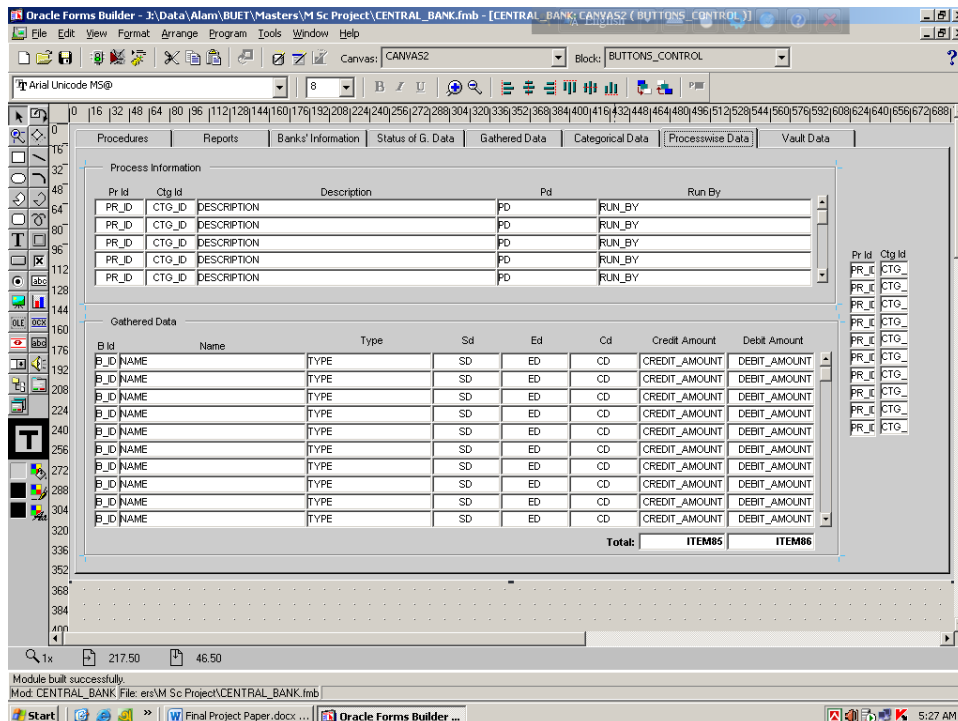
Fig. 7.18: Form Runtime (Gathered Data by Category)



7.2.6 Process Wise Data

This form will show all gathered data by process run by the users (Fig. 7.19).

Fig. 7.19: Form Layout (Gathered Data by Process)



After running the form, gathered data by category will be populated as fig shown below (Fig. 7.20).

Fig. 7.20: Form Runtime (Gathered Data by Process)

The screenshot shows the 'Bangladesh Central Bank' form runtime. The 'Gathered Data' tab is active, displaying two tables. The first table, 'Process Information', lists processes with columns for Pr Id, Ctg Id, Description, Pd, and Run By. The second table, 'Gathered Data', lists bank transactions with columns for B.Id, Name, Type, Sd, Ed, Cd, Credit Amount, and Debit Amount. A total row at the bottom of the second table shows a credit amount of 6,34,294.40 and a debit amount of .00.

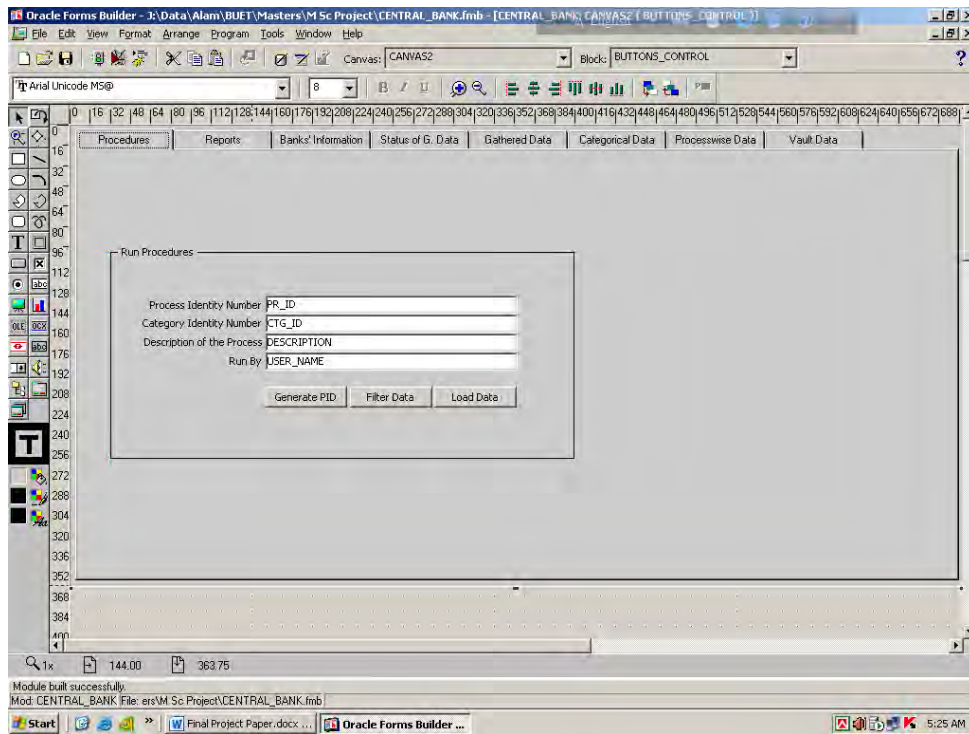
Pr Id	Ctg Id	Description	Pd	Run By
PR001	CTG01	A	19-FEB-2012	A
PR002	CTG02	B	19-FEB-2012	B
PR003	CTG03	O	19-FEB-2012	O
PR004	CTG04	Z	19-FEB-2012	Z
PR005	CTG08	CG	19-FEB-2012	REST

B.Id	Name	Type	Sd	Ed	Cd	Credit Amount	Debit Amount
B3	DUTCH-BANGLA BANK LIMITED	TAX	01-JAN-2011	01-JAN-2014	19-FEB-2012	1,81,269.80	
B4	HSBC LIMITED	TAX	01-JAN-2011	01-JAN-2014	19-FEB-2012	90,585.00	.00
B5	RUPALI BANK LIMITED	TAX	01-JAN-2011	01-JAN-2014	19-FEB-2012	90,634.90	.00
B6	BASIC BANK LIMITED	TAX	01-JAN-2011	01-JAN-2014	19-FEB-2012	90,634.90	
B1	SONALI BANK LIMITED	TAX	01-JAN-2011	01-JAN-2014	19-FEB-2012	90,534.90	
B2	BANGLADESH KRISHI BANK	TAX	01-JAN-2011	01-JAN-2014	19-FEB-2012	90,634.90	
Total:						6,34,294.40	.00

7.2.7 Running a Process to Filter and Load Data

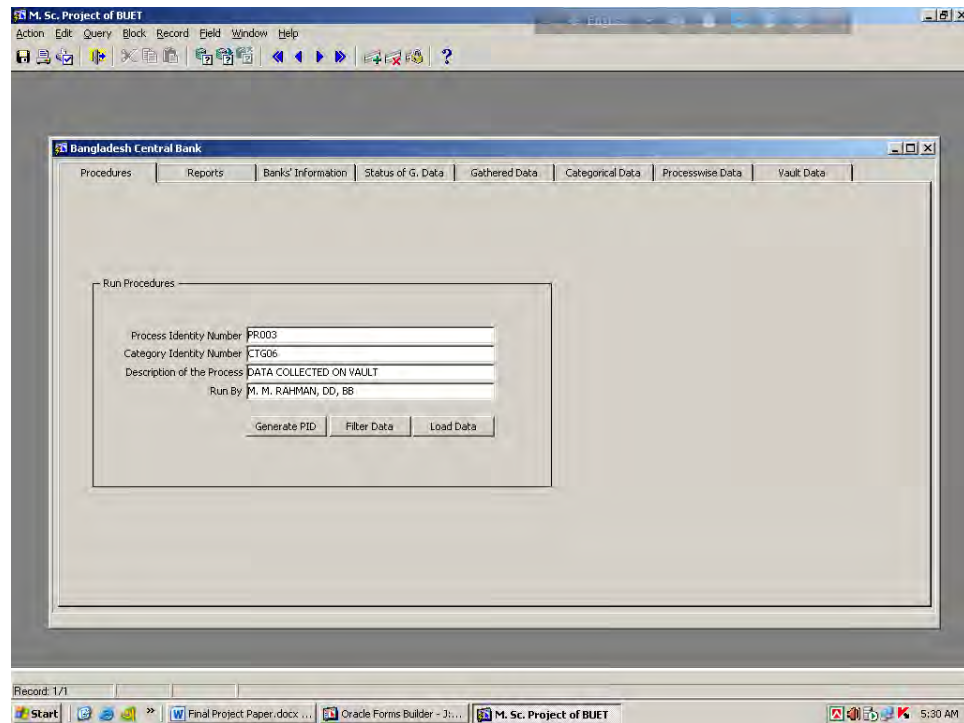
After receiving data from linked-server to the temporary schema we have to check that all data are reached successfully. By pushing the 'Filter Data' button we can check whether all data are available or not. Then we will capture the data and load it to the permanent schema by pressing the button 'Load Data' (Fig. 7.21 and 7.22).

Fig. 7.21: Form Layout (Filter and Load Data)



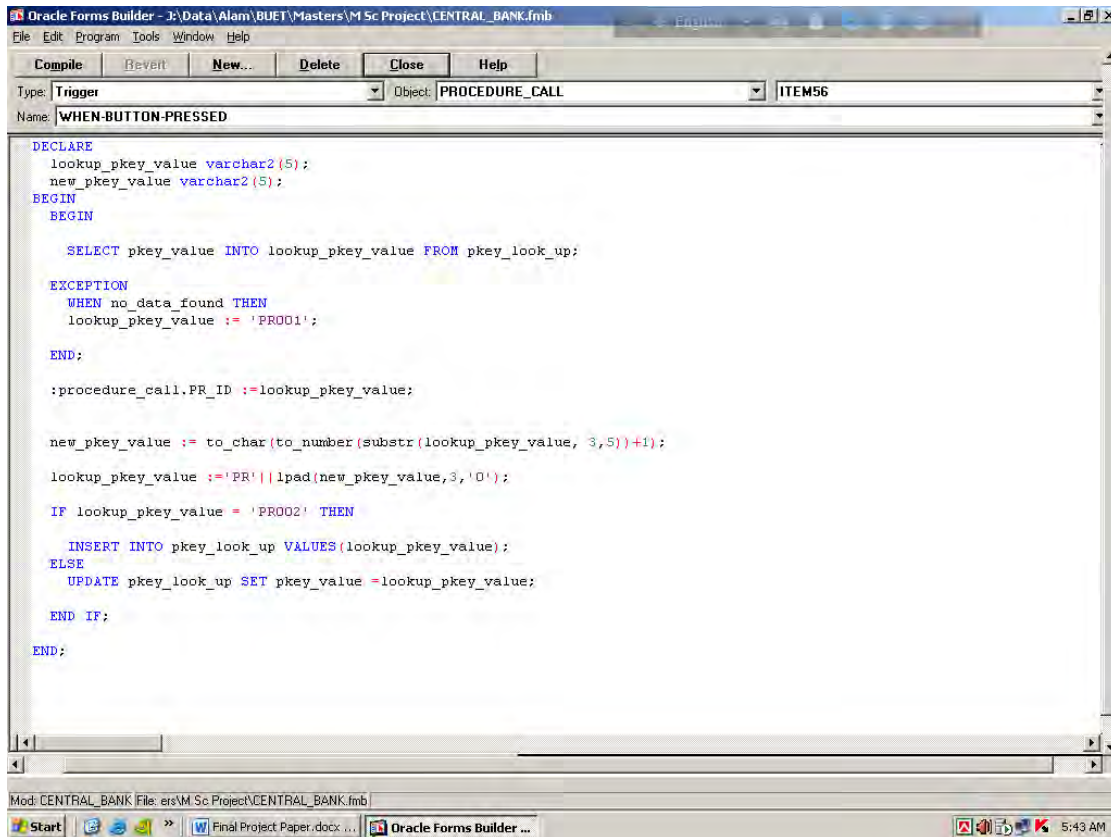
After running the form, it will look like as Fig. 7.22 shown below.

Fig. 7.22: Form Runtime (Filter and Load Data)



In the above form there is another button named 'Generate PID'. This button will produce a unique Process Identity Number for the integrity of the data. This process ID is very important to trace the data arrival process. The following procedure is written for this purpose (Fig. 7.23).

Fig. 7.23: Procedure to Generate PID



```

Oracle Forms Builder - E:\Data\Alam\BUET\Masters\M Sc Project\CENTRAL_BANK.fmb
File Edit Program Tools Window Help
Compile Revert New... Delete Close Help
Type: Trigger Object: PROCEDURE_CALL ITEM56
Name: WHEN-BUTTON-PRESSED

DECLARE
  lookup_pkey_value varchar2(5);
  new_pkey_value varchar2(5);
BEGIN
  BEGIN
    SELECT pkey_value INTO lookup_pkey_value FROM pkey_look_up;

  EXCEPTION
    WHEN no_data_found THEN
      lookup_pkey_value := 'PROO1';

  END;

  :procedure_call.PR_ID :=lookup_pkey_value;

  new_pkey_value := to_char(to_number(substr(lookup_pkey_value, 3,5))+1);
  lookup_pkey_value := 'PR' || lpad(new_pkey_value,3,'0');

  IF lookup_pkey_value = 'PROO2' THEN
    INSERT INTO pkey_look_up VALUES(lookup_pkey_value);
  ELSE
    UPDATE pkey_look_up SET pkey_value =lookup_pkey_value;

  END IF;

END;

```

Mod: CENTRAL_BANK File: ers\M Sc Project\CENTRAL_BANK.fmb
 Start Final Project Paper.docx Oracle Forms Builder ... 5:43 AM

The above procedure will also require a dummy table named LOOK_UP, which is created as follows:

```

CREATE TABLE PKEY_LOOK_UP
(PKEY_VALUE CHAR(5));

```

Procedure for 'Filter Data' button is written as follows (Fig. 7.24).

Fig. 7.24: Procedure for Filter Data Button

```

begin
  DECLARE
    alert_button NUMBER;
  begin
    if :procedure_call.PR_ID is null or :procedure_call.CTG_ID is null
      or :procedure_call.DESRIPTION is null or :procedure_call.USER_NAME is null then
      alert_button := Show_Alert('MESSAGE');
    else
      FILTER_DATA (:procedure_call.PR_ID, :procedure_call.CTG_ID, :procedure_call.DESRIPTION, :procedure_call.USER_NAME);
      alert_button := Show_Alert('SUCCESSFUL');
      -- :procedure_call.PR_ID := NULL;
      -- :procedure_call.CTG_ID := NULL;
      :procedure_call.DESRIPTION := NULL;
      :procedure_call.USER_NAME := NULL;
    end if;
  end;
end;

```

Procedure for 'Load Data' button is written as follows (Fig. 7.25).

