

A MULTI-THREAD RETRIEVAL APPROACH TO A
RELATIONAL MULTI-DATABASE
SYSTEM

By

HUI LIN

Bachelor of Science
Beijing Technology University
Beijing, P. R. China
1987

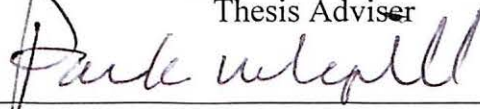
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2004

A MULTI-THREAD RETRIEVAL APPROACH TO A
RELATIONAL MULTI-DATABASE
SYSTEM

Thesis Approved:



Thesis Adviser



Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to thank Dr. G. Hedrick for his constant support and advice that made this work possible. I am grateful for his willingness to discuss the work at any time and his thoughtful criticism. I am grateful to other committee members, Dr. Nohpill Park and Dr. Debao Chen for their helpful suggestions, advisement, and readiness to be my co-referee.

I am especially grateful to Dr. Hong, the Vice Present of BarDyne, Inc., both for supporting the design and implementation of the Multi-database thread access interface kernel and for providing ideas for future directions of this work. In addition, I would like to give special appreciation to my parents for their strong encouragement at times of difficulty, love, support, and understanding throughout the whole process to implement this thesis.

Finally, I would like to thank the Computer Science Department for supporting me during my studies.

TABLE OF CONTENTS

Chapter I Introduction	1
1.1 Motivation.....	2
1.2 Thesis Outline	3
Chapter II Literature Review	4
2.1 Information Retrieval in the Relational Database System.....	4
2.1.1 Relational Database Query Language.....	4
2.1.2 ODBC	6
2.2 Major Object-Oriented Programming Languages	9
2.2.1 C++	9
2.2.2 Microsoft Foundation Class (MFC).....	11
2.3 Multi-database System Introduction.....	12
2.4 Multi-thread and Operating System.....	14
Chapter III Implementation of Multi-database Thread Retrieval Approach	20
3.1 Virtual Database Access Control Module.....	20
3.2 VC++ Implementation of multi-database thread retrieval approach	23
Chapter IV Performance Measurements	37
4.1 Schema and data conflicts in multi-database system.....	37
4.2 Performance Benchmark Application.....	47
4.3 Performance Measurements and Summary	48
4.3.1 Performance Measurements.....	48
4.3.2 Performance Test Summary	55

Chapter V Conclusions and Future Work	56
5.1 The key features of the multi-thread database access strategy	56
5.2 Future Work	57
Appendix A	
Class schema definition of multi-thread retrieval Approach to a Relational multi-database system	58
References	73
Glossary	75

LIST OF TABLES

Table	Page
2.1 Product Table	4
2.2 Supplier Table.....	4
2.3 Dealer Table	4
4.1 Definitions of Database (I) Table Department	38
4.2 Definitions of Database (I) Table Staff	38
4.3 Definitions of Database (I) Table Grade	39
4.4 Definitions of Database (I) Table Course	39
4.5 Definitions of Database (I) Table Student	39
4.6 Definitions of Database (I) Table Enroll	39
4.7 Definitions of Database (II) Table Department	40
4.8 Definitions of Database (II) Table Enroll	40
4.9 Definitions of Database (II) Table Advisor	41
4.10 Definitions of Database (II) Table Grade	41
4.11 Definitions of Database (II) Table Student	41
4.12 Definitions of Database (II) Table Course	42
4.13 Definitions of Database (III) Table Administrator	43
4.14 Definitions of Database (III) Table Student	43

4.15 Definitions of Database (III) Table Department	43
4.16 Definitions of Database (III) Table Employee	44
4.17 Definitions of Database (III) Table Course	44
4.18 Definitions of Database (III) Table Enroll	45
4.19 Definitions of Database (III) Table Grade	45
4.20 Query Performance of Query 1	49
4.21 Query Performance of Query 2	49
4.22 Query Performance of Query 3	49
4.23 Query Performance of Query 4	50
4.24 Query Performance of Query 5	50
4.25 Query Performance of Query 6	50
4.26 Query Performance of Query 7	51
4.27 Query Performance of Query 8	51
4.28 Query Performance of Query 9	51
4.29 Query Performance of Query 10	52
4.30 Query Performance of Query 11	52
4.31 Query Performance of Query 12	52

LIST OF FIGURES

Figure	Page
2.1 Basic Architecture View of ODBC Schema	7
2.2 ODBC Driver Configuration 1	8
2.3 ODBC Driver Configuration 2	9
2.4 Applications accessing several existing databases	14
2.5 Basic Concepts of Single Thread and Multi Thread	16
2.6 Many-to-One Thread Mode	17
2.7 One-to-One Thread Mode	18
2.8 Many-to-Many Thread Mode	19
3.1 Architecture of Multi-database Thread Retrieval Approach	22
3.2 CMDBDatabase Class Schema Definition	58
3.3 CMDBRecordset Class Schema Definition	60
3.4 CMDBSQLThreadTmp Class Schema Definition	64
3.5 CMDBException Class Schema Definition	66
3.6 CMDBGeneralMgr Class Schema Definition	67
3.7 CMDBThreadMgr Class Schema Definition	70
3.8 CMDBThreadQryMgr Class Schema Definition	72

3.9 Multi-database Query Processing	36
4.1 Relationships of Database (I) Tables	40
4.2 Relationships of Database (II) Tables	42
4.3 Relationships of Database (III) Tables	45
4.4 Screen Shot of Benchmark Application	48
4.5 Non-join Query Analyses	53
4.6 Join Query Analyses	53
4.7 Multi-Database Thread Query Analysis	54
4.8 Multi-Database Thread Overall Query Compare.....	54

CHAPTER I

INTRODUCTION

Modern databases have developed dramatically over the past twenty years in terms of the services and functionality they provide as well as the business areas they cover. Relational databases are still the market leader because of their easy query functionality and clear table structure. In a relational database system, the data is bound to a table, and one record (row) does not necessarily stand for a particular object since multi-valued relations must be stored in several database tables. Primary and foreign keys provide a rough approximation: insertion of a new key value in a "primary" (entity) relation approximates the creation of a new object.

In recent years, the value-based network/web system became increasingly complex, dynamic and potentially huge. In order to access and combine information in a logical and coherent manner within easy reach, a single relational database system is imperfect in several respects. The following characteristics of such a database structure are desirable, but unavailable; it should:

- Support the process of wide-area information delivery and management from multi-purpose data sources such as different kinds of databases from different vendors (IBM, Informix, Microsoft, NCR, Oracle, and Sybase), different data files etc.;
- Provide a high level of security control and restricted access to the sensitive data for Enterprise Resource Planning Project (ERP);
- Prevent from data being corrupted (i.e., backed up and organized in such a way as to minimize the risk of accidental deletion/alteration);

- Minimize the size of the database to allow for future growth because of widely-distributed information;

1.1 Motivation

The heavy demands of sharing data across different database platforms have introduced the need for relational multi-database systems (RMDBS). A RMDBS is “a system capable of operating over a network and encompassing a heterogeneous mix of different relational database systems, providing the user with a unified view of distributed and heterogeneous data” [Hurson, 1994];

During the past few years, researchers of RMDBS mainly focused on the following:

- How to define a good model of scalable, parallel RMDBS driven information retrieval engine; [Sheth, 1998]
- How to properly extend relational database query language - Structured Query Language (SQL) to allow both interactive users and application programs to access and modify the data stored in the RMDBS; [Litwin, 1990]

This thesis presents the implementation of a new method to extend access to a large RMDBS. The method is called Multi-thread information retrieval approach using multi-thread concepts and the C++ programming language. The power of this method comes from knowledge and technology of the multi-thread concept together with SQL developed over several decades. This method can be embodied seamlessly in traditional DBMS applications as an interface for creating and managing large amounts of different kind of data efficiently. The advantages of this approach include:

- It provides capabilities allowing safe and persistent storage of data over a network.
- It provides virtual database security control in the thread (the smallest executable unit of a process) mode;
- It provides a way to both populate and to manage complex data structures such as audio data, video data and graphic data without customizing the applications;

In this thesis, we also assume the readers are familiar with the basic concepts of a multi-tasking operating system, the C++ programming language and database query processing as described in introductory literature, for instance the books by [Simon, 1996], [Zaratian, 1996] and [Silberschatz, 2000].