Fig. 7.25: Procedure for Load Data Button

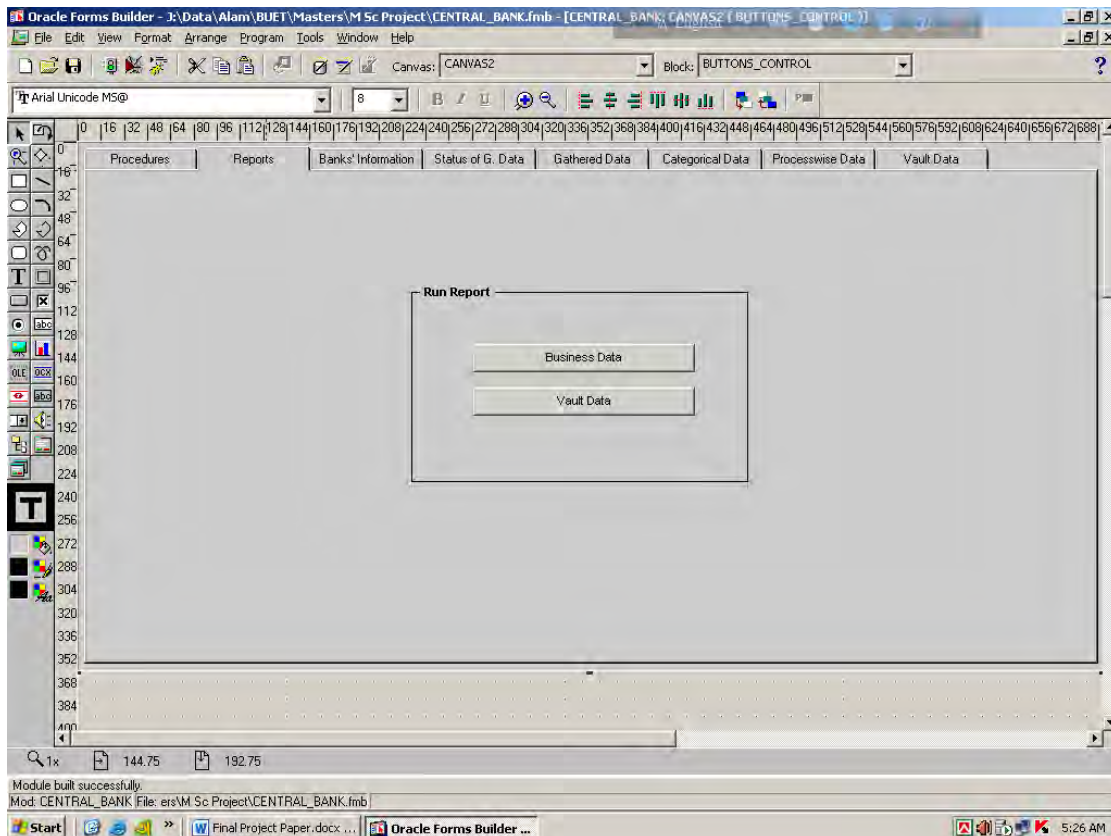
```

begin
  DECLARE
    alert_button NUMBER;
  begin
    if :procedure_call.PR_ID is null OR :procedure_call.CTG_ID IS NULL then
      alert_button := Show_Alert('MESSAGE');
    else
      FINALLY_LOAD_DATA (:procedure_call.PR_ID, :procedure_call.CTG_ID);
      alert_button := Show_Alert('SUCCESSFUL');
      :procedure_call.PR_ID := NULL;
      :procedure_call.CTG_ID := NULL;
      :procedure_call.DESRIPTION := NULL;
      :procedure_call.USER_NAME := NULL;
    end if;
  end;
end;

```

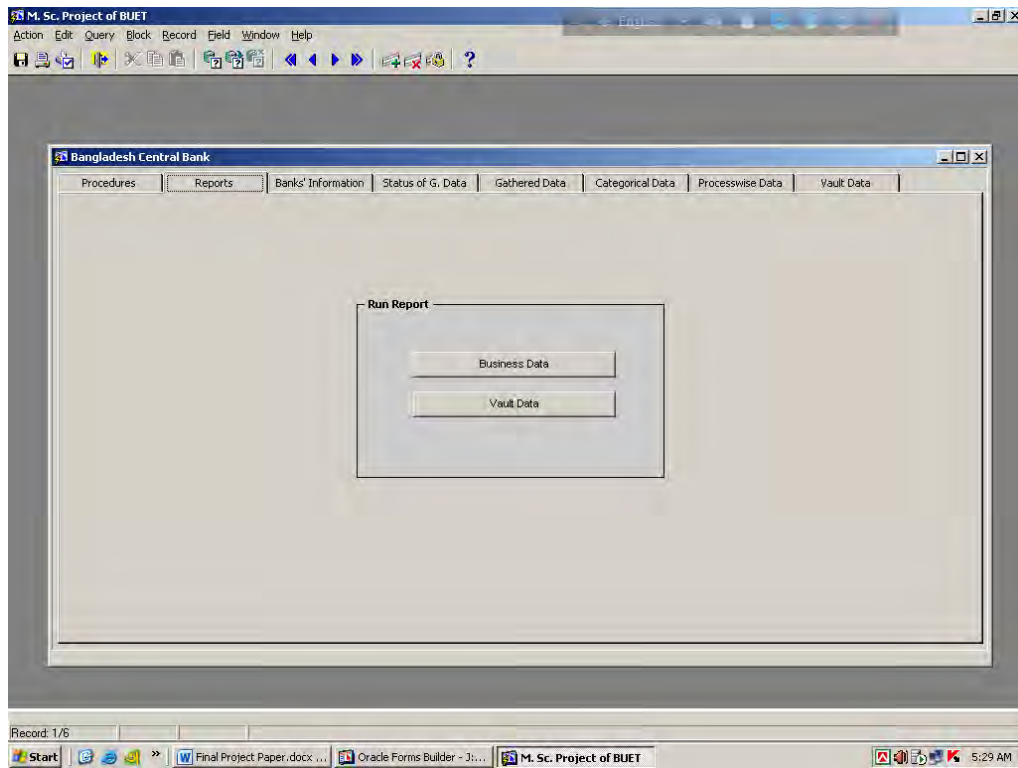

Finally, the following form is designed to call the necessary reports of the project. Here two buttons, 'Business Data' and 'Vault Data', are used to call the reports. All data related to tax, vat, export, import, etc. will be shown by the 'Business Data' report and liquid asset by the vault data report. Reports tab of the following form (Fig. 7.26) is designed to call the reports. Necessary programs are written for this purpose.

Fig. 7.26: Form Layout (Run Report)



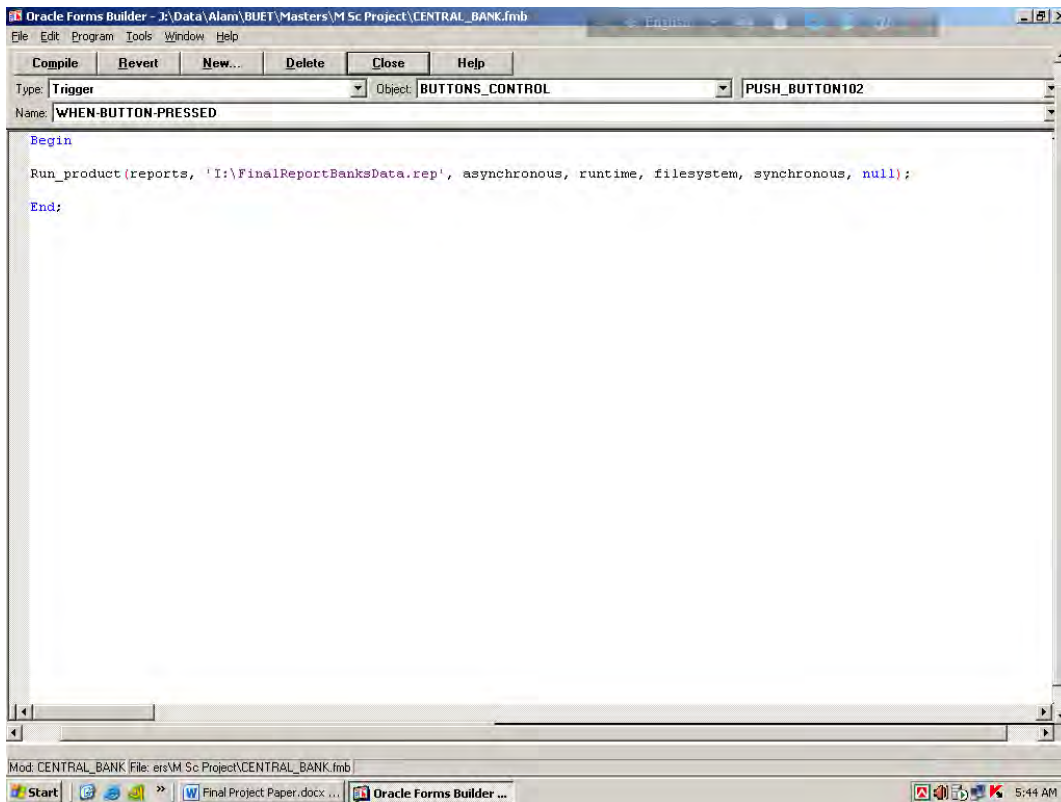
After running the form, it will look like as fig shown below (Fig. 7.27).

Fig. 7.27: Form Runtime (Run Report)



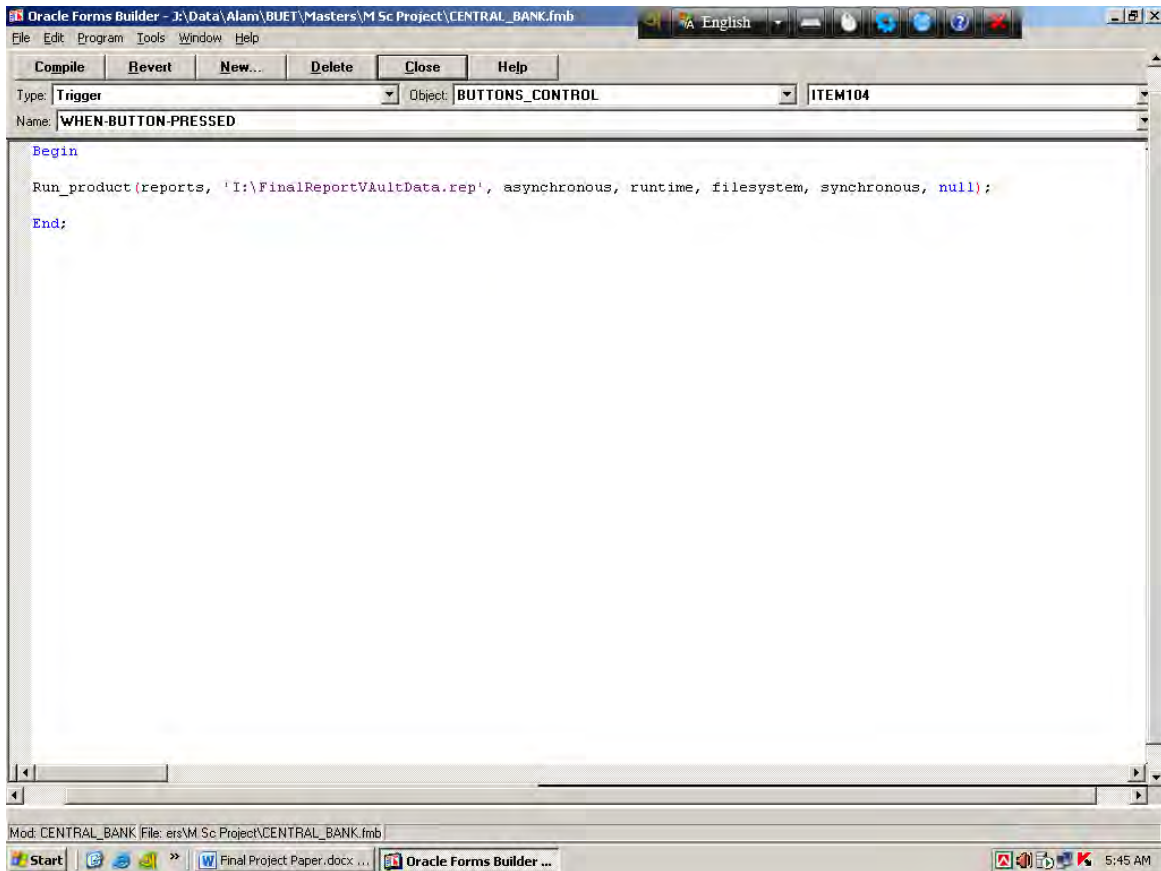
To call business reports following procedure will be used as shown below (Fig. 7.28).

Fig. 7.28: Procedure to Call Business Reports



To call liquid asset's report following procedure will be used as shown below (Fig. 7.29).

Fig. 7.29: Procedure to Call Liquid Asset's Reports

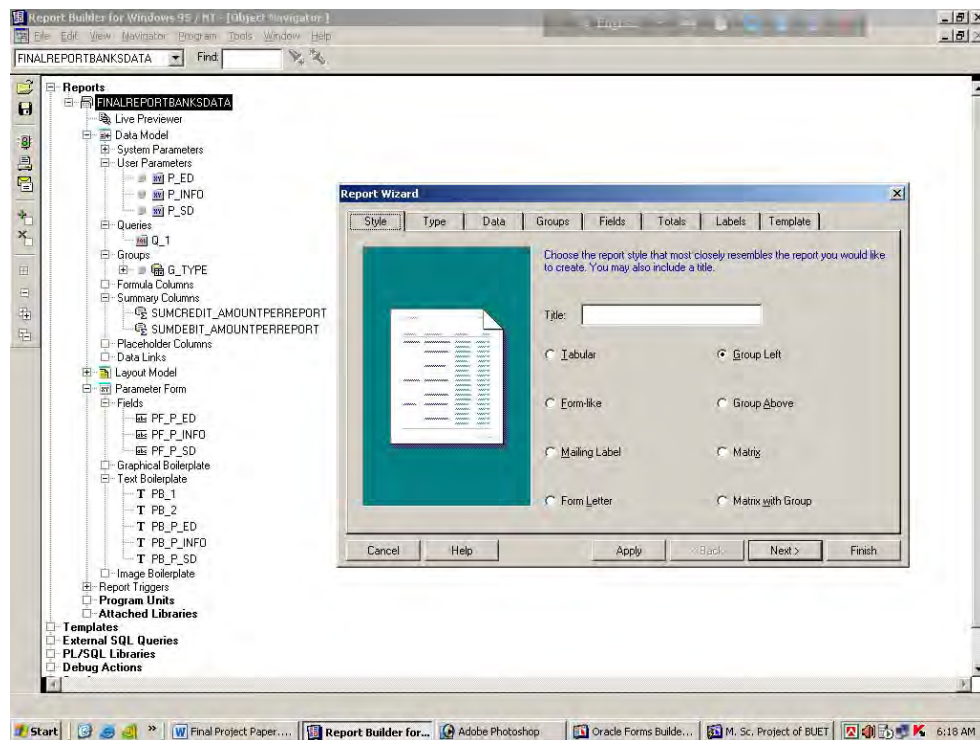


7.3 Report Design

Report is the ultimate state of data. In report, data stored in the database are represented as information. From management point of view, report is very important. Report presents status, progress, performance, etc., as per business needs. In this project we used permanent schema of the production server to run the reports.

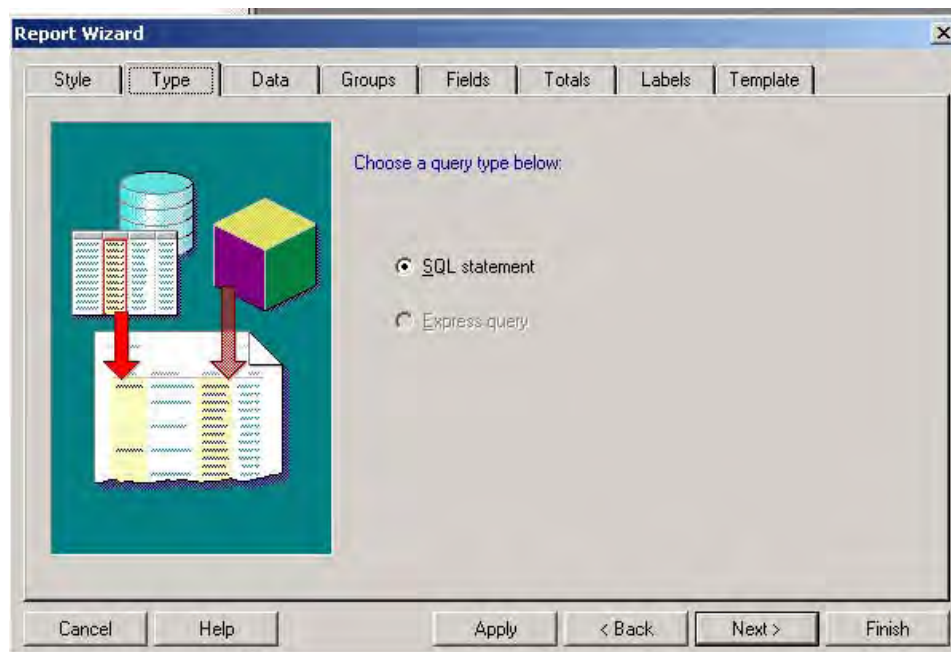
Here we used Oracle Report Design utility of Oracle Developer 6i. To design the business reports, at first we created a new module and added features to it. At first, we selected report type. In our project, we used 'Group Left' option of the Style tab from Report Wizard as follows (Fig. 7.30).

Fig. 7.30: Report Wizard (Style Tab)



Then we selected SQL Statement from 'Type' tab to indicate that data will be gathered by SQL commands, not by express query as follows (Fig. 7.31).

Fig. 7.31: Report Wizard (Type Tab)



We used the following query to gather data for these reports as shown in Fig. 7.32.

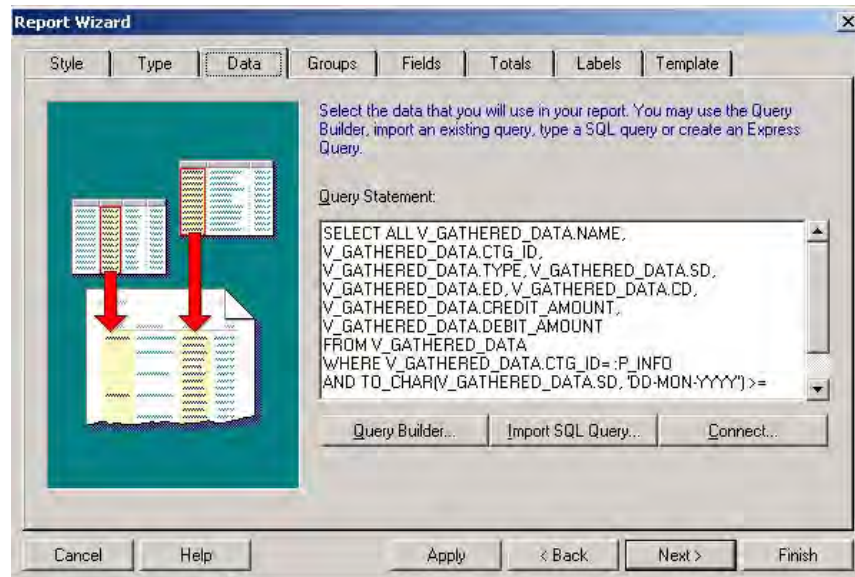
```
SELECT ALL V_GATHERED_DATA.NAME, V_GATHERED_DATA.CTG_ID,
```

```

V_GATHERED_DATA.TYPE, V_GATHERED_DATA.SD, V_GATHERED_DATA.ED, V_GATHERED_DATA.CD,
V_GATHERED_DATA.CREDIT_AMOUNT, V_GATHERED_DATA.DEBIT_AMOUNT
FROM V_GATHERED_DATA
WHERE V_GATHERED_DATA.CTG_ID= :P_INFO
AND TO_CHAR(V_GATHERED_DATA.SD, 'DD-MON-YYYY') >= :P_SD
AND TO_CHAR(V_GATHERED_DATA.ED, 'DD-MON-YYYY') <= :P_ED

```

Fig. 7.32: Report Wizard (Data Tab)



At this stage, we selected the fields that will be shown in the report and also the group field by which we will make groups of our data. Here group means categories of data, like, tax, vat, import and export, etc. (Fig. 7.33 and 7.34).

Fig. 7.33: Report Wizard (Group Tab)

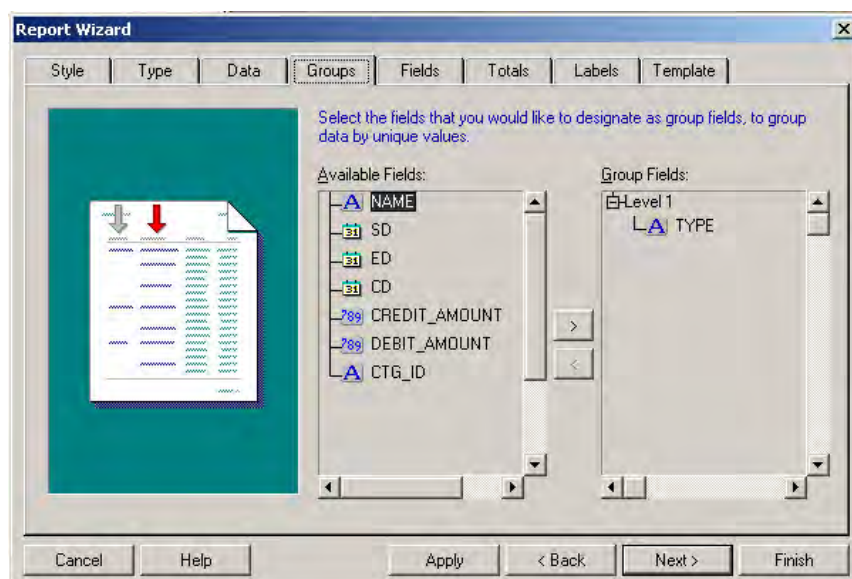
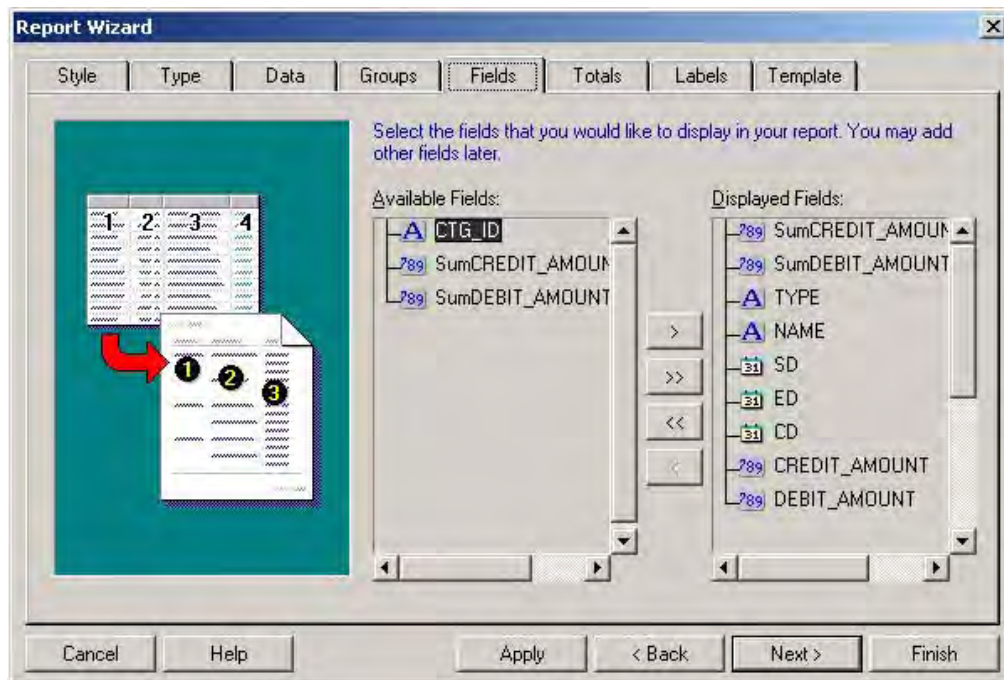
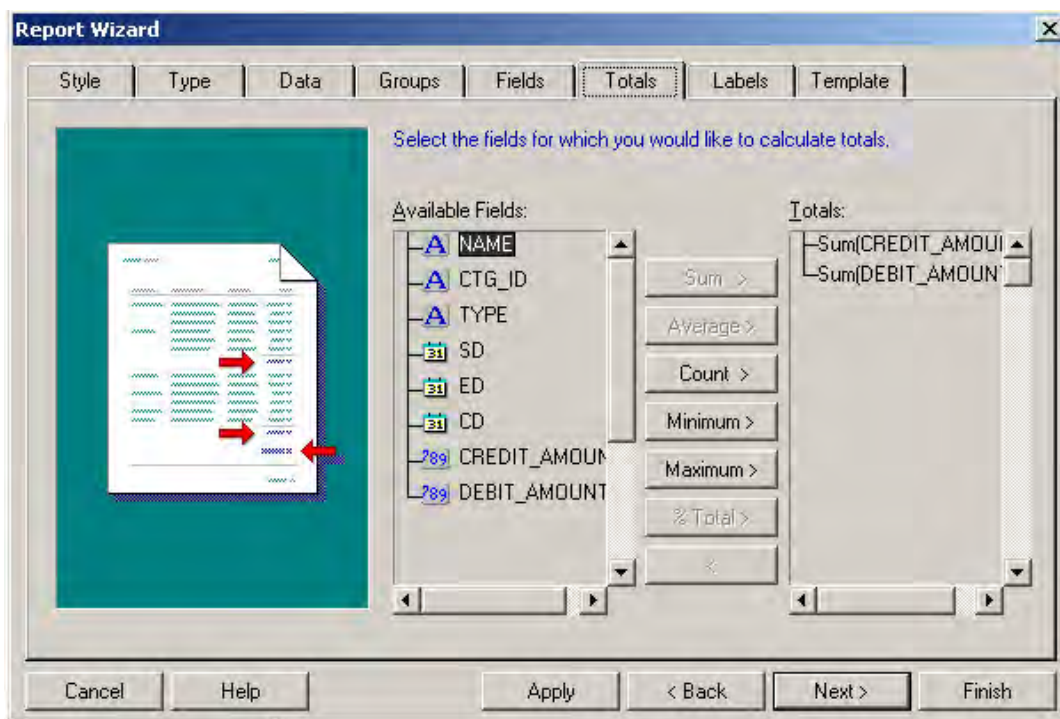


Fig. 7.34: Report Wizard (Fields Tab)



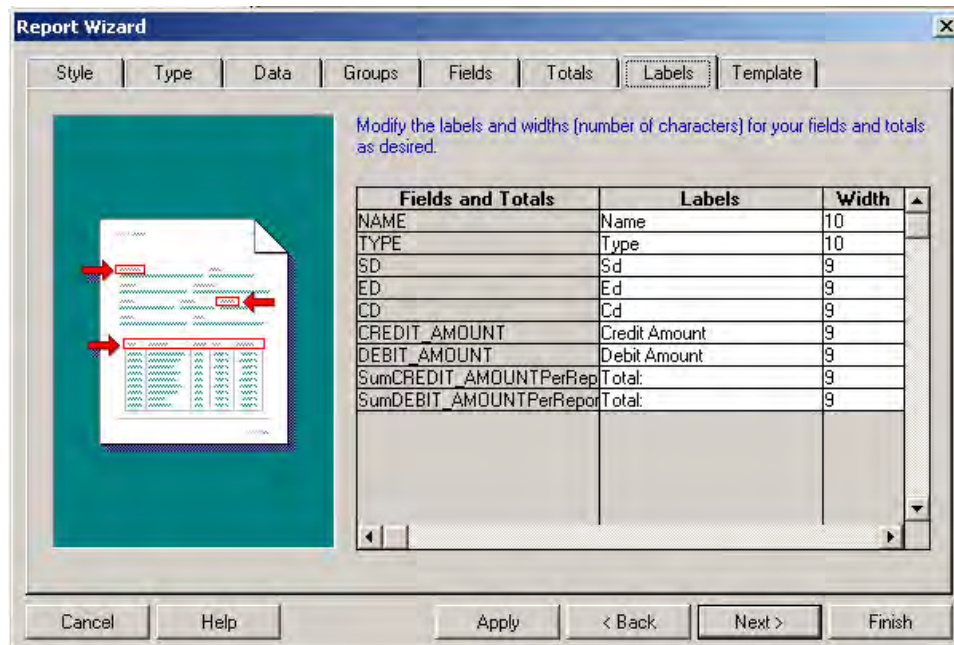
At this stage, we selected summary fields to show the sum of the data fields (country as a whole) for all banks as follows (Fig. 7.35).

Fig. 7.35: Report Wizard (Totals Tab)



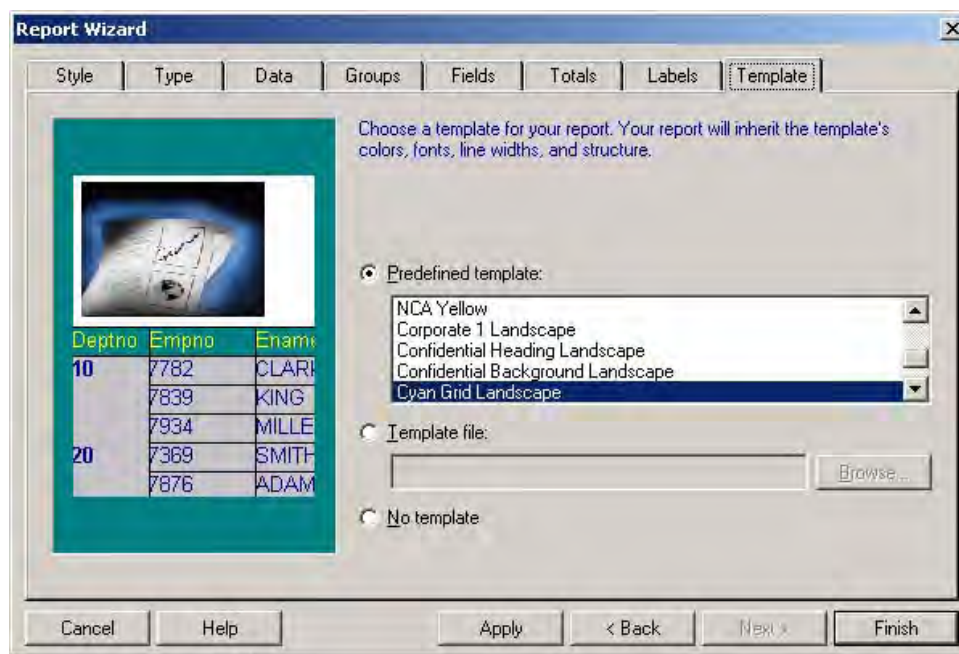
In the following figure (Fig. 7.36), we selected report headings and width of the data fields as follows.

Fig. 7.36: Report Wizard (Labels Tab)



Finally, we selected the template of our report, i.e., the layout of the data as follows and pressed 'Finish' button to complete the report design as follows (Fig. 7.37)

Fig. 7.37: Report Wizard (Template Tab)



After creation of the report, Data Model and Layout Model will be seen as Fig. 7.38 and 7.39 respectively.

Fig. 7.38: Data Model

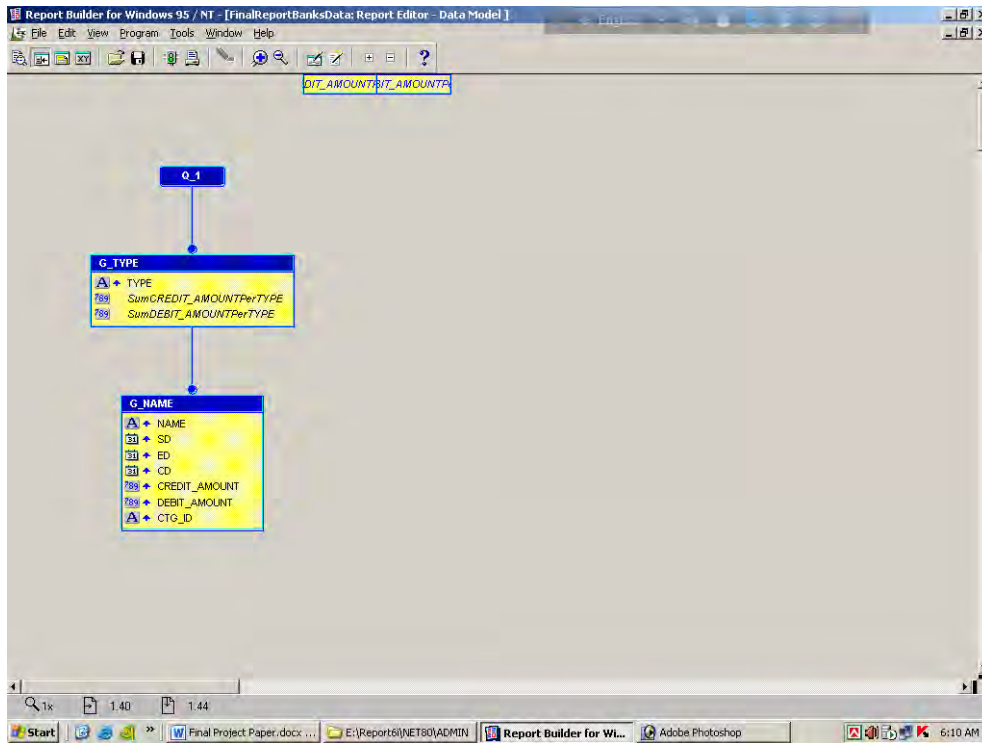
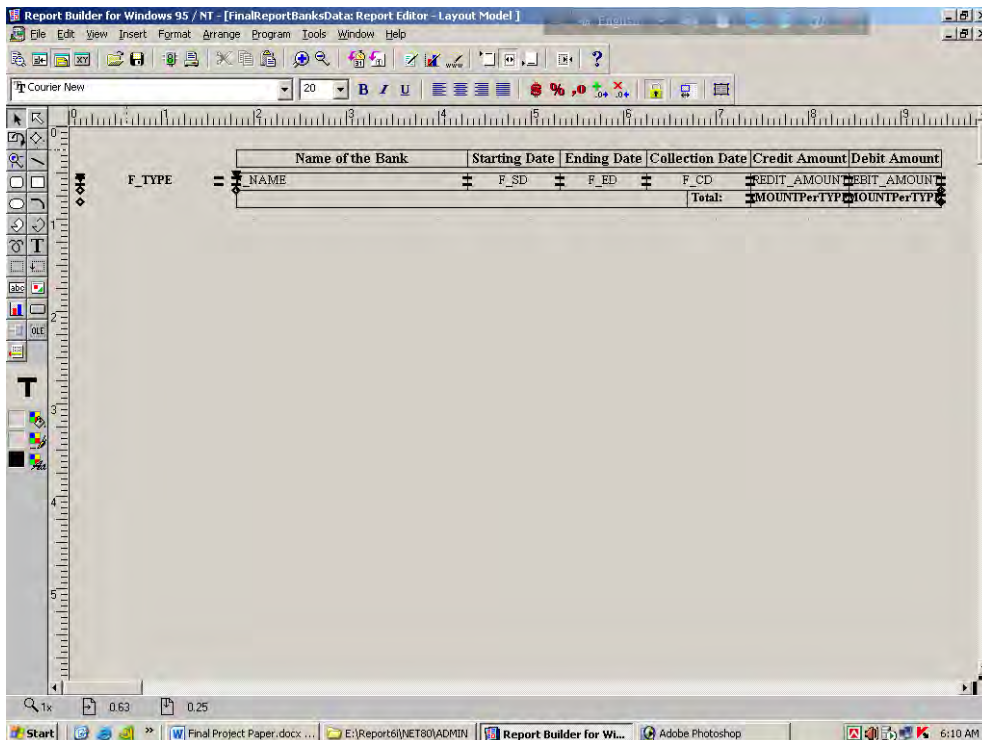
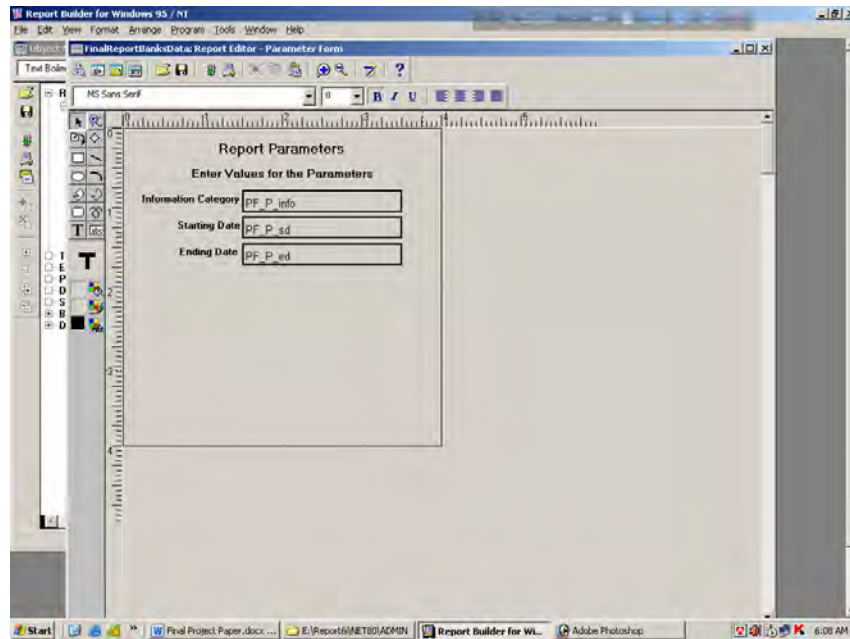