1.2 Thesis Outline

This thesis is organized as follows. Chapter 2 presents related work in the existing literature; Chapter 3 introduces the design and implementation of the multi-database thread retrieval approach; Chapter 4 presents performance measurements with an existing business multi-database system; Finally, Chapter 5 presents conclusions and future work.

CHAPTER II

LITERATURE REVIEW

2.1.1 Relational Database Query Language

In 1970, an IBM researcher, Dr. E. F. Codd, formally defined the core concept of a relational model in his paper “A Relational Model of Data for Large Shared Data Banks” [Codd, 1970]. This article generated many commercial applications. In the relational model presented by Dr. Codd, the interrelations among data are defined by a data structure called a relation. A relation is viewed as a named table, its rows are called tuples, and its columns are called attributes. Each attribute is associated with a name and a data type [Ullman, 1988]. The following tables (sets) can be used to understand the relational data schema.

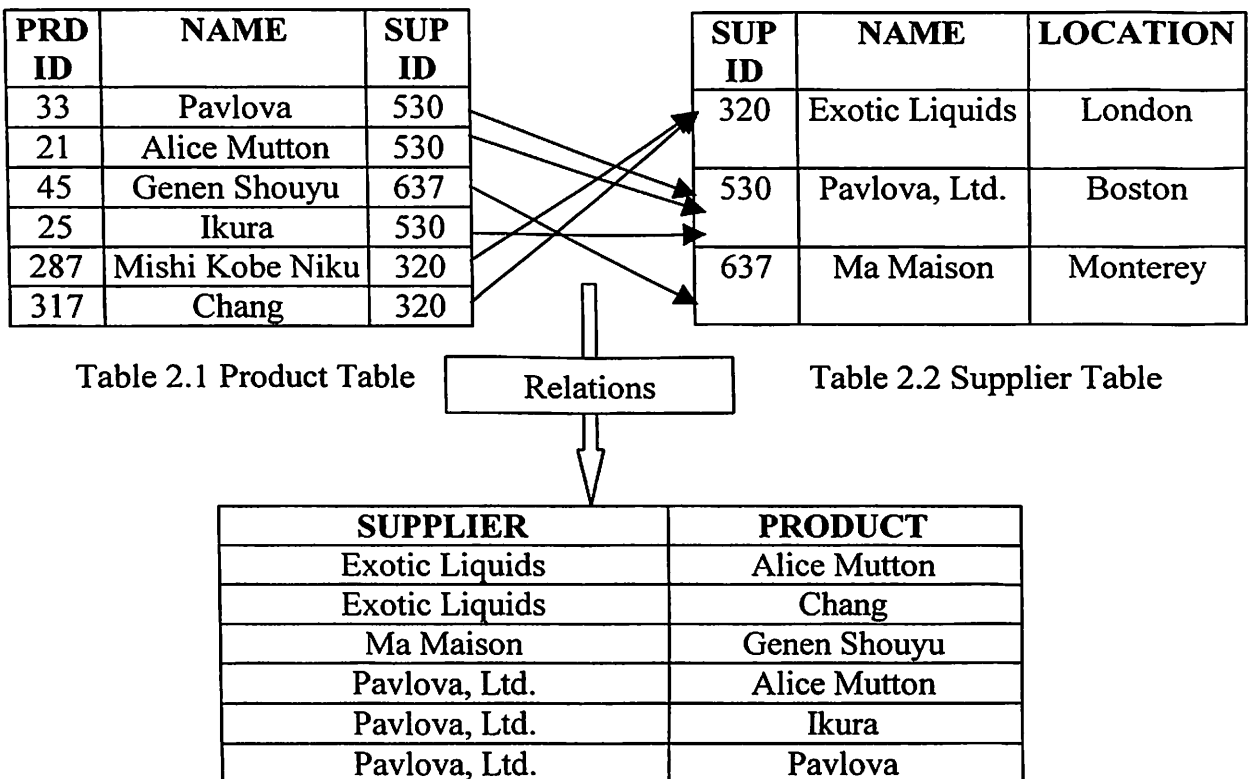


Table 2.3 Dealer Table

Table 2.1 and 2.2 shows an example with a PRODUCT table and a SUPPLIER table. The rows in these two tables represent individual product and supplier information. The PRDID (integer) attribute of the PRODUCT table and the SUPID (integer) attribute of the SUPPLIER table are the primary keys and their value is unique in each row. The SUPID attribute of the PRODUCT table is a foreign key that takes on values of the primary key SUPID of the SUPPLIER table. Table 2.3 shows a relationship table, called DEALER, used to represent associations between products and suppliers. Each row in the DEALER relationship table is a unique product-supplier pair [Elmasri, 1989].

From 1974 to 1975, based on the relational schema from Dr. Codd, IBM implemented a project named "System/R" to develop SEQUEL (Structured English Query Language). Then, it was completely rewritten to include multi-table and multi-user features and re-named as "SQL" (Structured Query Language) after 1977. Since that time, various SQL standards have developed over the years. SQL became a mature, powerful, and versatile relational database query language to query, define, manipulate and control data in the relational database. A relational database is a collection of related data stored in a computer, and a RDBMS (Relational Database Management System) is a collection of programs to manipulate relational databases [Date, 1993].

Since SQL relies on concepts such as table, indexes, keys, rows, columns, and set theory, the SQL statement returns information in the tabular format of the relational model. The most important SQL keywords are **INSERT**, **SELECT**, **DELETE**, **UPDATE**, **WHERE** and **ORDER BY** used to create, search, delete and update data in the database based on a declarative query condition. For example, the SUPPLIER table (Table 2.2) has a column NAME (string) that contains the name of the supplier and a column SUPID that contains the code of the supplier. The

PRODUCT table (Table 2.1) also has a column NAME (string) that specifies the name of the product and a column SUPID that specifies the code of the product's supplier. The typical queries that manipulate data in the Supplier table can be specified as following:

```
INSERT INTO SUPPLIER (NAME, SUPID, LOCATION) VALUES ("ABC", 1, "NY")
```

```
SELECT * FROM SUPPLIER ORDER BY NAME
```

```
DELETE FROM SUPPLIER WHERE NAME = 'ABC'
```

```
UPDATE SUPPLIER SET NAME = 'ABCD' WHERE SUPID = 1
```

In order to obtain the DEALER table (table 2.3), the SQL query (inter join) can be as follows:

```
SELECT SUPPLIER.NAME, PRODUCT.NAME FROM SUPPLIER, PRODUCT WHERE  
PRODUCT.SUPID = SUPPLIER.SUPID ORDER BY SUPPLIER.NAME [Date, 1993].
```

2.1.2 ODBC

Open database Connectivity (ODBC) is a standard database access method developed by Microsoft Corporation to enable any application to communicate with any database management system (DBMS) [Simon, 1996]. ODBC uses a middle layer named "Database Driver Manager" to translate the application's data queries into commands that the DBMS can understand. In other words, both the application and the DBMS must be ODBC-compatible; the DBMS must be capable of responding to ODBC query commands issued from application. Most major database server vendors and the suppliers of many desktop products now provide an ODBC interface through which the end-user has access to centrally stored data directly from the desktop products. With the leverage Microsoft currently has on the market place, ODBC now dominates client/server database connectivity. Figure 2.1 below shows a schematic view of the ODBC interface.