Fig. 7.39: Layout Model



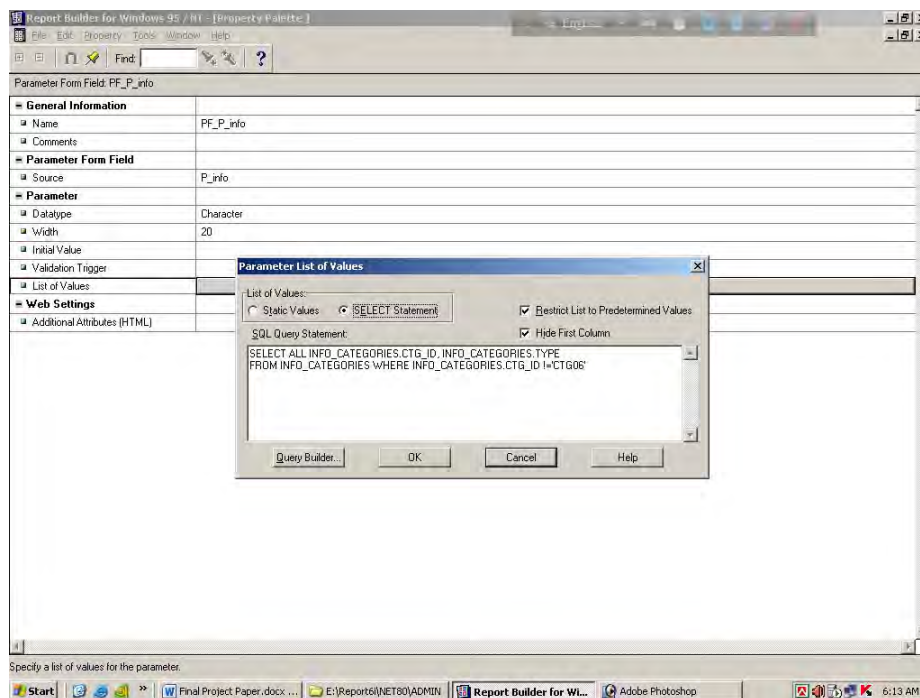
As we have designed a business report to show all categories of data, we have to pass some parameters to filter our data before running the report. Here, we will pass three parameters: information category, starting date and ending date. Starting and ending date will impose the date range for which data will be produced. Parameter form is designed as follows (Fig. 7.40).

Fig. 7.40: Report Parameter Form



To see the Information Category automatically in the parameter list, we passed SQL query into the List of Values Option of the report as follows (Fig. 7.41).

Fig. 7.41: Report LOV



When we will call the Business Reports from our Form designed in section 7.2, parameter box will be appeared as follows (Fig. 7.42).

Fig. 7.42: Report Parameters

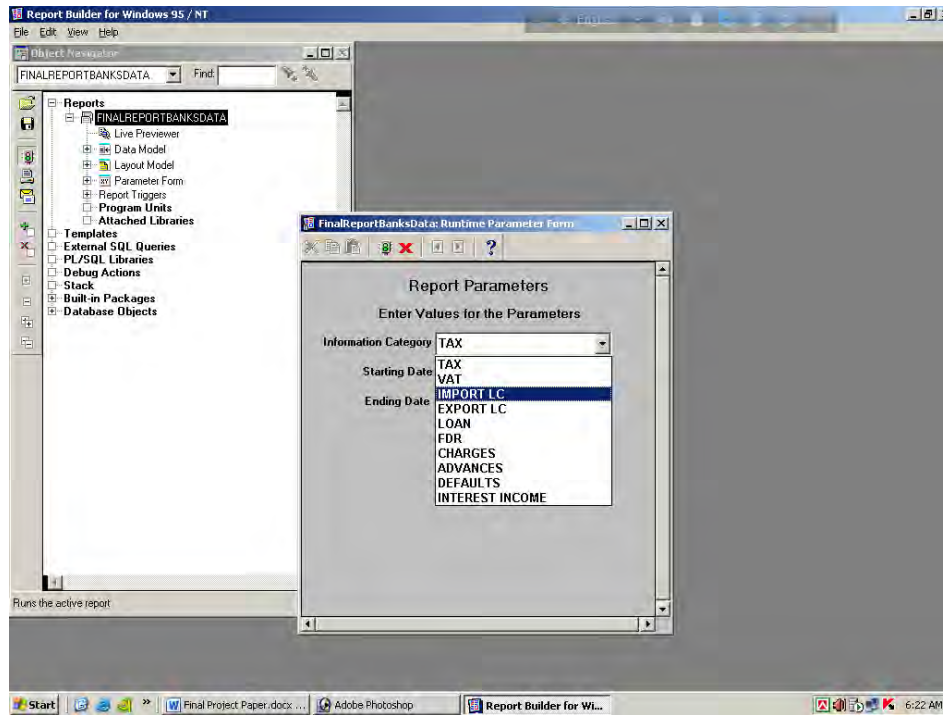
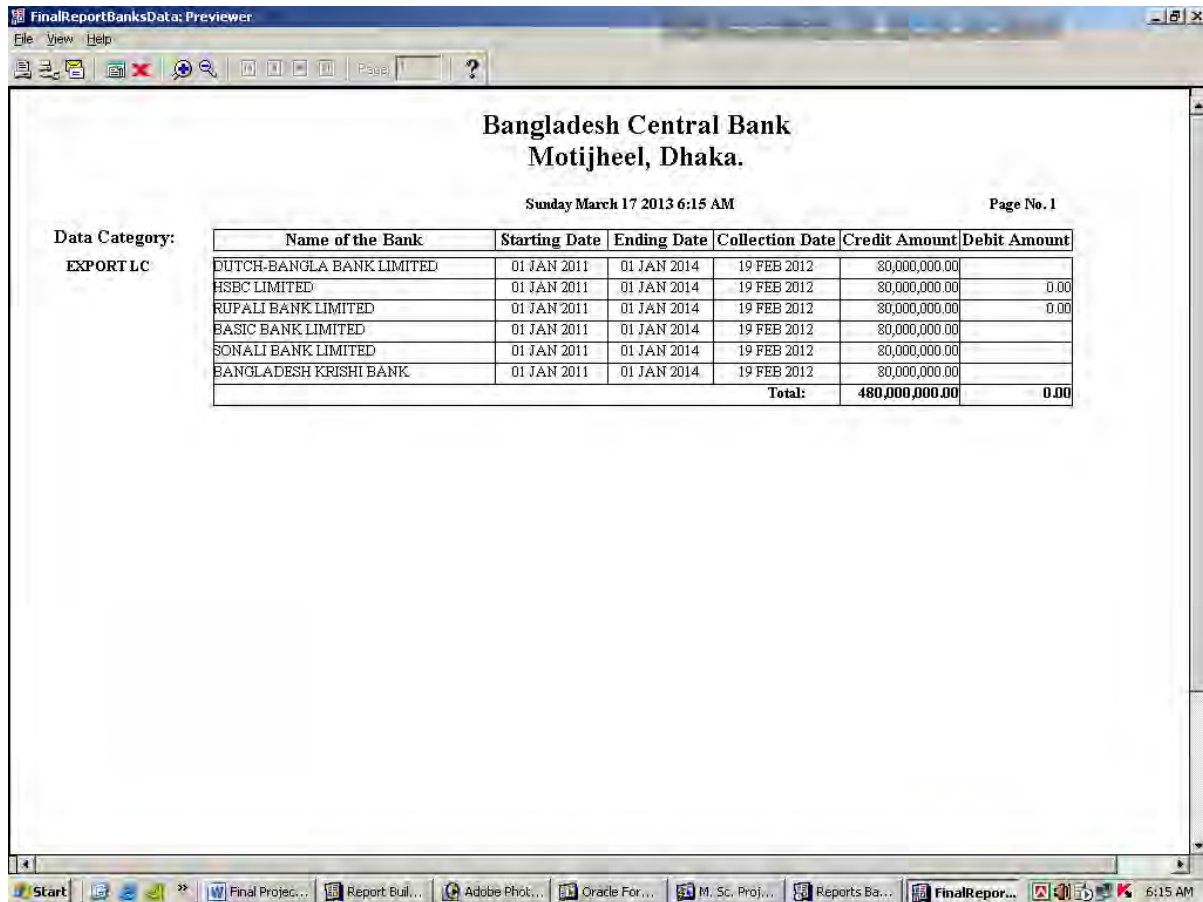


Fig. 7.43: Final Report (Run Time)

Bangladesh Central Bank Motijheel, Dhaka.						
Sunday March 17 2013 6:59 AM						
Page No. 1						
Data Category:	Name of the Bank	Starting Date	Ending Date	Collection Date	Credit Amount	Debit Amount
TAX	DUTCH-BANGLA BANK LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	181,269.80	
	HSBC LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	90,583.00	0.00
	RUPALI BANK LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	90,634.90	0.00
	BASIC BANK LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	90,634.90	
	SONALI BANK LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	90,534.90	
	BANGLADESH KRISHI BANK	01 JAN 2011	01 JAN 2014	19 FEB 2012	90,634.90	
	Total:				634,294.40	0.00

After giving the parameters properly, we will see the ultimate report as below. Here, we wanted to see the Tax and Export data from '01-Jan-2011' to '01-Jan-2014'. These are shown in Fig. 7.43 and 7.44.

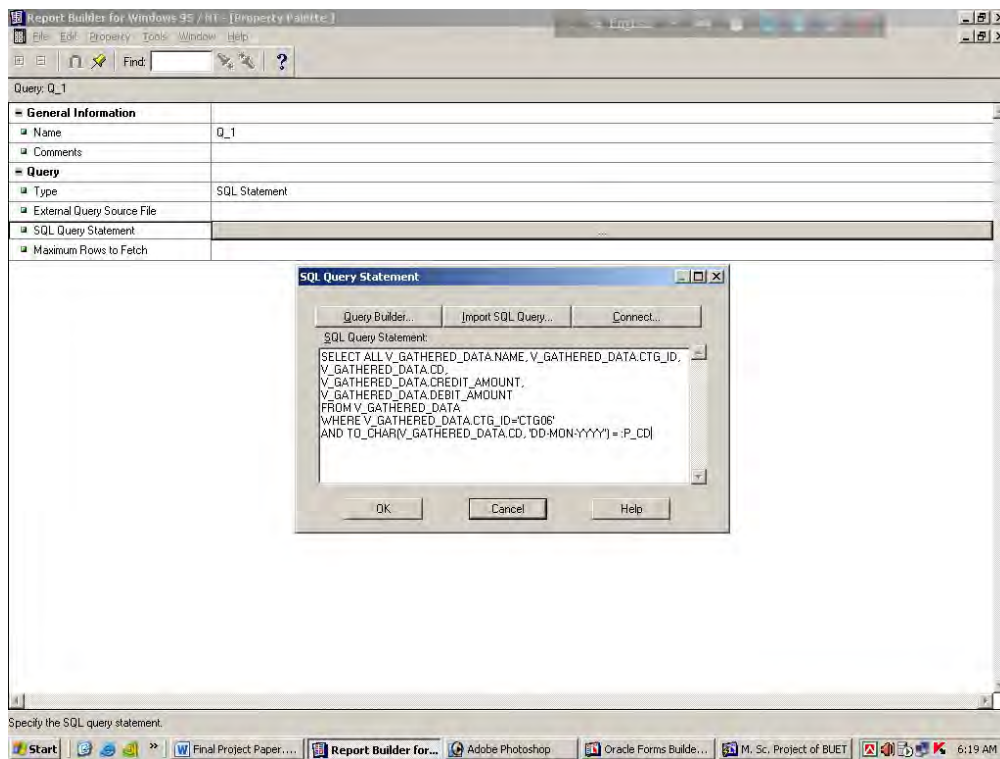
Fig. 7.44: Final Report (Run Time)



Bangladesh Central Bank Motijheel, Dhaka.						
Data Category:				Sunday March 17 2013 6:15 AM		Page No. 1
EXPORT LC						
Name of the Bank	Starting Date	Ending Date	Collection Date	Credit Amount	Debit Amount	
DUTCH-BANGLA BANK LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	80,000,000.00		
HSEC LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	80,000,000.00	0.00	
RUPALI BANK LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	80,000,000.00	0.00	
BASIC BANK LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	80,000,000.00		
SONALI BANK LIMITED	01 JAN 2011	01 JAN 2014	19 FEB 2012	80,000,000.00		
BANGLADESH KRISHI BANK	01 JAN 2011	01 JAN 2014	19 FEB 2012	80,000,000.00		
Total:				480,000,000.00	0.00	

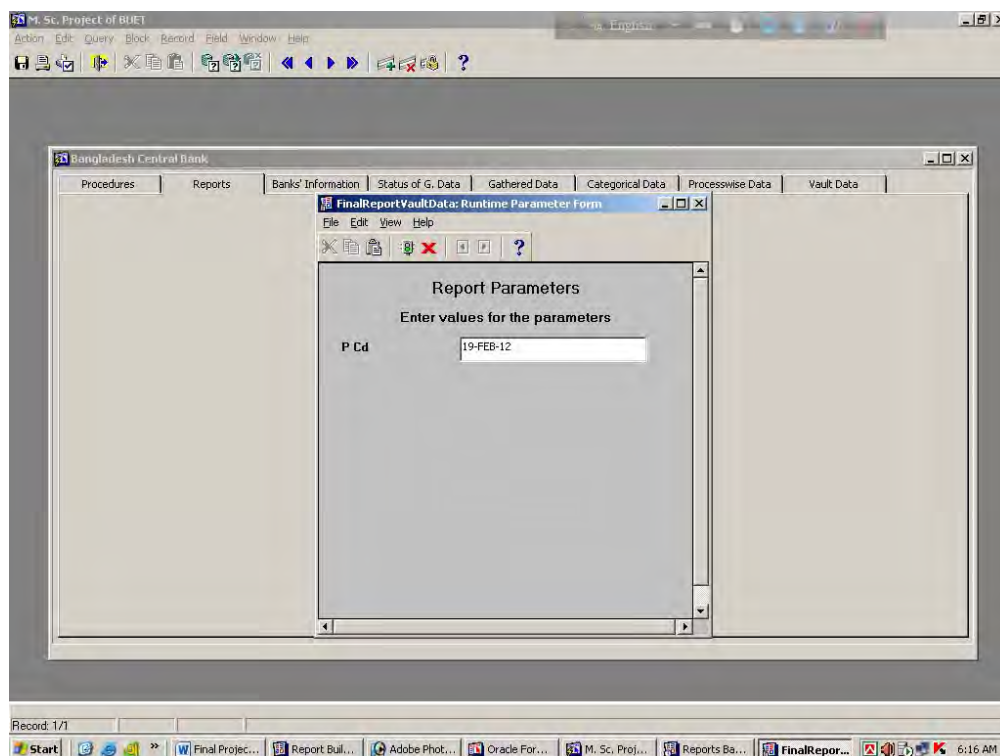
Process to see the 'Liquid Asset' is same as Business Data. Only difference is the data source, format and parameter to be passed. So, we avoided to discuss the redundant process of report design here. However, the following query is passed to gather data for the report (Fig. 7.45).

Fig. 7.45: SQL Query for Liquid Asset Report



When we will call the Vault Report (Liquid Asset) from our Form developed in section 7.2, parameter box will be appeared as follows (Fig. 7.46).

Fig. 7.46: Report Parameters



To see the current status of the vault we need only one parameter, i.e., current date. After passing the parameter, report will be looked like as follows (Fig. 7.47).

Fig. 7.47: Final Report (Run Time)

Central Bank of Bangladesh
Mitijheel, Dhaka.

Vault Status (Liquid Asset)

Sunday March 17 2013 6:16 AM Page No. 1

Name	Collection Date	Minimum Balance	Current Balance
DUTCH-BANGLA BANK LIMITED	19-FEB-12	100,000,000.00	200,000,000.00
HSBC LIMITED	19-FEB-12	55,000,000.00	85,000,000.00
RUPALI BANK LIMITED	19-FEB-12	87,000,000.00	160,000,000.00
BASIC BANK LIMITED	19-FEB-12	50,000,000.00	100,000,000.00
SONALI BANK LIMITED	19-FEB-12	58,000,000.00	85,000,000.00
BANGLADESH KRISHI BANK	19-FEB-12	87,000,000.00	140,000,000.00
Total:		437,000,000.00	770,000,000.00

7.4 Summary

After successful reception of data into the production server of the central bank from different commercial banks, this chapter opens the ways of the use of the final processed data by the ultimate end-users. In this chapter, we developed a set of forms and reports so that users can interact with data and take necessary hard copy print according to their needs. The end-users will also be able to pass parameters to filter the data according to the demand. Necessary procedures and functions that are written according the need of the forms and reports are also discussed here. Step-by-step configuration of report and form tools is also elaborated here. Tools are customized and configured efficiently so that the end-users can use it interactively. Finally, ultimate output is seen by run time report preview before printing.

Chapter 8

Security Issues

8.1 Introduction

As we are gathering data from production servers of different banks, security is a great concern here. We have tried to implement the maximum security features for this purpose. Security is provided in all stages, from banks' database servers to central bank's production server, including middle-tier linked-server also.

8.2 Network Domain Configuration

For network data access we created a domain named **banking.com** and created appropriate users to access the domain. We created domain users and administrators so that they can access and administer the whole process. As all banks' servers will be the member of the domain, there will be no chance to enter a non-user of the domain to access any data. We created the domain by MS-Active Directory Services of Windows Server.

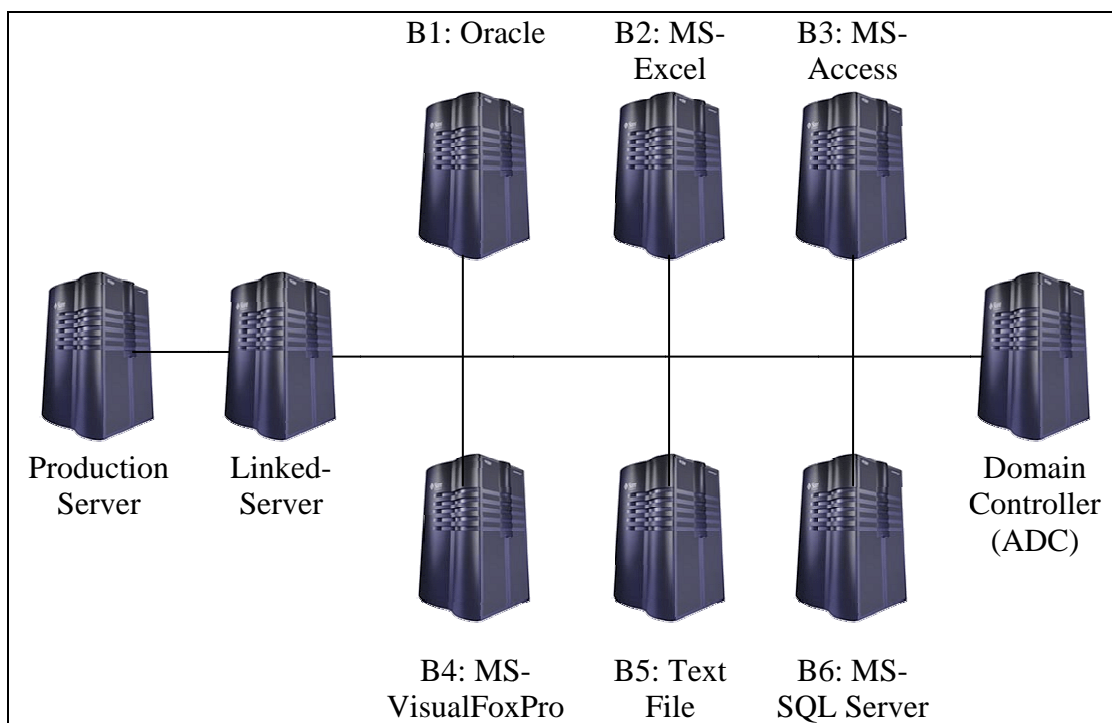


Fig. 8.1:Domain: banking.com

8.3 Database Views

After a table is created and populated with data, it may become necessary to prevent all users from accessing all columns of a table, for data security reasons. This would mean creating several tables having the appropriate number of columns and assigning specific users to each table, as required. This will answer data security requirements very well but will give rise to a great deal of redundant data being resident in tables, in the database.

To reduce redundant data to the minimum possible, database allows the creation of an object called a View. A View is mapped, to a SELECT sentence. The table on which the view is based is described in the FROM clause of the SELECT statement. The SELECT clause consists of a sub-set of the columns of the table. Thus a View, which is mapped to a table, will in effect have a sub-set of the actual columns of the table from which it is built. This technique offers a simple, effective way of hiding columns of a table.

An interesting fact about a View is that it is stored only as a definition in database. When a reference is made to a View, its definition is scanned, the base table is opened and the View created on top of the base table. Hence, a View holds no data at all, until a specific call to the View is made. This reduces redundant data on the HDD to a very large extent. When a View is used to manipulate table data, the underlying base table will be completely invisible. This will give the level of data security required.

The database engine treats a View just as though it was a base table. Hence a View can be queried exactly as though it was a base table. However, a query fired on a View will run slower than a query fired on a base table. This is because the View definition has to be retrieved from database system catalogue, the base table has to be identified and opened in memory and then the View has to be constructed on top of the base table, suitably masking table columns. Only then will the query actually execute and return the active data set.

In this project we used view for SQL Server, Oracle and MS-Visual FoxPro databases.

8.3.1 Creating Views for Bank-6 MS-SQL Server

```
CREATE VIEW V_TRANSACTIONS AS
SELECT A. A_NO, AT.DESCRPTION "ACCOUNT_TYPE", T.DESCRPTION,T.TD, T. DEBIT,
T.CREDIT,T.BALANCE FROM ACCOUNTS A,ACCOUNT_TYPE AT,TRANSACTIONS T
```

WHERE A. A_NO=T.A_NO AND A.T_ID=AT.T_ID

----- IN SQL SERVER THERE IS NO NEED OF SEMICOLON AT THE END OF THE VIEW DEFINITION-----

8.3.2 Creating Views for Bank-1 Oracle Server

CREATE OR REPLACE VIEW V_TRANSACTIONS AS

```
SELECT A. A_NO, AT.DESCRPTION "ACCOUNT_TYPE", T.DESCRPTION,T.TD, T. DEBIT,
T.CREDIT,T.BALANCE FROM ACCOUNTS A,ACCOUNT_TYPE AT,TRANSACTIONS T
WHERE A. A_NO=T.A_NO AND A.T_ID=AT.T_ID;
```

8.3.3 Creating Views for Bank-4 FoxPro Server

CREATE VIEW V_TRANSACTIONS

--WITH THE FOLLOWING CATEGORIES WE HAVE TO CREATE A VIEW MANUALLY

```
SELECT A. A_NO, AT.DESCRPTION "ACCOUNT_TYPE", T.DESCRPTION,T.TD, T. DEBIT,
T.CREDIT,T.BALANCE FROM ACCOUNTS A,ACCOUNT_TYPE AT,TRANSACTIONS T
WHERE A. A_NO=T.A_NO AND A.T_ID=AT.T_ID
```

--AFTER CREATING THE VIEW WE HAVE TO RUN THE VIEW

8.3.4 Creating View to See Filtered Data in Linked-Server

CREATE VIEW V_GATHERED_DATA AS

```
SELECT GD.B_ID,GD.CTG_ID,IC.TYPE, GD.SD,GD.ED,GD.CD,GD.CREDIT_AMOUNT,
GD.DEBIT_AMOUNT FROM GATHERED_DATA GD, INFO_CATEGORIES IC
WHERE IC.CTG_ID=GD.CTG_ID
```

CREATE OR REPLACE VIEW V_GATHERED_DATA AS

```
SELECT B.B_ID,B.NAME, GD.TR_ID, GD.PR_ID, GD.CTG_ID,IC.TYPE, GD.SD,GD.ED,GD.CD,GD.CREDIT_AMOUNT,
GD.DEBIT_AMOUNT FROM BANK B, GATHERED_DATA GD, INFO_CATEGORIES IC
WHERE IC.CTG_ID=GD.CTG_ID
AND B.B_ID=GD.B_ID;
```


8.3.5 Creating View for Production Server of the Central Bank

```
CREATE OR REPLACE VIEW V_FILTERED_DATA AS SELECT BANK.B_ID, BANK.NAME,
FILTERING_DATA.STATUS, INFO_CATEGORIES.TYPE ,PROCESS_INFO.PR_ID
FROM PROCESS_INFO, INFO_CATEGORIES, BANK, FILTERING_DATA
WHERE ((PROCESS_INFO.CTG_ID=INFO_CATEGORIES.CTG_ID)
AND (FILTERING_DATA.B_ID=BANK.B_ID)
AND (FILTERING_DATA.PR_ID=PROCESS_INFO.PR_ID));
```

```
CREATE VIEW V_COLLECTED_DATA AS SELECT BANK.B_ID, BANK.NAME, FILTERING_DATA.PR_ID, GATHERED_DATA.SD,
GATHERED_DATA.ED, GATHERED_DATA.CD, GATHERED_DATA.CREDIT_AMOUNT,
GATHERED_DATA.DEBIT_AMOUNT, INFO_CATEGORIES.TYPE
FROM BANK, FILTERING_DATA, GATHERED_DATA, INFO_CATEGORIES
WHERE ((FILTERING_DATA.B_ID=BANK.B_ID)
AND (GATHERED_DATA.B_ID=BANK.B_ID)
AND (GATHERED_DATA.CTG_ID=INFO_CATEGORIES.CTG_ID))
```

```
CREATE VIEW V_FILTER_DATA AS SELECT ALL BANK.B_ID,
BANK.NAME, FILTERING_DATA.PR_ID, FILTERING_DATA.STATUS
FROM BANK, FILTERING_DATA
WHERE (FILTERING_DATA.B_ID=BANK.B_ID)
```

/

8.4 Database User Security

Database security is provided by creating appropriate database users and providing database role and privileges to that user. In our project a database user is created who will be a read only user, i.e., he/she can only view (select) the data. Moreover, view is created first for data abstraction, that is also a read only data and then user is permitted to select the data from that view. So there is no chance of the user to alter any data of bank's production server. So it is ensured that any user of the central bank can only select the data and there is no chance to alter any data of the database, even to see any data that is not granted to the user.

8.4.1 MS-SQL Server

In MS-SQL Server we have created a user named test and provided the SELECT permission only as needed. This process is described by the following figures (Fig. 8.2,8.3, 8.4 and 8.5).

Fig. 8.2: Creating Database User (Step-1)

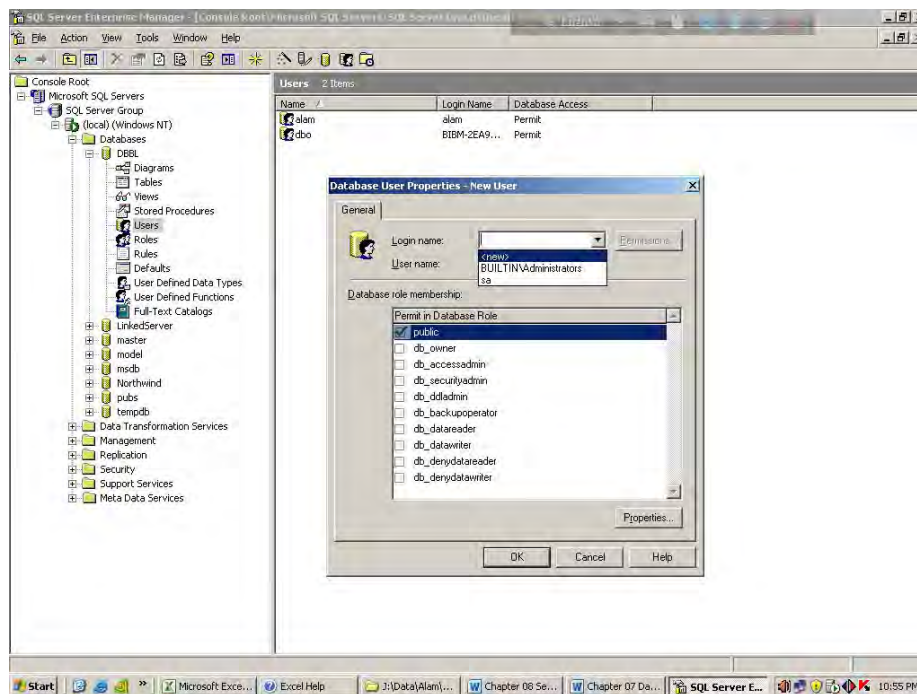


Fig. 8.3: Creating Database User (Step-2)

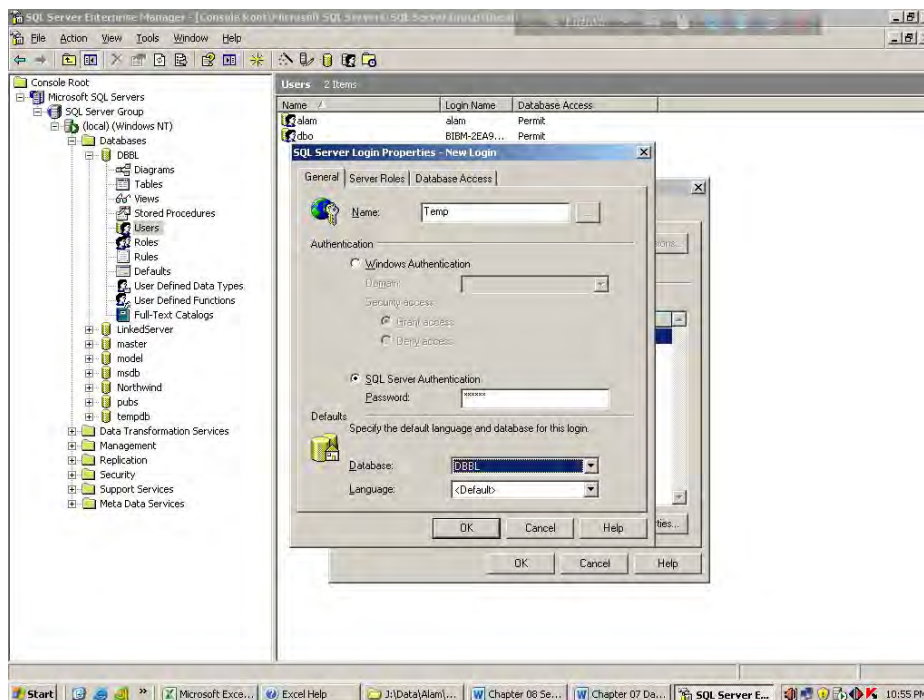


Fig. 8.4: Creating Database User (Step-3)