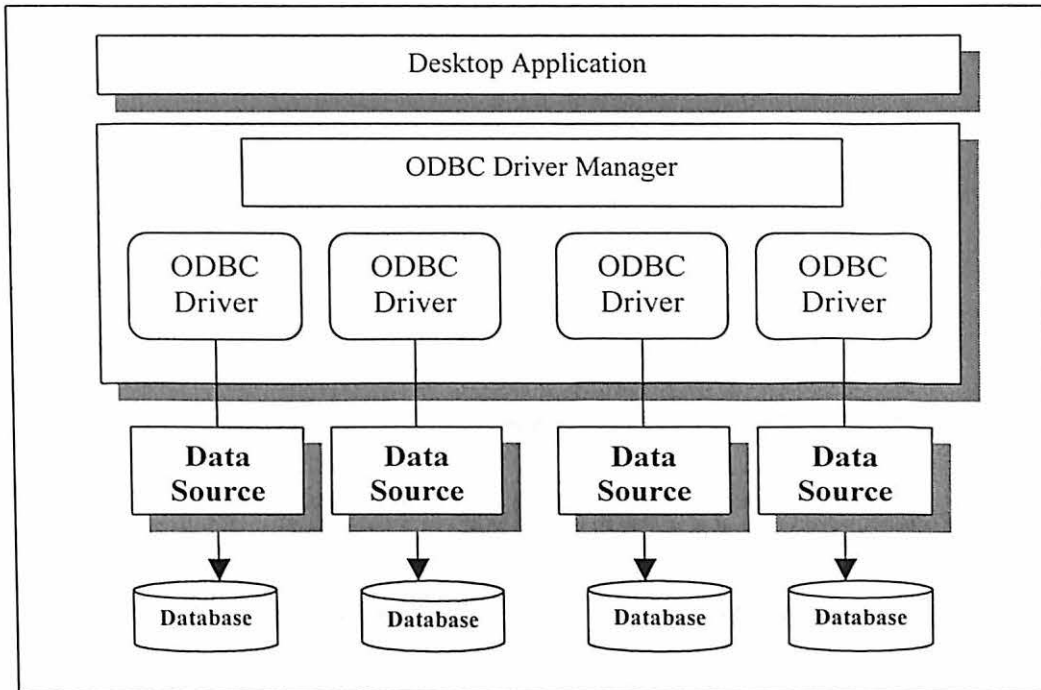


Figure 2.1 Basic Architecture View of ODBC Schema

The Driver Manager is composed of a set of standard Microsoft-supplied DLLs (Dynamic Link Library); it loads the appropriate database driver to response calls from desktop applications. The data is accessed in relational form by passing SQL command strings.

There are two main types of ODBC driver configurations. The first configuration manipulates database using native vendor's networking software (e.g. Ingres, Oracle). ODBC driver manager passes the SQL request to the database vendor network software that communicates with the network database server. This configuration is illustrated in Figure 2.2 below.

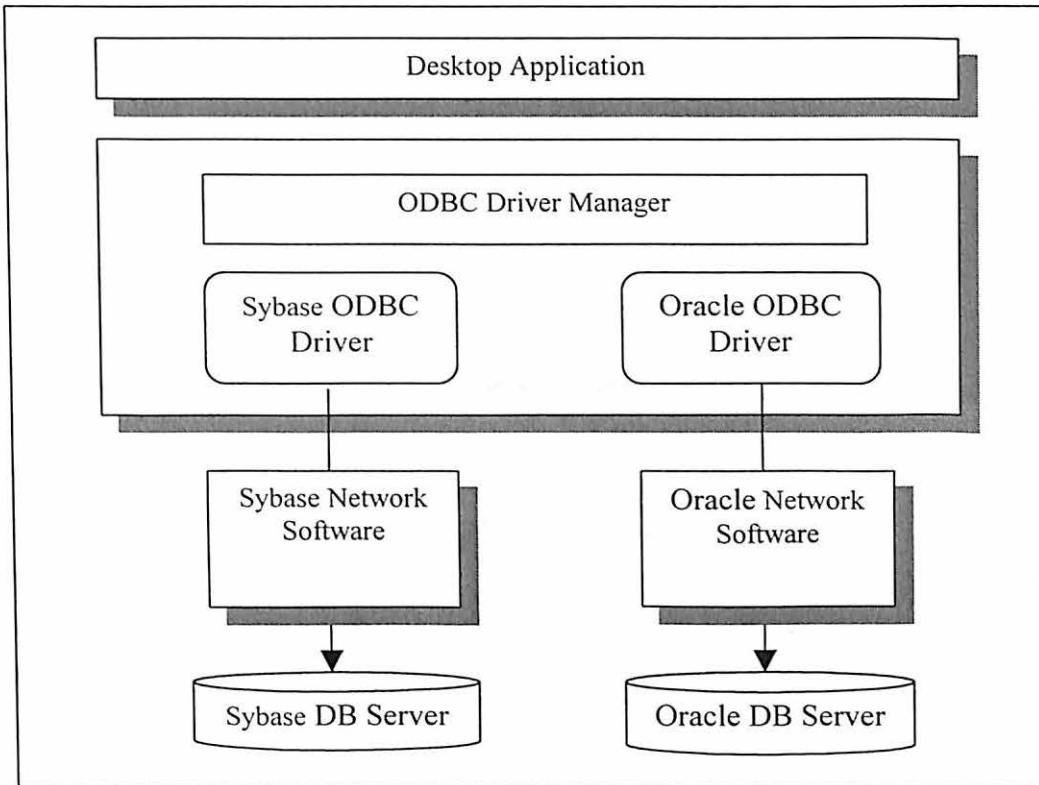


Figure 2.2 ODBC Driver Configuration 1

Oracle drivers in this configuration are from Microsoft and Oracle, the Sybase drivers are distributed by Sybase.

The second configuration does not require the database vendors networking software.

These drivers have one or more components that run on the client machine and also a component that runs on the database server. One particular advantage of this configuration is only one driver is used on the client, regardless of the number of database servers connected to process SQL requests generated by front-end applications. A typical configuration is illustrated in Figure 2.3 below.

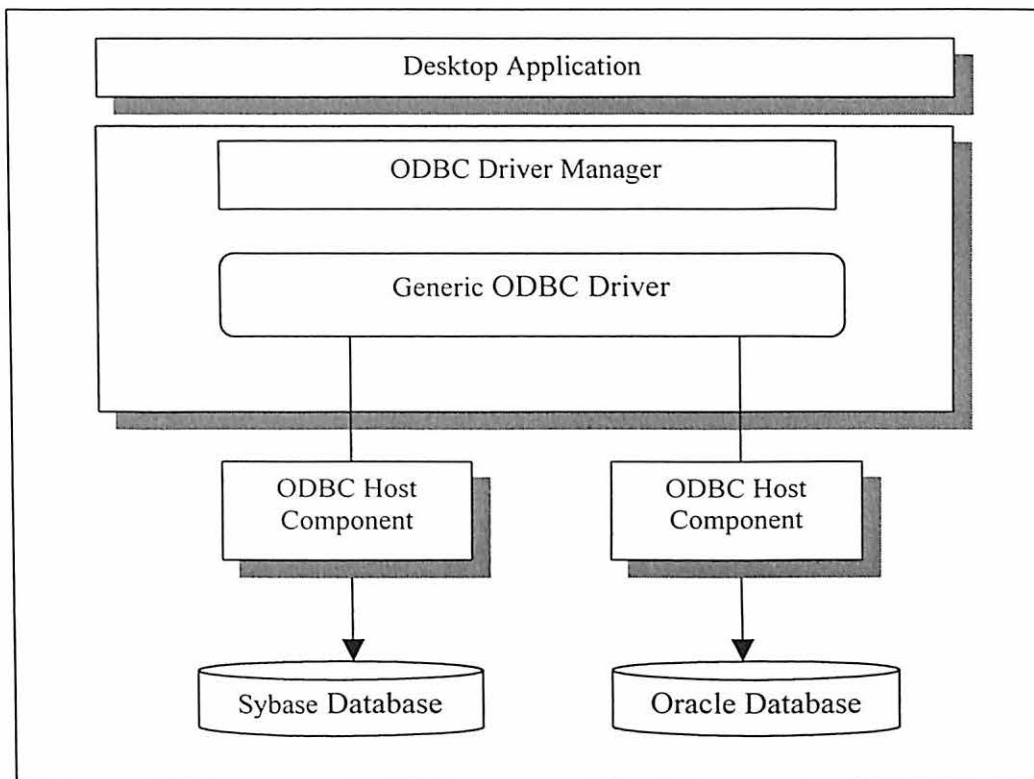


Figure 2.3 ODBC Driver Configuration 2

2.2 Major Object-Oriented Programming Languages

2.2.1 C++

During 1969-1973, Bell Labs developed the UNIX operating system, and at the same time, Dennis Ritchie designed and developed C as a programming language originally for the implementation of UNIX operating system on a PDP-11 computer [Zaratian, 1998]. Bjarne Stroustrup at Bell Labs wrote C++ as an extension of C during 1983-1985. He added classes and object-oriented features to C and formed so called "C with Classes". After its first release, C++ developed significantly with the power and efficiency of C, classes and object-oriented concepts.