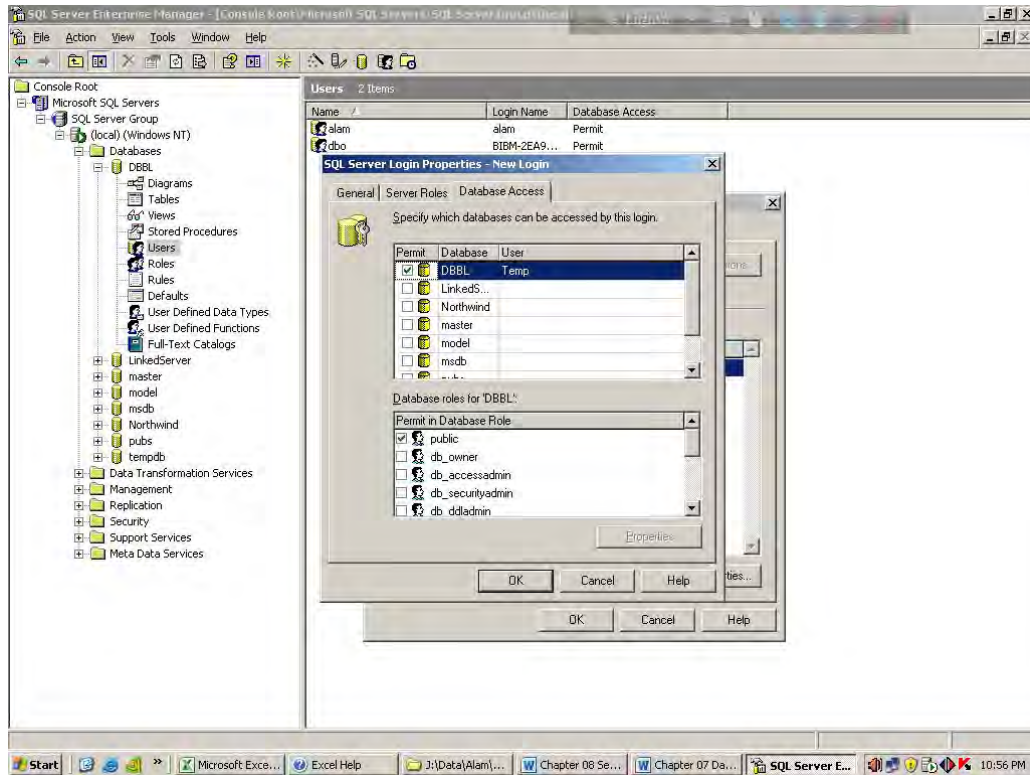
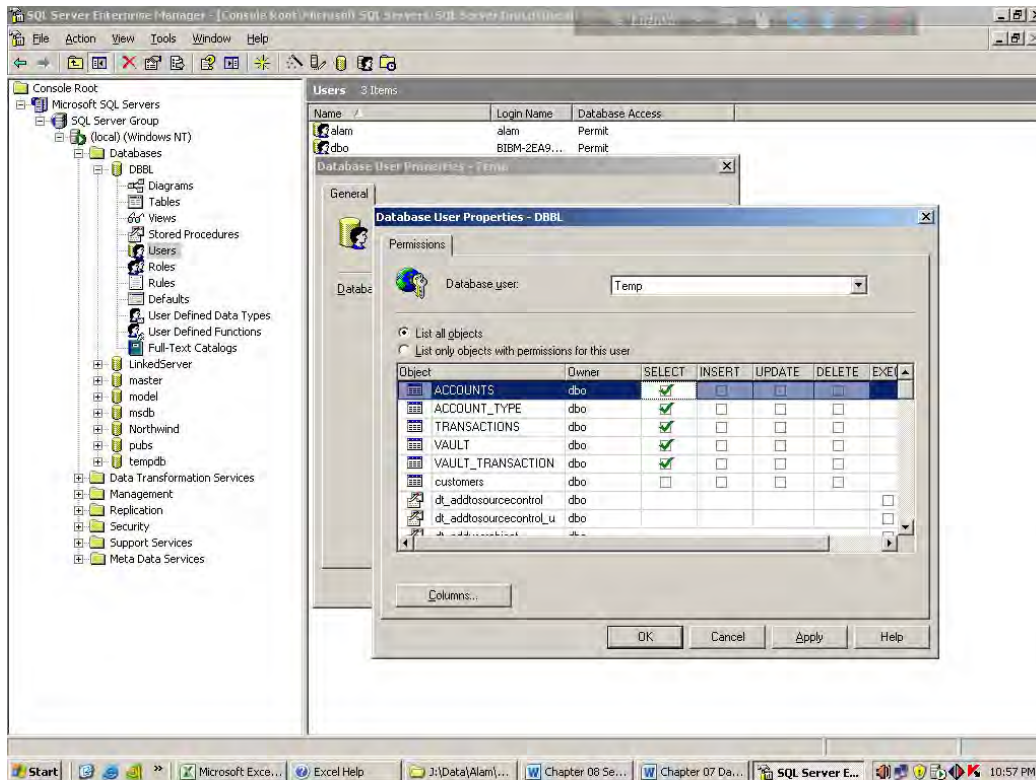


Fig. 8.5: Creating Database User (Step-4)



8.4.2 Oracle

In case of Oracle Server, we have created a user named test and granted the SELECT permission only as needed. This process is described by the following SQL commands.

At first we have connected to Oracle server as System Administrator (DBA). The user name 'test' is created by the following command.

```
SQL> create user test identified by antara;
```

User created.

Now permission is granted to 'test' so that he/she can connect to the oracle server as follows.

```
SQL> grant connect to test;
```

Grant succeeded.

He/she is also permitted to share the resource (data) of the oracle server as follows.

```
SQL> grant resource to test;
```

Grant succeeded.

Now, a user named 'test' is created successfully as a database user of Oracle. Now, the owner of the schema, 'bb', has given read only permission to the user named 'test' so that he/she can only select the data of the view **v_gathered_data**. By using the following commands we have done this.

```
SQL> connect
```

Enter user-name: bb

Enter password: *****

Connected.

```
SQL> grant select on v_gathered_data to test;
```

Grant succeeded.

8.4.3 Others

Other than SQL Server and Oracle, i.e., for MS-Access, MS-Visual FoxPro, MS-Excel and Text files, there is no built-in database user management system. So, we cannot create a user in those cases and provide grant permissions to the users like Oracle or SQL Server. In this situation, we provided read only sharing permissions to the folder where data files are kept by the operating system. As read only permissions are provided, it is ensured that no alteration of data is possible by the remote users of the central bank. Fig. 8.6 and 8.7 describes this process.

For more and better security management it is also suggested that, if banks feel that security is not ensured or don't have trust regarding security, they can provide an online replica of the production server or data files.

Fig. 8.6: Sharing Folder Permissions (Step-1)

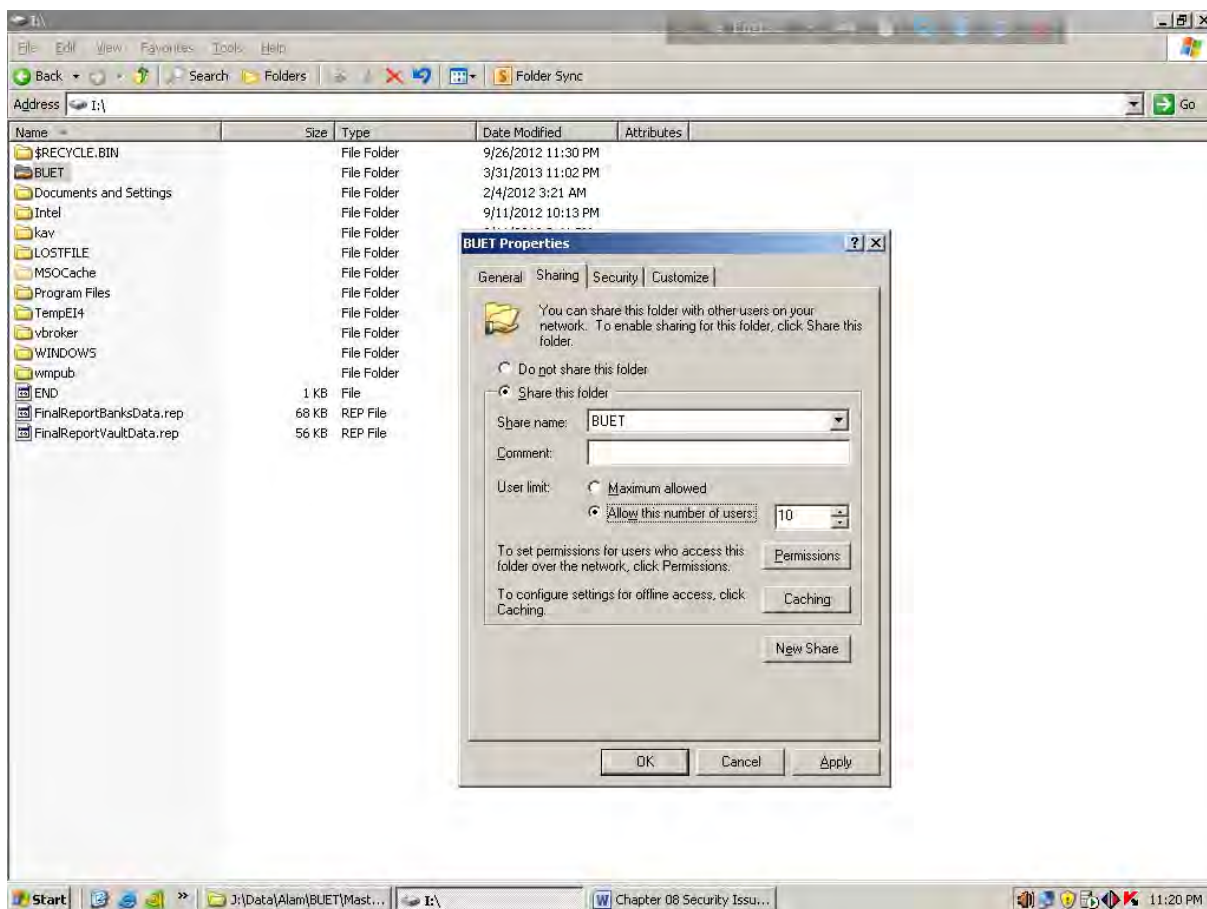
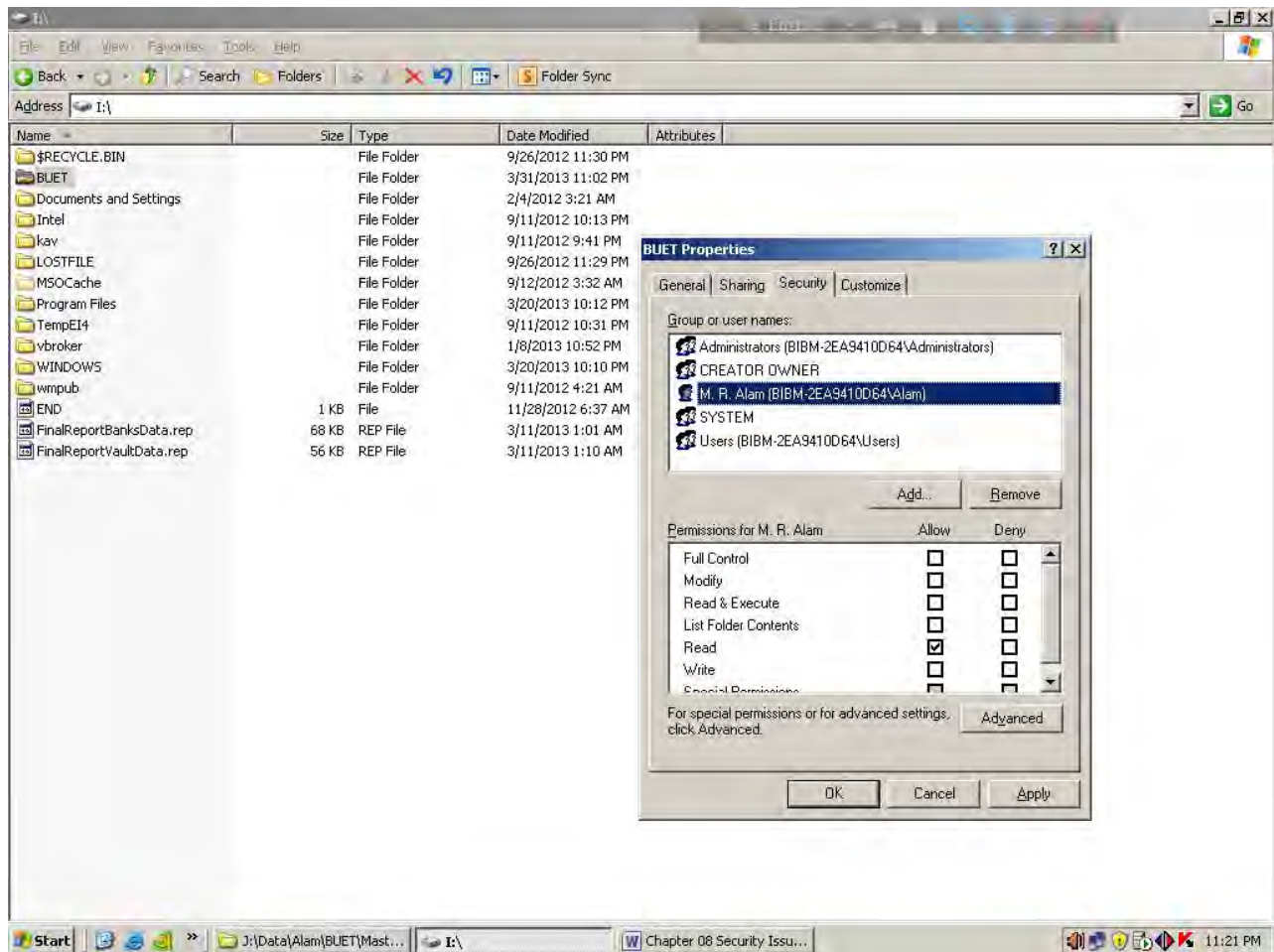


Fig. 8.7: Sharing Folder Permissions (Step-2)



8.5 Summary

Finally, in this chapter we elaborated the most important part of the simulated project, i.e., data security. The process of domain configuration, creation of appropriate views and user level security management are also discussed here. Mechanism regarding database object use, granting and revoking roles and privileges are also elaborated. Sharing folder permissions are also shown in case of non-database files like MS-Excel and ASCII text files.

Chapter 9

Conclusion

9.1 Conclusion

Banking is the backbone of modern economy. Now-a-days, modern banking totally depends on information and communication technology. For the economic development of Bangladesh it is very much important to develop the banking sector. Formulation and implementation of the national economic policy very much depends on information regarding banking. Bangladesh Bank (BB) being a regulatory body and the central bank of Bangladesh collects, filters, organizes and processes periodical data from different commercial banks operating in Bangladesh for better policy, monitoring and management of the banking sector. There are a number of departments and authorities involved with this process. But it has been facing severe problems to have proper information regarding export, import, profit, tax, loans & advances, defaults, liquid asset, etc. from different commercial banks for quick policy making. At present, in most of the cases, all the information is collected manually and stored in separate manual registers, text files and MS-Excel sheets in different departments which do not follow a uniform format. This decentralized information is complicated to aggregate for monitoring and policy purpose from a central point. By using current infrastructure it is not also possible for BB to access a heterogeneous banking system where different banks use different banking software and database. Keeping this problem in mind, in this project, we tried to develop a generalized database to collect, filter, organize process and store periodical data in a unique standard format.

Firstly, we studied the banking system rigorously to explore the entities and their attributes. We got number of entities incurred with this system. Then we explored the attributes of each entity. After getting the entities and their attributes, we found out the relationship among the entities. We also got normal relationship among the entities.

The central bank is the ultimate user of the system. So in the proposed system we used the Linked-Server concept of Microsoft SQL-Server to gather data by using distributed query technique. After filtering and processing in SQL-Server, data is exported to the production server of the central bank, which is running an Oracle database management system. Security issues are

strictly handled in this regard. Finally, a set of forms and parameterized reports have been designed that can be run from different departments by using central form and report server which will assist the regulatory body for proper monitoring, control and making policy decision. However, this system will require reliable communication system but a very low bandwidth is enough to gather data, as only summarized data will travel from banks' servers to linked-server.

This system is simulated in the HUAWEI-Lab of the IICT department successfully. Now, with the proper permission of BB and cooperation of different commercial banks, this system can be implemented for the betterment of the banking sector and economy of the country. This report would be very useful for the central bank's authority, if they implement the developed database.

9.2 Future Works

Any central bank of this world where banking system is same as Bangladesh (mainly developing and neighboring countries like us) can implement this database for their banking system. Even a bank which has a heterogeneous scattered branch banking system, having different banking software and databases, can also customize this system to gather data for their head office policy purpose. Corporate organizations, like garments and manufacturing industries, which have different industries across the country can also use similar system to gather data to the central administrative offices. International organizations which have separate offices all over the globe can also get benefits from a system like this.

References

- [1] “The Scheduled Bank Statistics”, Bangladesh Bank, Dhaka, Bangladesh, 2012.
- [2] Rahman M. R., “IT readiness of banks for business in cyberspace”, a keynote paper presented in an workshop, organized by BIBM, 2010.
- [3] Mia, Rahman and Uddin, “E-banking: evolution, status and prospects”, *The Cost and Management*, vol.35, no.1, pp. 36-48., 2007.
- [4] Rahman, M. Mizanur, "Present status of e-banking in Bangladesh", *Journal of the Institute of the Bankers, Bangladesh*, vol. 50, no. 1, 2003.
- [5] BIBM Survey, "Computerization and information technology in the banking sector", 2010.
- [6] Raihan, et al., "Computerization and information technology in the banking sector: hindrances and remedies", *Bank Parikrama*, vol. XXVI, no. 1, 2010.
- [7] “Guideline for Information and Communication Technology for Scheduled Banks and Financial Institutions”, Bangladesh Bank, Dhaka, Bangladesh, 2005.
- [8] “Guideline for ICT Security for Scheduled Banks and Financial Institutions”, Bangladesh Bank, Dhaka, Bangladesh, 2010.
- [9] Rahman, Atiur, “Digital Bangladesh Bank”, *The Daily Star*, July 05, 2010.
- [10] Banstola, A., “Prospects and challenges of e-banking in Nepal”, *The Journal of Nepalese Business Studies*, vol. IV, no. 1, Dec. 2007.
- [11] Bhasin T. M., “E-commerce in Indian banking”, India: Authors Press, 2008.
- [12] Rahman M.M., “Innovative technology and bank profitability: The Bangladesh experience”, working paper series: WP 0803, Policy Analysis Unit, Bangladesh Bank, 2007.

Appendix-I

a)

```
INSERT INTO BANK_VALUES('B4','SONALI BANK LIMITED');
INSERT INTO BANK_VALUES('B2','BANGLADESH KRISHI BANK');
INSERT INTO BANK_VALUES('B1','DUTCH-BANGLA BANK LIMITED');
INSERT INTO BANK_VALUES('B6','HSBC LIMITED');
INSERT INTO BANK_VALUES('B5','RUPALI BANK LIMITED');
INSERT INTO BANK_VALUES('B3','BASIC BANK LIMITED');

INSERT INTO INFO_CATEGORIES_VALUES('CTG01','TAX');
INSERT INTO INFO_CATEGORIES_VALUES('CTG02','VAT');
INSERT INTO INFO_CATEGORIES_VALUES('CTG03','IMPORT LC');
INSERT INTO INFO_CATEGORIES_VALUES('CTG04','EXPORT LC');
INSERT INTO INFO_CATEGORIES_VALUES('CTG05','LOAN');
INSERT INTO INFO_CATEGORIES_VALUES('CTG06','LIQUID ASSET');
INSERT INTO INFO_CATEGORIES_VALUES('CTG07','FDR');
INSERT INTO INFO_CATEGORIES_VALUES('CTG08','CHARGES');
INSERT INTO INFO_CATEGORIES_VALUES('CTG09','ADVANCES');
INSERT INTO INFO_CATEGORIES_VALUES('CTG10','DEFAULTS');
INSERT INTO INFO_CATEGORIES_VALUES('CTG11','INTEREST INCOME');
```

b)

```
INSERT INTO BANK_VALUES('B1','SONALI BANK LIMITED');
INSERT INTO BANK_VALUES('B2','BANGLADESH KRISHI BANK');
INSERT INTO BANK_VALUES('B3','DUTCH-BANGLA BANK LIMITED');
INSERT INTO BANK_VALUES('B4','HSBC LIMITED');
INSERT INTO BANK_VALUES('B5','RUPALI BANK LIMITED');
INSERT INTO BANK_VALUES('B6','BASIC BANK LIMITED');

INSERT INTO INFO_CATEGORIES_VALUES('CTG01','TAX');
INSERT INTO INFO_CATEGORIES_VALUES('CTG02','VAT');
INSERT INTO INFO_CATEGORIES_VALUES('CTG03','IMPORT LC');
INSERT INTO INFO_CATEGORIES_VALUES('CTG04','EXPORT LC');
INSERT INTO INFO_CATEGORIES_VALUES('CTG05','LOAN');
```

```

INSERT INTO INFO_CATEGORIES VALUES('CTG06','LIQUID ASSET');
INSERT INTO INFO_CATEGORIES VALUES('CTG07','FDR');
INSERT INTO INFO_CATEGORIES VALUES('CTG08','CHARGES');
INSERT INTO INFO_CATEGORIES VALUES('CTG09','ADVANCES');
INSERT INTO INFO_CATEGORIES VALUES('CTG10','DEFAULTS');
INSERT INTO INFO_CATEGORIES VALUES('CTG11','INTEREST INCOME');

```

c)

```

INSERT INTO BANK VALUES('B1','SONALI BANK LIMITED');
INSERT INTO BANK VALUES('B2','BANGLADESH KRISHI BANK');
INSERT INTO BANK VALUES('B3','DUTCH-BANGLA BANK LIMITED');
INSERT INTO BANK VALUES('B4','HSBC LIMITED');
INSERT INTO BANK VALUES('B5','RUPALI BANK LIMITED');
INSERT INTO BANK VALUES('B6','BASIC BANK LIMITED');

```

```

INSERT INTO INFO_CATEGORIES VALUES('CTG01','TAX');
INSERT INTO INFO_CATEGORIES VALUES('CTG02','VAT');
INSERT INTO INFO_CATEGORIES VALUES('CTG03','IMPORT LC');
INSERT INTO INFO_CATEGORIES VALUES('CTG04','EXPORT LC');
INSERT INTO INFO_CATEGORIES VALUES('CTG05','LOAN');
INSERT INTO INFO_CATEGORIES VALUES('CTG06','LIQUID ASSET');
INSERT INTO INFO_CATEGORIES VALUES('CTG07','FDR');
INSERT INTO INFO_CATEGORIES VALUES('CTG08','CHARGES');
INSERT INTO INFO_CATEGORIES VALUES('CTG09','ADVANCES');
INSERT INTO INFO_CATEGORIES VALUES('CTG10','DEFAULTS');
INSERT INTO INFO_CATEGORIES VALUES('CTG11','ATM');

```

d)

```

INSERT INTO CUSTOMERS (C_ID,NAME,ADDRESS, N_ID,PHONE) VALUES('C001', 'MD. HAFIZUR RAHMAN', '413,
MIRPUR,DHAKA','1235695823254','01775858256');

INSERT INTO CUSTOMERS (C_ID,NAME,ADDRESS, N_ID,PHONE) VALUES('C002', 'MD. HELEL UDDIN', 'H-3, R-
5,DHANMONDI,DHAKA','1885895823254','01556323244');

INSERT INTO CUSTOMERS (C_ID,NAME,ADDRESS, N_ID,PHONE) VALUES('C003', 'MS. TAHMINA AKHTER',
'H#12,R#15,SHYAMOLI,DHAKA','2365251425632','9003031');

INSERT INTO CUSTOMERS (C_ID,NAME,ADDRESS, N_ID,PHONE) VALUES('C004', 'MS. KANIZ RBBI',
'744,KAZIPARA,MIRPUR,DHAKA','4569871425632','9003088, 0174586958');

```

```

INSERT INTO ACCOUNT_TYPE VALUES('T1','SAVINGS');
INSERT INTO ACCOUNT_TYPE VALUES('T2','CURRENT');
INSERT INTO ACCOUNT_TYPE VALUES('T3','FDR');
INSERT INTO ACCOUNT_TYPE VALUES('T4','LOAN AND ADVANCE');
INSERT INTO ACCOUNT_TYPE VALUES('T5','IMPORT LC');
INSERT INTO ACCOUNT_TYPE VALUES('T6','EXPORT LC');

INSERT INTO ACCOUNTS VALUES('A001','T1','C001', GETDATE()-300, 'SALAMA AKHTER: SISTER',500.00,6673);
INSERT INTO ACCOUNTS VALUES('A002','T1','C002', GETDATE()-295, 'RABBANI MAHBUB: SON',500.00,2618.1);
INSERT INTO ACCOUNTS VALUES('A003','T1','C003', GETDATE()-290, 'MOKBUL HOSSAIN: FATHER',5000.00,49200);
INSERT INTO ACCOUNTS VALUES('A004','T1','C004', GETDATE()-280, 'FUAD HASSAN: BROTHER',1000.00,89650);
INSERT INTO ACCOUNTS VALUES('A005','T2','C001', GETDATE()-270, 'SALAMA AKHTER: SISTER',50000.00,54145);
INSERT INTO ACCOUNTS VALUES('A006','T3','C002', GETDATE()-170, 'RABBANI MAHBUB: SON',0,798870);
INSERT INTO ACCOUNTS VALUES('A007','T4','C003', GETDATE()-150, 'MOKBUL HOSSAIN: FATHER',0,9640);
INSERT INTO ACCOUNTS VALUES('A008','T5','C004', GETDATE()-130, 'FUAD HASSAN: BROTHER',0,6889750);
INSERT INTO ACCOUNTS VALUES('A009','T6','C001', GETDATE()-100, 'SALAMA AKHTER: SISTER',0,49119750);
INSERT INTO ACCOUNTS VALUES('A010','T2','C002', GETDATE()-80, 'RABBANI MAHBUB: SON',50000.00,399310);
INSERT INTO ACCOUNTS VALUES('A011','T2','C003', GETDATE()-60, 'MOKBUL HOSSAIN: FATHER',50000.00,93705);
INSERT INTO ACCOUNTS VALUES('A012','T6','C004', GETDATE()-30, 'FUAD HASSAN: BROTHER',0,80192605);

INSERT INTO TRANSACTIONS VALUES('TR00000001', 'A001','CASH RECEIVE',GETDATE()-290,NULL,5000,5000);
INSERT INTO TRANSACTIONS VALUES('TR00000002', 'A001','CHEQUE',GETDATE()-215,1000,NULL,4000);
INSERT INTO TRANSACTIONS VALUES('TR00000003', 'A001','TRANSFER',GETDATE()-180,NULL,5000,9000);
INSERT INTO TRANSACTIONS VALUES('TR00000004', 'A001','ATM',GETDATE()-100,2000,NULL,7000);
INSERT INTO TRANSACTIONS VALUES('TR00000005', 'A002','CASH RECEIVE',GETDATE()-290,NULL,2000,2000);
INSERT INTO TRANSACTIONS VALUES('TR00000006', 'A002','CHEQUE',GETDATE()-200,500,NULL,1500);
INSERT INTO TRANSACTIONS VALUES('TR00000007', 'A002','TRANSFER',GETDATE()-100,NULL,1500,3000);
INSERT INTO TRANSACTIONS VALUES('TR00000008', 'A002','ATM',GETDATE()-50,100,NULL,2900);
INSERT INTO TRANSACTIONS VALUES('TR00000009', 'A003','CASH RECEIVE',GETDATE()-200,NULL,50000,50000);
INSERT INTO TRANSACTIONS VALUES('TR00000010', 'A004','CHEQUE CLEARING',GETDATE()-100,NULL,100000,100000);
INSERT INTO TRANSACTIONS VALUES('TR00000011', 'A005','CASH RECEIVE',GETDATE()-200,NULL,55000,55000);
INSERT INTO TRANSACTIONS VALUES('TR00000012', 'A006','CHEQUE CLEARING',GETDATE()-150,NULL,800000,800000);
INSERT INTO TRANSACTIONS VALUES('TR00000013', 'A007','CHEQUE CLEARING',GETDATE()-120,NULL,200000,200000);
INSERT INTO TRANSACTIONS VALUES('TR00000014', 'A007','CASH WITHDRAWAL',GETDATE()-100,200000,NULL,0);

```

```
INSERT INTO TRANSACTIONS VALUES('TR00000015', 'A007','CASH RECEIVE',GETDATE()-80,NULL, 10000,10000);
INSERT INTO TRANSACTIONS VALUES('TR00000016', 'A008','CHEQUE CLEARING',GETDATE()-125,NULL,50000000,50000000);
INSERT INTO TRANSACTIONS VALUES('TR00000017', 'A008','TRANSFER',GETDATE()-100,NULL,20000000,70000000);
INSERT INTO TRANSACTIONS VALUES('TR00000018', 'A008','CHEQUE CLEARING',GETDATE(),60000000,NULL,10000000);
INSERT INTO TRANSACTIONS VALUES('TR00000019', 'A009','CHEQUE CLEARING',GETDATE()-90,NULL,25000000,25000000);
INSERT INTO TRANSACTIONS VALUES('TR00000020', 'A009','CHEQUE CLEARING',GETDATE()-50,NULL,45000000,70000000);
INSERT INTO TRANSACTIONS VALUES('TR00000021', 'A009','CHEQUE CLEARING',GETDATE()-25,NULL,10000000,80000000);
INSERT INTO TRANSACTIONS VALUES('TR00000022', 'A010','CHEQUE CLEARING',GETDATE()-60,NULL,400000,400000);
INSERT INTO TRANSACTIONS VALUES('TR00000023', 'A011','TRANSFER',GETDATE()-40,NULL,95000,95000);
INSERT INTO TRANSACTIONS VALUES('TR00000024', 'A012','TRANSFER',GETDATE()-40,NULL,195000,195000);
INSERT INTO TRANSACTIONS VALUES('TR00000025', 'A001','VAT','31-DEC-12',70,NULL,6930);
INSERT INTO TRANSACTIONS VALUES('TR00000026', 'A002','VAT','31-DEC-12',29,NULL,2871);
INSERT INTO TRANSACTIONS VALUES('TR00000027', 'A003','VAT','31-DEC-12',500,NULL,49500);
INSERT INTO TRANSACTIONS VALUES('TR00000028', 'A004','VAT','31-DEC-12',1000,NULL,99000);
INSERT INTO TRANSACTIONS VALUES('TR00000029', 'A005','VAT','31-DEC-12',550,NULL,54450);
INSERT INTO TRANSACTIONS VALUES('TR00000030', 'A006','VAT','31-DEC-12',800,NULL,799200);
INSERT INTO TRANSACTIONS VALUES('TR00000031', 'A007','VAT','31-DEC-12',100,NULL,9900);
INSERT INTO TRANSACTIONS VALUES('TR00000032', 'A008','VAT','31-DEC-12',100000,NULL,9900000);
INSERT INTO TRANSACTIONS VALUES('TR00000033', 'A009','VAT','31-DEC-12',800000,NULL,79200000);
INSERT INTO TRANSACTIONS VALUES('TR00000034', 'A010','VAT','31-DEC-12',400,NULL,399600);
INSERT INTO TRANSACTIONS VALUES('TR00000035', 'A011','VAT','31-DEC-12',950,NULL,94050);
INSERT INTO TRANSACTIONS VALUES('TR00000036', 'A012','VAT','31-DEC-12',1950,NULL,193050);
INSERT INTO TRANSACTIONS VALUES('TR00000037', 'A001','TAX','31-DEC-12',7,NULL,6923);
INSERT INTO TRANSACTIONS VALUES('TR00000038', 'A002','TAX','31-DEC-12',2.9,NULL,2868.1);
INSERT INTO TRANSACTIONS VALUES('TR00000039', 'A003','TAX','31-DEC-12',50,NULL,49450);
INSERT INTO TRANSACTIONS VALUES('TR00000040', 'A004','TAX','31-DEC-12',100,NULL,89900);
INSERT INTO TRANSACTIONS VALUES('TR00000041', 'A005','TAX','31-DEC-12',55,NULL,54395);
INSERT INTO TRANSACTIONS VALUES('TR00000042', 'A006','TAX','31-DEC-12',80,NULL,799120);
INSERT INTO TRANSACTIONS VALUES('TR00000043', 'A007','TAX','31-DEC-12',10,NULL,9890);
INSERT INTO TRANSACTIONS VALUES('TR00000044', 'A008','TAX','31-DEC-12',10000,NULL,9890000);
INSERT INTO TRANSACTIONS VALUES('TR00000045', 'A009','TAX','31-DEC-12',80000,NULL,79120000);
INSERT INTO TRANSACTIONS VALUES('TR00000046', 'A010','TAX','31-DEC-12',40,NULL,399560);
INSERT INTO TRANSACTIONS VALUES('TR00000047', 'A011','TAX','31-DEC-12',95,NULL,93955);
INSERT INTO TRANSACTIONS VALUES('TR00000048', 'A012','TAX','31-DEC-12',195,NULL,192855);
INSERT INTO TRANSACTIONS VALUES('TR00000049', 'A001','CHARGES','31-DEC-12',250,NULL,6673);
```

```

INSERT INTO TRANSACTIONS VALUES('TR00000050', 'A002','CHARGES','31-DEC-12',250,NULL,2618.1);
INSERT INTO TRANSACTIONS VALUES('TR00000051', 'A003','CHARGES','31-DEC-12',250,NULL,49200);
INSERT INTO TRANSACTIONS VALUES('TR00000052', 'A004','CHARGES','31-DEC-12',250,NULL,89650);
INSERT INTO TRANSACTIONS VALUES('TR00000053', 'A005','CHARGES','31-DEC-12',250,NULL,54145);
INSERT INTO TRANSACTIONS VALUES('TR00000054', 'A006','CHARGES','31-DEC-12',250,NULL,798870);
INSERT INTO TRANSACTIONS VALUES('TR00000055', 'A007','CHARGES','31-DEC-12',250,NULL,9640);
INSERT INTO TRANSACTIONS VALUES('TR00000056', 'A008','CHARGES','31-DEC-12',250,NULL,9889750);
INSERT INTO TRANSACTIONS VALUES('TR00000057', 'A009','CHARGES','31-DEC-12',250,NULL,79119750);
INSERT INTO TRANSACTIONS VALUES('TR00000058', 'A010','CHARGES','31-DEC-12',250,NULL,399310);
INSERT INTO TRANSACTIONS VALUES('TR00000059', 'A011','CHARGES','31-DEC-12',250,NULL,93705);
INSERT INTO TRANSACTIONS VALUES('TR00000060', 'A012','CHARGES','31-DEC-12',250,NULL,192605);