C++ has been standardized by ANSI (The American National Standards Institute), BSI (The British Standards Institute), DIN (The German national standards organization), several other national standards bodies, and ISO (The International Standards Organization). "ARM C++" added exceptions and templates, and ISO C++, added RTTI, namespaces, and a standard library. With C++, programmers could improve the quality of code they produced and reusable code was easier to write.

The major concepts in C++ are encapsulation, inheritance, polymorphism, class and object [Khoshafian, 1993].

Encapsulation allows the programmer to control both the scope of names, and access to functions and/or values stored inside an object;

Inheritance is a mechanism by which new classes are defined from existing classes. Subclasses inherit operations of their parent class. Inheritance is the mechanism by which reusability is facilitated. It is a mechanism for sharing behavior and attributes between classes;

Polymorphism means that some code or operations or objects behave differently in different contexts.

For example, the + (plus) operator in C++:

23 + 431 : integer addition

"cde" + "abcde" : string concatenation

5.6 + 122.0 : floating point addition

Typically, when the term polymorphism is used with C++, it refers to using virtual methods;

Class is an encapsulation of variable and function declarations, called data members and function members respectively. Variables can have any type, but they must have unique names

within the scope of the class. Functions can have the same name, even within the scope, but must have different signatures. A class can also have default constructor and destructor that take no arguments and are used to initialize and terminate an object;

Object is an instance of a class. In more common terms, an object is a variable of a given data type. In C++, a class declaration also declares a new type. Objects have a lifetime, either governed by a local scope, or they can be created or destroyed dynamically;

2.2.2 Microsoft Foundation Class (MFC)

The Microsoft Foundation Class Library (MFC) was a C++ class library first released in 1992 (with Microsoft C++ 7.0) to support application development on Microsoft Windows. Written in C++, MFC provides much of the code necessary for managing windows, menus, and dialog boxes; performing basic input/output; storing collections of data objects; and so on. The MFC Library consists of numerous classes that are thin wrappers for high level Application Programming Interfaces (APIs) such as WinSock and ODBC. All the Win32 Kernel, GDI, and User Objects have associated MFC classes. The MFC library is called a vertical library, as it uses class inheritance heavily with very little C++ templates.

The MFC library can either be linked statically or dynamically into the desktop applications. If it is linked dynamically, then the application is very small. At runtime, the application uses MFC classes through the MFC dynamic link libraries. These DLLs are usually found in the Windows SYSTEM subdirectory. If many applications are MFC based and use MFC dynamically, the tremendous amount of hard drive space will be saved. Also, if the library is already loaded when one application is running, then the next application that uses MFC

ynamically loads faster.

MFC is most often used in GUI (Graphic User Interface) applications; it can be used to develop any type of application. MFC shortens development time; makes code more portable without reducing programming freedom and flexibility; provides easy access to the user-interface elements and technologies, like Active, OLE, and Internet programming. Furthermore, MFC simplifies database programming through Data Access Objects (DAO) and Open Database Connectivity (ODBC), and network programming through Windows Sockets [Zaratian, 1998].

2.3 Multi-Database System Introduction

The computing environments have become increasingly distributed through the use of Internet and other computer communication networks. What we are experiencing is an ever-increasing access to more or less structured information that is both very dynamic and changing continuously. In this environment it is becoming increasingly critical to develop methods for building systems that combine relevant data from many sources, and then present them in a form that is comprehensible for users. It is important to develop tools that facilitate the efficient development and maintenance of information systems in a highly dynamic and distributed environment. The area of distributed databases deals with design and management of uniform databases whose contents are distributed transparently over several database nodes in a computer network. Parallel databases deal with high performance databases whose data automatically is distributed over many internal data servers. The area of multi-database systems deals with managing and querying data from collections of heterogeneous databases.

A multi-database system (MDBS) integrates a set of autonomous and heterogeneous local database. In such a system, each local database consists of a local DBMS and a database; global

Transactions are executed independently under the control of the MDDBS; local transactions are submitted directly to a local DBMS by local applications [Hurson, 1994]. An MDDBS should provide a mechanism to manage transactions globally; users in a MDDBS can access information from multiple sources through global transactions. However, global transactions are long-lived and involve operations on multiple and autonomous local databases. Moreover, MDDBS do not have any information about the existence and execution order of local transactions.

Figure 2.4 illustrates the situation that multi-database systems in use. Various database systems on the market are shared among application domains that differ in service support and price. In general, a number of databases distributed over a computer network are managed by some local DBMS and each of them is used by one or more applications. The multi-database system is heterogeneous and autonomous and applications can access data from any of these component databases. The solution for data integration in a MDDBS is to construct a front-end system that supports a single common data model and a single global query language on top of different types of existing local databases. The front-end system plus the underlying database systems is the so-called multi-database system.

Local database schema is the conceptual schema of a MDDBS. For each local schema, there is a corresponding component schema. The component schema represents the same information as the local schema, but the common data model is used instead of the data model of the component database system. A query against a component schema is translated to queries against the underlying local schema. The results of these queries are then processed to form an answer to the initial query [Hurson, 1994]. An integrated or global schema is an integration of multiple local schemas. The global schema makes it possible to access data from multiple databases as though it was stored in a single database.

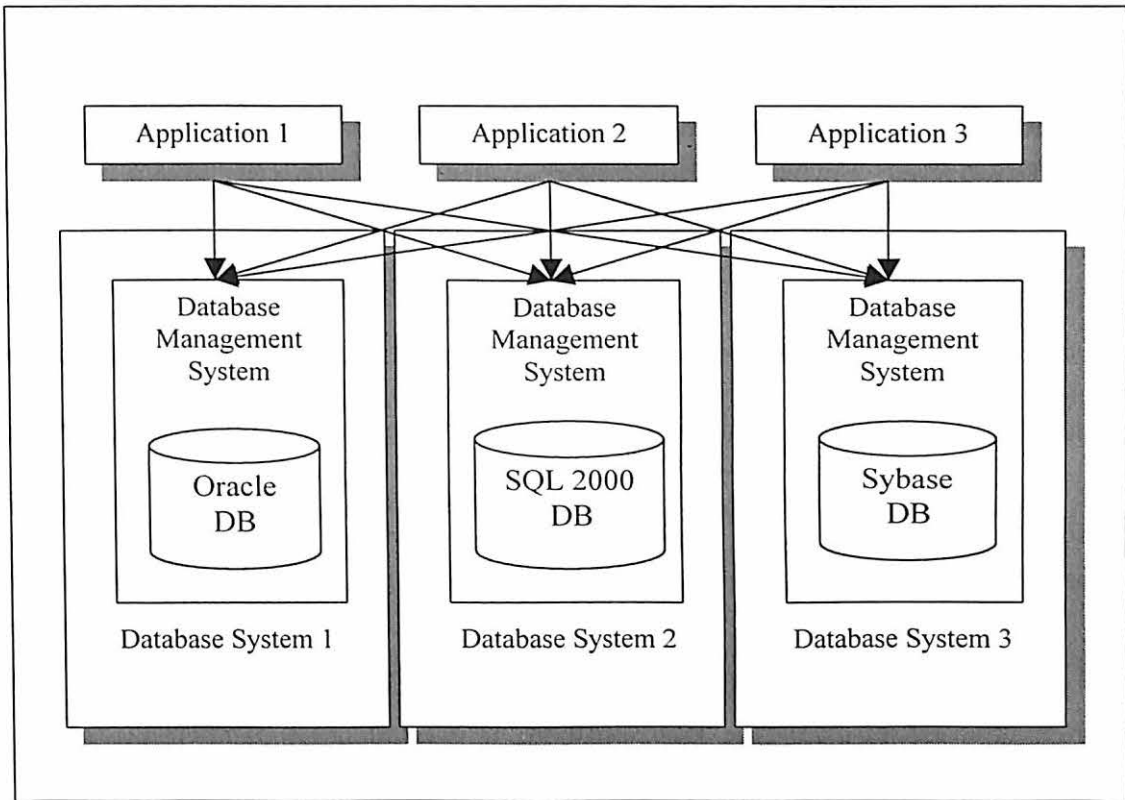


Figure 2.4 Applications accessing several existing databases

2.4 Multi-thread in Operating System

The concept of an application having multiple threads has been around for some time.

Various modern microprocessor-based operating systems such as Apple, Windows 2000, IBM OS/2 and UNIX have supported threads or some form of add-on thread package for years. They provide an extensive library and system call facility to support this feature. In these operating systems, a thread, sometimes, referred to as lightweight task, can be considered as a separate flow of execution, each thread operating in parallel with the other threads. Different threads may be executing the same code sequence or different code sequences. Conceptually, a thread is a basic

unit of resource utilization that comprises a thread ID, a program counter (PC), a register set, and a stack [Silberschatz, 2000]. Due to the minimal context state requirements, threads have a very fast context switch time. Since threads operate within the application's context, each thread has full application global access and shares the same address space, file access paths, and other system resources associated with the application. If a process has multiple threads of control, it can perform more than one task at a time.

The two most important thread benefits are ease of logical program structure and performance. Program structure is simplified because each application task can be coded as an almost independent subsystem. If tasks interact and/or share resources, they must use synchronization objects, which the API provides. Performance is enhanced since some threads can make progress while one or more other threads may be in a wait state. For example, a keyboard thread waiting for a keystroke does not have to block all other code executions.

Concurrent multi-thread execution means two or more threads are in process at the same time. If one thread blocks for some reason, another thread from the same program executes in its place. This feature is especially relevant to the I/O bound application. Parallelism occurs when two or more threads execute simultaneously across multiple processors, utilizing the power of multi-processor systems. The power of multi-threads not only resides in the ability to have multiple flows of execution within an application, but also in the clever way these multiple threads may interact and interrelate. Multi-threaded benefits, particularly performance, are directly related to efficient thread management in the multithreaded application development.

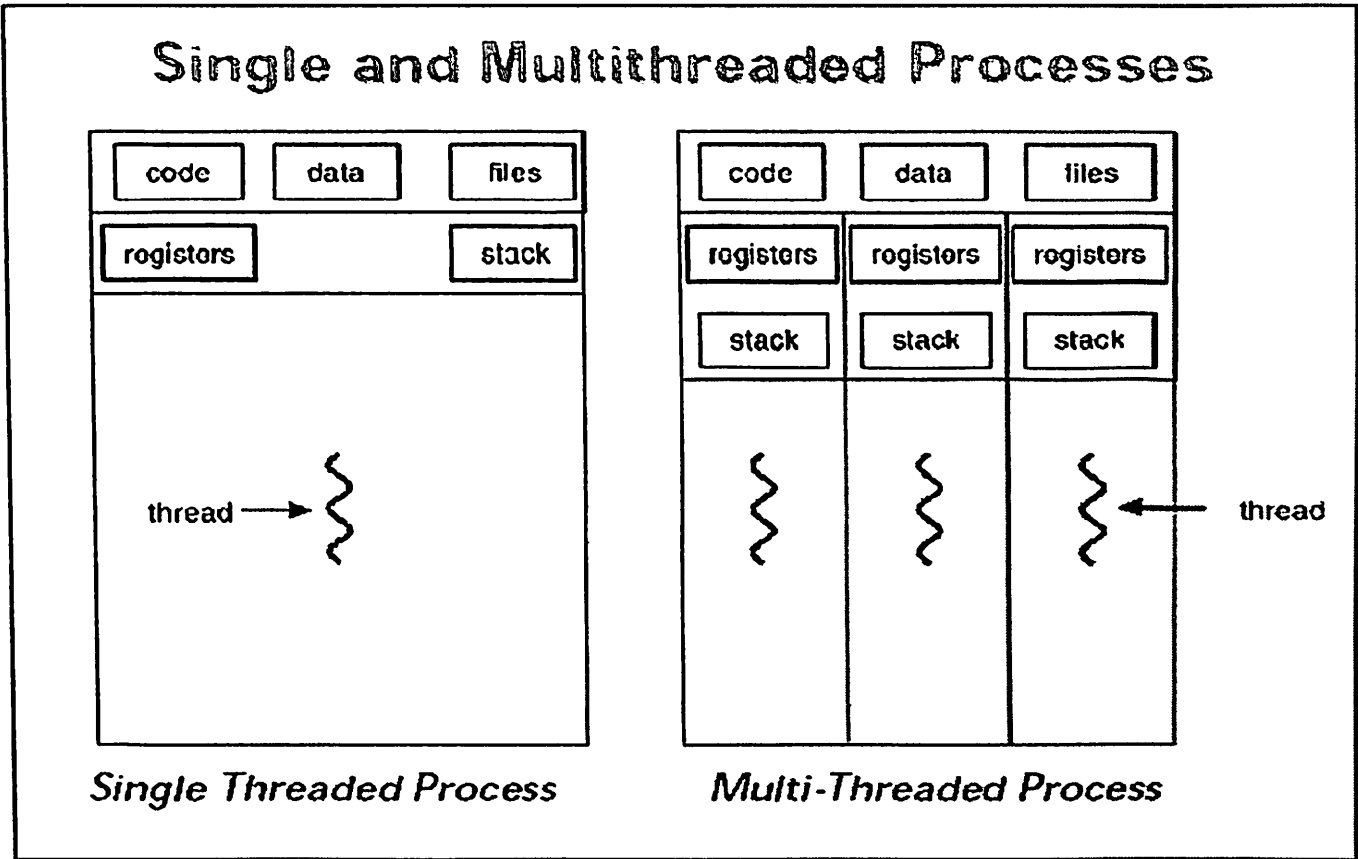


Figure 2.5 Basic Concepts of Single Thread and Multi Thread

Many operating systems support both user and kernel threads. There are three types of threading implementations [Silberschatz, 2000]:

1. Many-to-One (Figure 2.6)

Used on systems that do not support kernel threads, many user-level threads map to a single kernel thread. The benefits are thread management is done in user space, and so it is efficient; the drawback is multiple threads are unable to run in parallel on multiprocessors. A typical system of this type is Solaris 2 (the current is Solaris 9);