```

```

INSERT INTO TRANSACTIONS VALUES('TR00000061', 'A012','EXPORT LC',GETDATE()-40,50000000,NULL,50192605);
INSERT INTO TRANSACTIONS VALUES('TR00000062', 'A012','EXPORT LC',GETDATE()-30,10000000,NULL,60192605);
INSERT INTO TRANSACTIONS VALUES('TR00000063', 'A012','EXPORT LC',GETDATE()-20,20000000,NULL,80192605);
INSERT INTO TRANSACTIONS VALUES('TR00000064', 'A008','IMPORT LC',GETDATE()-40,NULL,1000000,8889750);
INSERT INTO TRANSACTIONS VALUES('TR00000065', 'A008','IMPORT LC',GETDATE()-30,NULL,1000000,7889750);
INSERT INTO TRANSACTIONS VALUES('TR00000066', 'A008','IMPORT LC',GETDATE()-10,NULL,1000000,6889750);
INSERT INTO TRANSACTIONS VALUES('TR00000067', 'A009','IMPORT LC',GETDATE()-40,NULL,1000000,69119750);
INSERT INTO TRANSACTIONS VALUES('TR00000068', 'A009','IMPORT LC',GETDATE()-30,NULL,1000000,59119750);
INSERT INTO TRANSACTIONS VALUES('TR00000069', 'A009','IMPORT LC',GETDATE()-10,NULL,1000000,49119750);

```

```

INSERT INTO VAULT VALUES('V001', 'DHANMONDI BRANCH', 'MD. ZAMIL AHMED', 25000000, 30000000);
INSERT INTO VAULT VALUES('V002', 'SUNAMGONJ BRANCH', 'MS. ANTARA ZERIN',20000000, 40000000);
INSERT INTO VAULT VALUES('V003', 'KHULNA BRANCH', 'MD. AKHTER HAMID', 13000000, 15000000);
INSERT INTO VAULT VALUES('V004', 'RAJSHAHI BRANCH', 'MD. ZAMIL AHMED', 10000000, 15000000);

```

```

INSERT INTO VAULT_TRANSACTION VALUES('VTR0000001', 'V001', GETDATE(), NULL, 50000000,50000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000002', 'V001', GETDATE(), NULL, 30000000,80000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000003', 'V001', GETDATE(), NULL, 10000000,90000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000004', 'V001', GETDATE()+1, 10000000,NULL,80000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000005', 'V001', GETDATE()+2, 20000000,NULL,60000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000006', 'V001', GETDATE()+3, 30000000,NULL,30000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000007', 'V002', GETDATE(), NULL, 2000000,2000000);

```

```

INSERT INTO VAULT_TRANSACTION VALUES('VTR0000008', 'V002', GETDATE(), NULL, 1000000,3000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000009', 'V002', GETDATE(), NULL, 5000000,8000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000010', 'V002', GETDATE()+1, 1000000,NULL,7000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000011', 'V002', GETDATE()+2, 1000000,NULL,6000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000012', 'V002', GETDATE()+3, 2000000,NULL,4000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000013', 'V003', GETDATE(), NULL, 25000000,25000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000014', 'V003', GETDATE()+1, 10000000,NULL,15000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000015', 'V004', GETDATE(), NULL, 25000000,25000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000016', 'V004', GETDATE()+1, 10000000,NULL,15000000);

```

e)

```

INSERT INTO CUSTOMERS (C_ID,NAME,ADDRESS, N_ID,PHONE) VALUES('C001', 'MD. HAFIZUR RAHMAN', '413,
MIRPUR,DHAKA','1235695823254','01775858256');

```

```

INSERT INTO CUSTOMERS (C_ID,NAME,ADDRESS, N_ID,PHONE) VALUES('C002', 'MD. HELEL UDDIN', 'H-3, R-
5,DHANMONDI,DHAKA','1885895823254','01556323244');

```

```

INSERT INTO CUSTOMERS (C_ID,NAME,ADDRESS, N_ID,PHONE) VALUES('C003', 'MS. TAHMINA AKHTER',
'H#12,R#15,SHYAMOLI,DHAKA','2365251425632','9003031');

```

```

INSERT INTO CUSTOMERS (C_ID,NAME,ADDRESS, N_ID,PHONE) VALUES('C004', 'MS. KANIZ RBBI',
'744,KAZIPARA,MIRPUR,DHAKA','4569871425632','9003088, 0174586958');

```

```

INSERT INTO ACCOUNT_TYPE VALUES('T1','SAVINGS');

```

```

INSERT INTO ACCOUNT_TYPE VALUES('T2','CURRENT');

```

```

INSERT INTO ACCOUNT_TYPE VALUES('T3', 'FDR');

```

```

INSERT INTO ACCOUNT_TYPE VALUES('T4','LOAN');

```

```

INSERT INTO ACCOUNT_TYPE VALUES('T5','IMPORT LC');

```

```

INSERT INTO ACCOUNT_TYPE VALUES('T6', 'EXPORT LC');

```

```

INSERT INTO ACCOUNTS VALUES('A001','T1','C001', SYSDATE-300, 'SALAMA AKHTER: SISTER',500.00,6673);

```

```

INSERT INTO ACCOUNTS VALUES('A002','T1','C002', SYSDATE-295, 'RABBANI MAHBUB: SON',500.00,2618.1);

```

```

INSERT INTO ACCOUNTS VALUES('A003','T1','C003', SYSDATE-290, 'MOKBUL HOSSAIN: FATHER',5000.00,49200);

```

```

INSERT INTO ACCOUNTS VALUES('A005','T2','C001', SYSDATE-270, 'SALAMA AKHTER: SISTER',50000.00,54145);

```

```

INSERT INTO ACCOUNTS VALUES('A006','T3','C002', SYSDATE-170, 'RABBANI MAHBUB: SON',0,798870);

```

```

INSERT INTO ACCOUNTS VALUES('A007','T4','C003', SYSDATE-150, 'MOKBUL HOSSAIN: FATHER',0,9640);

```

```

INSERT INTO ACCOUNTS VALUES('A008','T5','C004', SYSDATE-130, 'FUAD HASSAN: BROTHER',0,6889750);

```

```

INSERT INTO ACCOUNTS VALUES('A009','T6','C001', SYSDATE-100, 'SALAMA AKHTER: SISTER',0,49119750);

```

```

INSERT INTO ACCOUNTS VALUES('A010','T2','C002', SYSDATE-80, 'RABBANI MAHBUB: SON',50000.00,399310);

```

```

INSERT INTO ACCOUNTS VALUES('A011','T2','C003', SYSDATE-60, 'MOKBUL HOSSAIN: FATHER',50000.00,93705);

```

```
INSERT INTO ACCOUNTS VALUES('A012','T6','C004', SYSDATE-30, 'FUAD HASSAN: BROTHER',0,80192605);

INSERT INTO TRANSACTIONS VALUES('TR00000001', 'A001','CASH RECEIVE',SYSDATE-290,NULL,5000,5000);
INSERT INTO TRANSACTIONS VALUES('TR00000002', 'A001','CHEQUE',SYSDATE-215,1000,NULL,4000);
INSERT INTO TRANSACTIONS VALUES('TR00000003', 'A001','TRANSFER',SYSDATE-180,NULL,5000,9000);
INSERT INTO TRANSACTIONS VALUES('TR00000004', 'A001','ATM',SYSDATE-100,2000,NULL,7000);
INSERT INTO TRANSACTIONS VALUES('TR00000005', 'A002','CASH RECEIVE',SYSDATE-290,NULL,2000,2000);
INSERT INTO TRANSACTIONS VALUES('TR00000006', 'A002','CHEQUE',SYSDATE-200,500,NULL,1500);
INSERT INTO TRANSACTIONS VALUES('TR00000007', 'A002','TRANSFER',SYSDATE-100,NULL,1500,3000);
INSERT INTO TRANSACTIONS VALUES('TR00000008', 'A002','ATM',SYSDATE-50,100,NULL,2900);
INSERT INTO TRANSACTIONS VALUES('TR00000009', 'A003','CASH RECEIVE',SYSDATE-200,NULL,50000,50000);
INSERT INTO TRANSACTIONS VALUES('TR00000011', 'A005','CASH RECEIVE',SYSDATE-200,NULL,55000,55000);
INSERT INTO TRANSACTIONS VALUES('TR00000012', 'A006','CHEQUE CLEARING',SYSDATE-150,NULL,800000,800000);
INSERT INTO TRANSACTIONS VALUES('TR00000013', 'A007','CHEQUE CLEARING',SYSDATE-120,NULL,200000,200000);
INSERT INTO TRANSACTIONS VALUES('TR00000014', 'A007','CASH WITHDRAWAL',SYSDATE-100,200000,NULL,0);
INSERT INTO TRANSACTIONS VALUES('TR00000015', 'A007','CASH RECEIVE',SYSDATE-80,NULL, 10000,10000);
INSERT INTO TRANSACTIONS VALUES('TR00000016', 'A008','CHEQUE CLEARING',SYSDATE-125,NULL,5000000,50000000);
INSERT INTO TRANSACTIONS VALUES('TR00000017', 'A008','TRANSFER',SYSDATE-100,NULL,20000000,70000000);
INSERT INTO TRANSACTIONS VALUES('TR00000018', 'A008','CHEQUE CLEARING-50',SYSDATE,60000000,NULL,10000000);
INSERT INTO TRANSACTIONS VALUES('TR00000019', 'A009','CHEQUE CLEARING',SYSDATE-90,NULL,25000000,25000000);
INSERT INTO TRANSACTIONS VALUES('TR00000020', 'A009','CHEQUE CLEARING',SYSDATE-50,NULL,45000000,70000000);
INSERT INTO TRANSACTIONS VALUES('TR00000021', 'A009','CHEQUE CLEARING',SYSDATE-25,NULL,10000000,80000000);
INSERT INTO TRANSACTIONS VALUES('TR00000022', 'A010','CHEQUE CLEARING',SYSDATE-60,NULL,400000,400000);
INSERT INTO TRANSACTIONS VALUES('TR00000023', 'A011','TRANSFER',SYSDATE-40,NULL,95000,95000);
INSERT INTO TRANSACTIONS VALUES('TR00000024', 'A012','TRANSFER',SYSDATE-40,NULL,195000,195000);
INSERT INTO TRANSACTIONS VALUES('TR00000025', 'A001','VAT','31-DEC-12',70,NULL,6930);
INSERT INTO TRANSACTIONS VALUES('TR00000026', 'A002','VAT','31-DEC-12',29,NULL,2871);
INSERT INTO TRANSACTIONS VALUES('TR00000027', 'A003','VAT','31-DEC-12',500,NULL,49500);
INSERT INTO TRANSACTIONS VALUES('TR00000029', 'A005','VAT','31-DEC-12',550,NULL,54450);
INSERT INTO TRANSACTIONS VALUES('TR00000030', 'A006','VAT','31-DEC-12',800,NULL,799200);
INSERT INTO TRANSACTIONS VALUES('TR00000031', 'A007','VAT','31-DEC-12',100,NULL,9900);
INSERT INTO TRANSACTIONS VALUES('TR00000032', 'A008','VAT','31-DEC-12',100000,NULL,9900000);
INSERT INTO TRANSACTIONS VALUES('TR00000033', 'A009','VAT','31-DEC-12',800000,NULL,79200000);
INSERT INTO TRANSACTIONS VALUES('TR00000034', 'A010','VAT','31-DEC-12',400,NULL,399600);
INSERT INTO TRANSACTIONS VALUES('TR00000035', 'A011','VAT','31-DEC-12',950,NULL,94050);
```



```
INSERT INTO TRANSACTIONS VALUES('TR00000036', 'A012','VAT','31-DEC-12',1950,NULL,193050);
INSERT INTO TRANSACTIONS VALUES('TR00000037', 'A001','TAX','31-DEC-12',7,NULL,6923);
INSERT INTO TRANSACTIONS VALUES('TR00000038', 'A002','TAX','31-DEC-12',2.9,NULL,2868.1);
INSERT INTO TRANSACTIONS VALUES('TR00000039', 'A003','TAX','31-DEC-12',50,NULL,49450);
INSERT INTO TRANSACTIONS VALUES('TR00000041', 'A005','TAX','31-DEC-12',55,NULL,54395);
INSERT INTO TRANSACTIONS VALUES('TR00000042', 'A006','TAX','31-DEC-12',80,NULL,799120);
INSERT INTO TRANSACTIONS VALUES('TR00000043', 'A007','TAX','31-DEC-12',10,NULL,9890);
INSERT INTO TRANSACTIONS VALUES('TR00000044', 'A008','TAX','31-DEC-12',10000,NULL,9890000);
INSERT INTO TRANSACTIONS VALUES('TR00000045', 'A009','TAX','31-DEC-12',80000,NULL,79120000);
INSERT INTO TRANSACTIONS VALUES('TR00000046', 'A010','TAX','31-DEC-12',40,NULL,399560);
INSERT INTO TRANSACTIONS VALUES('TR00000047', 'A011','TAX','31-DEC-12',95,NULL,93955);
INSERT INTO TRANSACTIONS VALUES('TR00000048', 'A012','TAX','31-DEC-12',195,NULL,192855);
INSERT INTO TRANSACTIONS VALUES('TR00000049', 'A001','CHARGES','31-DEC-12',250,NULL,6673);
INSERT INTO TRANSACTIONS VALUES('TR00000050', 'A002','CHARGES','31-DEC-12',250,NULL,2618.1);
INSERT INTO TRANSACTIONS VALUES('TR00000051', 'A003','CHARGES','31-DEC-12',250,NULL,49200);
INSERT INTO TRANSACTIONS VALUES('TR00000053', 'A005','CHARGES','31-DEC-12',250,NULL,54145);
INSERT INTO TRANSACTIONS VALUES('TR00000054', 'A006','CHARGES','31-DEC-12',250,NULL,798870);
INSERT INTO TRANSACTIONS VALUES('TR00000055', 'A007','CHARGES','31-DEC-12',250,NULL,9640);
INSERT INTO TRANSACTIONS VALUES('TR00000056', 'A008','CHARGES','31-DEC-12',250,NULL,9889750);
INSERT INTO TRANSACTIONS VALUES('TR00000057', 'A009','CHARGES','31-DEC-12',250,NULL,79119750);
INSERT INTO TRANSACTIONS VALUES('TR00000058', 'A010','CHARGES','31-DEC-12',250,NULL,399310);
INSERT INTO TRANSACTIONS VALUES('TR00000059', 'A011','CHARGES','31-DEC-12',250,NULL,93705);
INSERT INTO TRANSACTIONS VALUES('TR00000060', 'A012','CHARGES','31-DEC-12',250,NULL,192605);
INSERT INTO TRANSACTIONS VALUES('TR00000061', 'A012','EXPORT LC',SYSDATE-40,50000000,NULL,50192605);
INSERT INTO TRANSACTIONS VALUES('TR00000062', 'A012','EXPORT LC',SYSDATE-30,10000000,NULL,60192605);
INSERT INTO TRANSACTIONS VALUES('TR00000063', 'A012','EXPORT LC',SYSDATE-20,20000000,NULL,80192605);
INSERT INTO TRANSACTIONS VALUES('TR00000064', 'A008','IMPORT LC',SYSDATE-40,NULL,1000000,8889750);
INSERT INTO TRANSACTIONS VALUES('TR00000065', 'A008','IMPORT LC',SYSDATE-30,NULL,1000000,7889750);
INSERT INTO TRANSACTIONS VALUES('TR00000066', 'A008','IMPORT LC',SYSDATE-10,NULL,1000000,6889750);
INSERT INTO TRANSACTIONS VALUES('TR00000067', 'A009','IMPORT LC',SYSDATE-40,NULL,1000000,69119750);
INSERT INTO TRANSACTIONS VALUES('TR00000068', 'A009','IMPORT LC',SYSDATE-30,NULL,1000000,59119750);
INSERT INTO TRANSACTIONS VALUES('TR00000069', 'A009','IMPORT LC',SYSDATE-10,NULL,1000000,49119750);

INSERT INTO VAULT VALUES('V001', 'DHANMONDI BRANCH', 'MD. ZAMIL AHMED', 25000000, 30000000);
INSERT INTO VAULT VALUES('V002', 'SUNAMGONJ BRANCH', 'MS. ANTARA ZERIN',20000000, 40000000);
```

```
INSERT INTO VAULT VALUES('V003', 'KHULNA BRANCH', 'MD. AKHTER HAMID', 13000000, 15000000);

INSERT INTO VAULT_TRANSACTION VALUES('VTR0000001', 'V001',SYSDATE, NULL, 50000000,50000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000002', 'V001',SYSDATE, NULL, 30000000,80000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000003', 'V001',SYSDATE, NULL, 10000000,90000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000004', 'V001',SYSDATE+1, 10000000,NULL,80000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000005', 'V001',SYSDATE+2, 20000000,NULL,60000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000006', 'V001',SYSDATE+3, 30000000,NULL,30000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000007', 'V002',SYSDATE, NULL, 2000000,2000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000008', 'V002',SYSDATE, NULL, 1000000,3000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000009', 'V002',SYSDATE, NULL, 5000000,8000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000010', 'V002',SYSDATE+1, 1000000,NULL,7000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000011', 'V002',SYSDATE+2, 1000000,NULL,6000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000012', 'V002',SYSDATE+3, 2000000,NULL,4000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000013', 'V003',SYSDATE, NULL, 25000000,25000000);
INSERT INTO VAULT_TRANSACTION VALUES('VTR0000014', 'V003',SYSDATE+1, 10000000,NULL,15000000);

COMMIT;
```