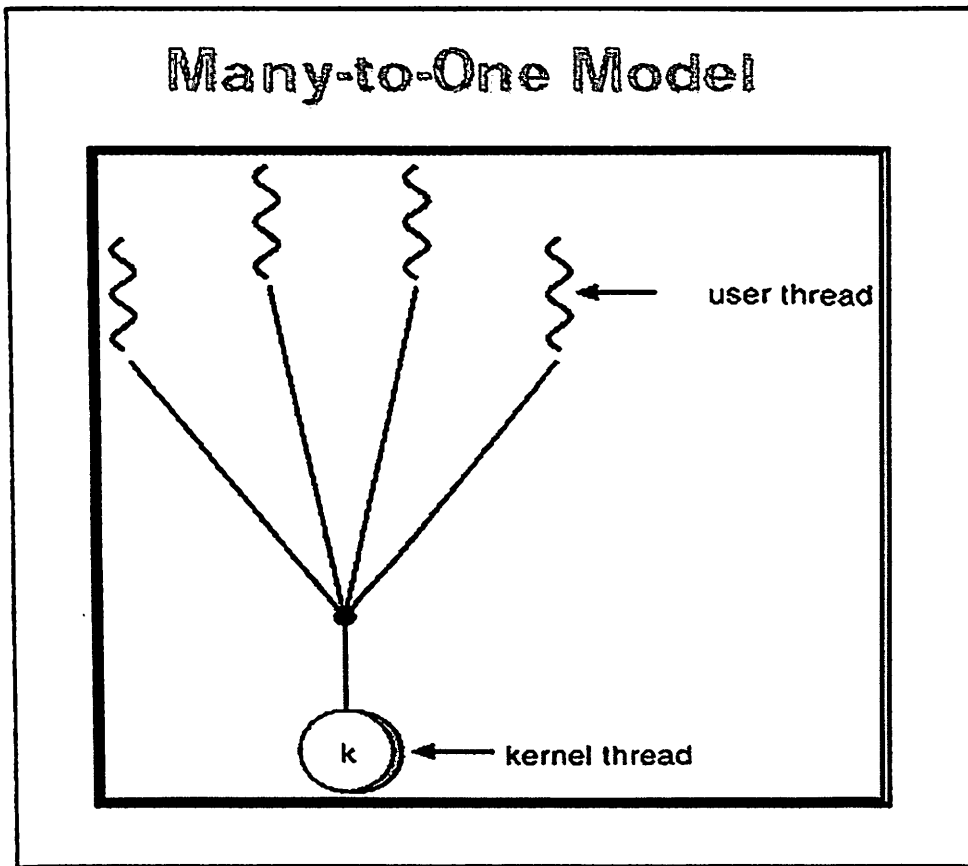


Figure 2.6 Many-to-One Thread Mode

2. One-to-One (Figure 2.7)

Each user-level thread maps to a kernel thread. The benefits are more concurrency and multiple threads can be run in parallel on multi-processors. The drawback is overhead of thread creation and management. The typical systems are Windows 95/98/NT/2000 and IBM OS/2;

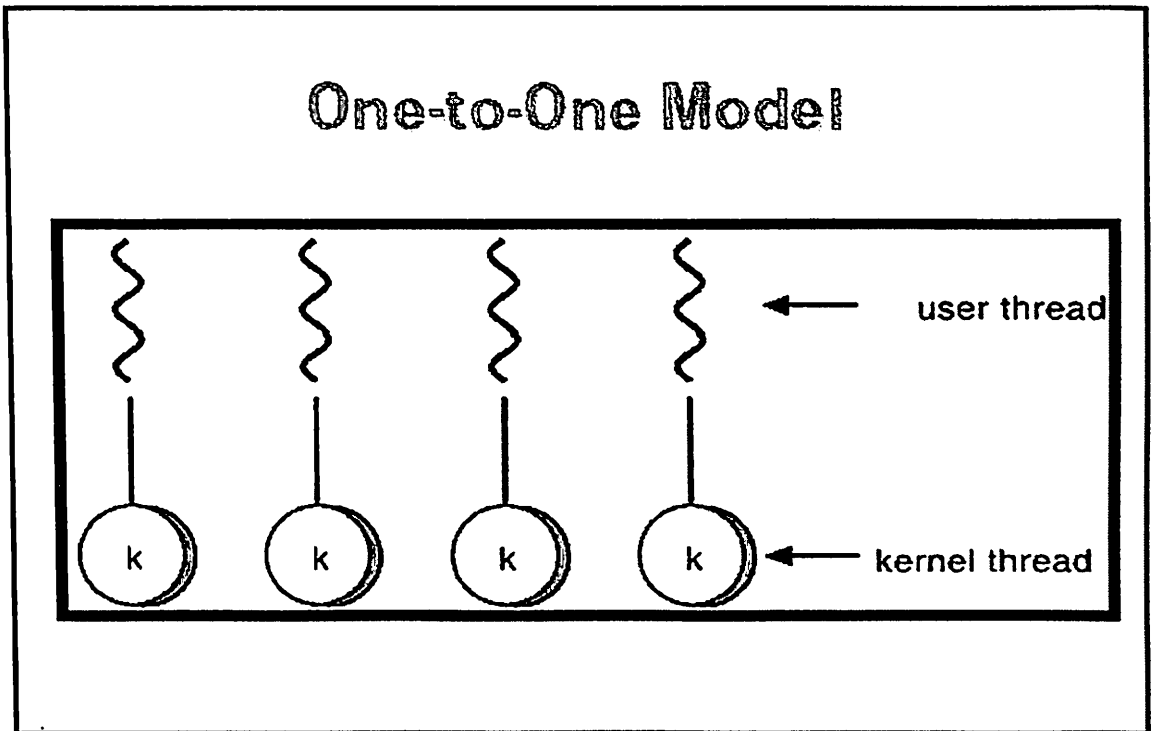


Figure 2.7 One-to-One Thread Mode

3. Many-to-Many (Figure 2.8)

Many-to-Many thread systems multiplex many user level threads to a smaller or equal number of kernel threads. The benefits are the model suffers from neither the shortcomings of many-to-one nor one-to-to models. Concurrently, there are no expensive threads allow the operating system to create a sufficient number of kernel threads. The typical systems are Solaris 2 and Windows NT/2000;

Many-to-Many Model

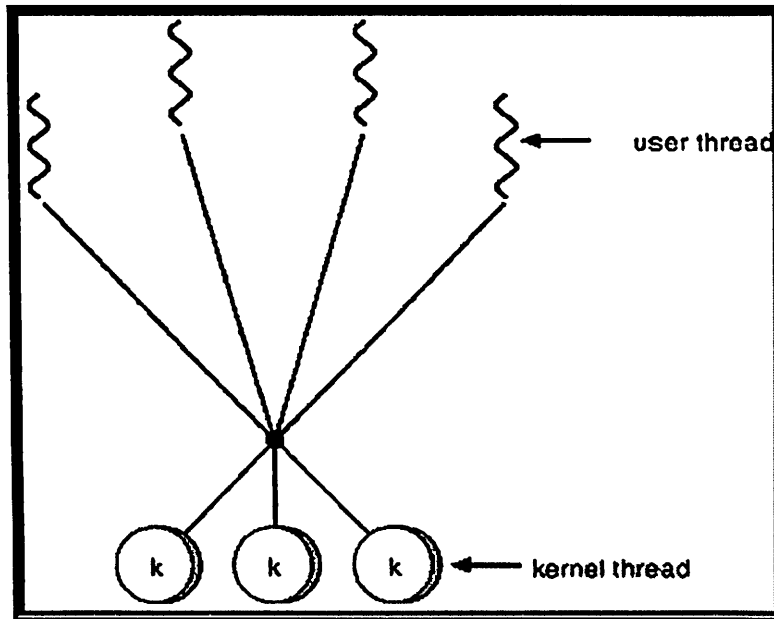


Figure 2.8 Many-to-Many Thread Mode

CHAPTER III

ARCHITECTURE AND IMPLEMENTATION OF MULTI-DATABASE THREAD RETRIEVAL APPROACH

This chapter presents the major architectural principles and implementation designs underlying the schema of the multi-database thread retrieval approach (Figure 3.1). Section 3.1 gives detailed descriptions of the architecture of three modules:

- Virtual Database Access Control Module
- Multi-database Thread Allocate /De-Allocate Module
- Multi-database Thread Query Module

Section 3.2 provides all generic class data types and functionality definitions implemented using Microsoft Visual C++ (MFC 6.0).

3.1 Architecture Descriptions of Three Multi-Databases

Thread Retrieval Approach Modules

- Virtual Database Access Control Module

Virtual Database Access Control Module is a general query controller in the architecture. As soon as this module receives SQL queries submitted by front-end database application, it starts to analyze the SQL syntax and determines table names and query criteria. Next, it invokes module service (functions) to pass table names to Multi-database Thread Allocate /De-allocate Module to obtain local (child) database connection information (thread mode). Based on the local (children) database connection information and query criteria obtained, this module generates a series of

local SQL queries and sends them to the Multi-database Thread Query Module to obtain the query results (recordset) then returns query results to the application;

- Multi-database Thread Allocate /De-Allocate Module

Multi-database Thread Allocate /De-Allocate Module are used for the following functionalities:

- Allocate local database connection threads;
- Release local database connection threads;
- Maintain a memory registry table of each local database name, all local database table/field names and database connections threads;
- Respond to the requests from Virtual Database Access Control Module to provide local (children) database connections (thread mode)

- Multi-database Thread Query Module

When the Virtual Database Access Control Module passes local SQL and database connections to the module, it uses the Open Database Connectives (ODBC) Driver to contact to the Local (Children) Database Pool for SQL query results (recordset). If query results (recordset) come from different local databases and need inter-join query (a SQL statement is used to combine the data contained in two relational database tables based upon a common attribute.), then this module transfer these recordsets into a temporary database for the inter-join query. Finally, this module returns all query results (recordset) to the Virtual Database Access Control Module.

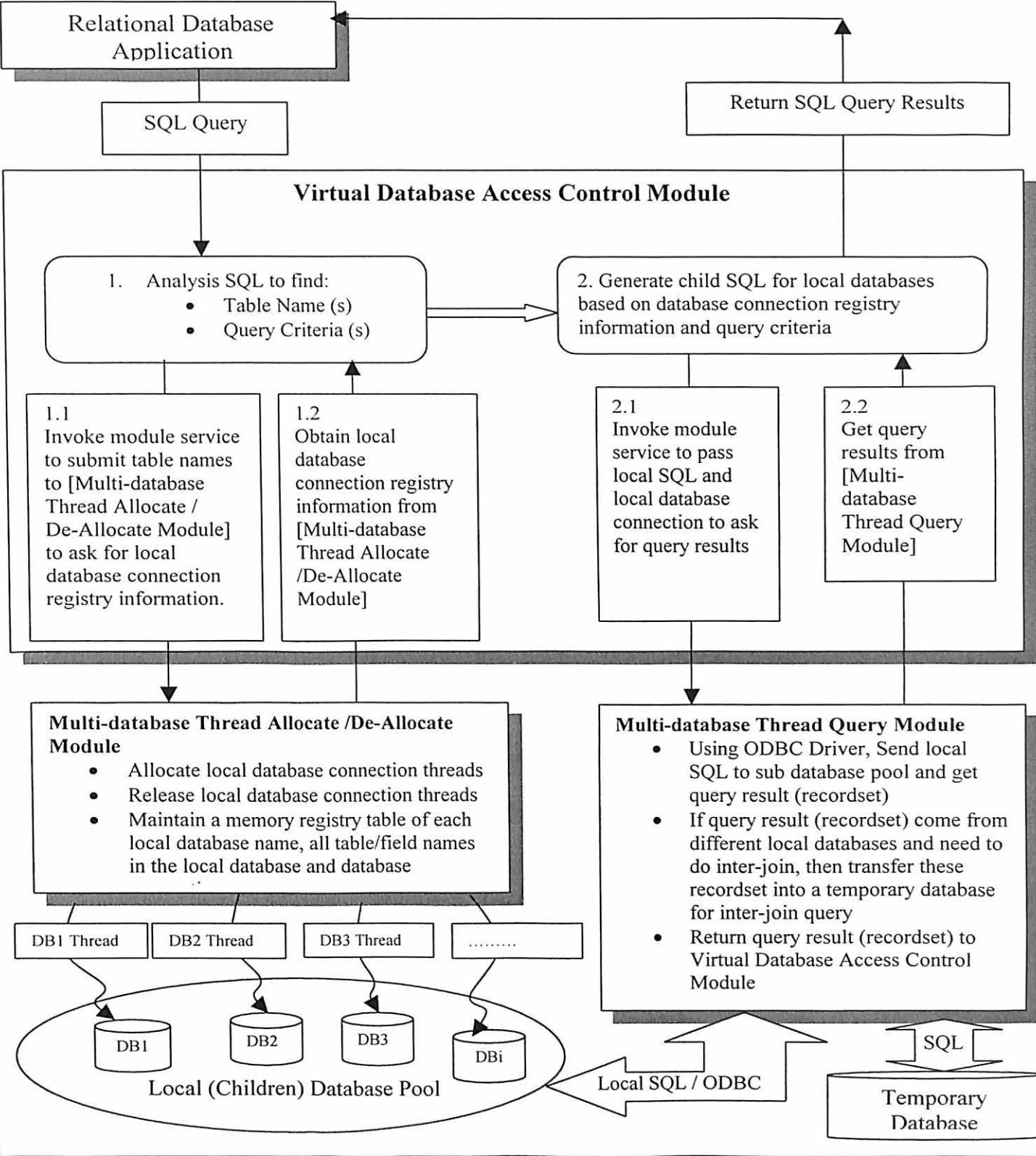


Figure 3.1 Architecture of Multi-database Thread Retrieval Approach

3.2 VC++ Implementation of multi-database thread retrieval approach

Class Name: CMDBDatabase

Member Functions: Figure 3.2 (Appendix A)

Member Variables: Figure 3.2 (Appendix A)

Class Description:

CMDBDatabase is a class derived from class CDatabase; CDatabase is a MFC class provides the functionality required to access records stored in a database. In some ways, this class also is analogous to the stream classes used to access standard data files, application can directly create an instance of this class in order to connect to the database. CDatabase is composed of a set of high-level API for accessing databases through driver presents a single interface to a wide variety of database systems, APIs is more complex than using the MFC classes. Any application that can use ODBC can query and work with data in any database that has an ODBC driver, and that is virtually every database system available. ODBC drivers are Examples include Microsoft SQL Server, Microsoft Access, Borland® dBASE®, and xBASE. As well, ODBC drivers are available to connect to databases ranging from Oracle™ to Microsoft Excel™.

The CDatabase wrapper class encapsulates the CMDBDatabase object, and all connection information is contained within it. Template class CMDBSQLThreadTmp will declare and instantiate the object of CMDBDatabase, then store this connection information for all processing related to the database and assign it to Multi-database Thread Allocate /De-Allocate Module registry table (memory table), therefore, it is an ODBC thread-safe connection to any number of

databases. A CMDBDatabase object represents a database connection through which system can operate on the data source hosted by multi-database management system (MDBMS).

To use CMDBDatabase, CMDBSQLThreadTmp constructs a CMDBDatabase object and call its Open member function. This opens a database connection. Then construct CMDBRecordset objects for operating on the connected data source, pass the CMDBRecordset constructor a pointer to CMDBDatabase object. When system finish using the connection, call the Close member function and destroy the CMDBDatabase object. Close closes any CMDBRecordset object that has not closed previously.

Using CMDBDatabase makes data access in Multi-database Thread Allocate /De-Allocate Module easier. It provides a flexible multi-database access method for working with table structure, saving queries for reuse, and so on, in many cases, for multiple users.

Class Name: CMDBRecordset

Member Functions: Figure 3.3 (Appendix A)

Member Variables: Figure 3.3 (Appendix A)

Class Description:

Like the ODBC MFC wrapper class, the CMDBRecordset is managed and maintained by the class 'CRecordSet'. There are many similarities in nature to the ODBC wrapper, and it would appear that applications programmatically perform the same functions. For each SQL statement

hat is executed on any database, a CMDBRecordset must exist to receive the data; therefore, a CMDBRecordset object will be instantiated for each query or action.

CRecordset is a MFC class that provides the remaining functionality required to access records stored in a database. Typically, this class will not be used directly; instead, CMDBRecordset, the derived class from the CRecordset, will be used to describe specific data from multi-database. In some ways, it is analogous to the read/write operations in a stream class.

The CMDBRecordset object is used to hold a set of records from a multi-database table. A CMDBRecordset object is consist of records and columns (fields). In multi-database Thread Query Module, this object is the most important and the most used object to manipulate data from a database. The two basic member methods associated with CMDBRecordset object are Open and Close used to retrieve and discard data from a database. When CMDBRecordset opens a recordset, the current record pointer will point to the first record and the BOF and EOF properties are False. If there are no records, the BOF and EOF property are TRUE. CMDBRecordset object can support two types of updating:

Immediate updating - all changes are written immediately to the database once you call the Update method.

Batch updating - the provider will cache multiple changes and then send them to the database with the UpdateBatch method.

In CMDBRecordset, there are four different cursor types defined:

Dynamic cursor - Allows user to see additions, changes, and deletions by other users.

Keyset cursor - Like a dynamic cursor, except that one user cannot see additions by other users, and it prevents access to records that other users have deleted. Data changes by other users will still be visible.

Static cursor - Provides a static copy of a recordset for a user to use to find data or generate reports. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when system opens a client-side CMDBRecordset object.

Forward-only cursor - Allows a user to only scroll forward through the recordset. Additions, changes, or deletions by other users will not be visible.

The cursor type can be set by the CursorType property or by the CursorType parameter in the Open method.

Class Name: CMDBSQLThreadTmp

Member Functions: Figure 3.4 (Appendix A)

Member Variables: Figure 3.4 (Appendix A)

Class Description:

CMDBSQLThreadTmp, the class template derived from MFC class CWinThread, is a user-interface thread template commonly used to handle CMDBDatabase object connection and respond to database events independently of threads executing other portions of the application. This template class can make any member function of any class executed in different thread, without requiring to define any static or global functions; the other class member methods require inheritance in the same way that CWinThread under MFC work.

CMDBSQLThreadTmp objects allow multiple threads within a given application using the Win32 api CreateEvent() call, they typically exist for the duration of the thread. The member

variable 'm_bAutoDelete' can be set to FALSE to modify a CMDBSQLThreadTmp object's behaviors. A CMDBSQLThreadTmp object can be declared and implemented using the DECLARE_DYNCREATE and IMPLEMENT_DYNCREATE macros from CWinThread. There are two general types of threads that CMDBSQLThreadTmp supports: worker threads and user-interface threads. Worker threads have no message pump: for example, a thread that performs background calculations in a spreadsheet application. User-interface threads have a message pump and process messages received from the system; both of these two types wrap a thread handle, ID and most commonly used thread methods, like Pause(), Suspend() and Resume(), plus, it has value semantics and full copy constructor and assignment operator.

CMDBSQLThreadTmp has three advantages:

1. Allow user to start a thread on any member function of any object

by providing it a certain signature (type define):

T& thObject, void (T::*pfnOnFunRunning)(),

template can accomodate both __cdecl and __stdcall funtions as well.

2. It is type safe.

3. It has pluggable thread creators.

The CMDBSQLThreadTmp class is the major feature of class CMDBThreadMgr used to create a CMDBDatabase thread-safe connection. Usually, class CMDBSQLThreadTmp is passed by CMDBThreadMgr with a CMDBDatabase dynamic object on its stand-alone global function named 'allocate_MDBThread()', and, thread-local data of the CMDBThreadMgr to maintain thread-specific information is managed by CMDBSQLThreadTmp objects. It is the goal of

CMDBSQLThreadTmp that CMDBSQLThreadTmp impose as little overhead as possible in accessing those appropriate class features.

Class Name: CMDBException

Member Functions: Figure 3.5 (Appendix A)

Member Variables: Figure 3.5 (Appendix A)

Class Description:

During the time program is executing, function calls could bring the system three outcomes:

1. Function executes normally and returns;
2. Function passed by some mistakes in arguments or inappropriate contextes causes a system error;
3. When abnormal outside conditions such as low memory or I/O errors happened, the function was influenced and handled by the exception.

Therefore, the using of exceptions can be summed up in one guideline: exception is especially for handling errors rather than for predictable normal cases; when an abnormal condition happened, system should throw an exception. Exception handling has several benefits as follows:

- It allows programmers to separate error-handling code from normal code. programmers can surround the code that they expect to execute most of the time with a try block;

- Programmers can place error-handling code in catch clauses -- code that they don't expect to get executed often, if ever. This arrangement has the nice benefit of making the "normal" codeless cluttered.

Class `CMDBException` is a derived child class based on MFC's class `CException`.

`CException` is the base class for all exceptions in the Microsoft Foundation Class Library that is indicated by being thrown (`THROW`) and caught (`CATCH`). Because `CException` is an abstract base class system cannot create `CException` objects directly; the `CMDBException` object (object of `CException`-style derived class) can be created using `IMPLEMENT_DYNAMIC` macro.

Class `CMDBException` is centered around the three keywords: try, catch, and throw, the general purpose of which is to attempt to execute code and handle unexpected exceptional conditions thrown from classes such as `CMDBDatabase`, `CMDBRecordset`, `CMDBSQLThread`, `CMDBThreadMgr`, `CMDBThreadQryMgr` and `CMDBGeneralMgr`. This consists of utilizing a try block (with its attendant handlers). The basic exception control structure code is described below:

```
try  
{  
  A series of statements;  
}  
catch(CMDBException e)  
{  
  // catch all exceptions and cleanup  
  throw;  
}
```

The code checks to see if a resource is available and if not, throws an exception. The handler for the CMDBException presumably does something meaningful about the exceptional state. From above sample control code, the typical use of exception is to prevent continued operation if the program cannot obtain the required resource. Another example of this use of exceptions is new(), which will throw the standard exception bad memory allocation if the required amount of memory is not available.

Class Name: CMDBGeneralMgr

Member Functions: Figure 3.6 (Appendix A)

Member Variables: Figure 3.6 (Appendix A)

Class Description:

Class CMDBGeneralMgr presents an implementation of Virtual Database Access Control Module. Because each SQL stores the information about the database relationships along with their (possible) associations with tables, columns, and joins, CMDBGeneralMgr passes SQL to its member functions such as 'analysis_SQL' and performs following actions:

- Semantically evaluates and analyzes the SQL statement together with the lexical interpretation;
- Find out relevant information about all table and column such as name, keyword criteria (SELECT, FROM, WHERE, ORDER BY and so on) and other specific matching criteria (<, >, =, NOT, NULL and so on);

- Syntactic checking of the SQL constraints; semantics analysis and lexical analysis may record data type of a field; do constraint incompatibility check, i.e., check that the different constraints of a same table or field that do not conflict with one another.
- Constraint consistency check uses the error handler (CMDBException) to set a tag to avoid repeating semantic or lexical errors;
- Invoke member functions related to Class CMDBThreadMgr (Multi-database Thread Allocate /De-Allocate Module) to obtain all child database connection information;
- Generate a set of local SQL statements based on information about child database connection, keyword criteria and specific matching criteria requesting through an SQL query; constraint refers to an invariant or to an operation;
- Invoke member functions to send all local SQL statements to Class CMDBThreadQryMgr (Multi-database Thread Query Module) to obtain SQL query results; pass back query results to the front-end application;

Class CMDBGeneralMgr is an interface between front-end database application and back-end multi-database. It allows efficient multi-database access control and protects databases from random and harmful attacks involving searches for network or system. Two classes accompany with CMDBGeneralMgr are: CMDBThreadMgr and CMDBThreadQryMgr. All of these three classes together perform the role of an intermediary between the applications and relational data sources and resolve heterogeneous multi-data source access from a single query.

With Class CMDBGeneralMgr, security access control policies and decisions can be set based on who is making the request, where they are making the request from and what data they are. Every child database has an access control list (ACL) that specifies the level of access that users and servers have to that database. Although access levels are the same for users and servers, those assigned to users determine the tasks that they can perform in a database, while those assigned to servers determine what information within the database the servers can replicate.

Class Name: CMDBThreadMgr

Member Functions: Figure 3.7 (Appendix A)

Member Variables: Figure 3.7 (Appendix A)

Class Description:

Class CMDBThreadMgr presents an implementation of Multi-database Thread Allocate /De-Allocate Module. Multi-database Thread Allocate /De-Allocate Module is a collection (pool) of multi-database (CMDBDatabase) connection in multi-threaded mode (CMDBSQLThreadTmp). The CMDBThreadMgr's member function 'allocate_MDBThread' is responsible to create connection of all child databases that are running in thread mode. A database connection in thread mode is a unit of multi-database that can be scheduled by the operating system.

Each database connection (thread mode) has one of six states associated with it at any given time during its life. They are:

READY: indicates the connection is ready

STANDBY: indicates it will be the next connection to run on a multi-database system.

RUNNING: indicates connection execution on a database. Execution will continue until the connection (thread) is preempted by a higher priority connection, it terminates, its time quantum ends, or it calls a blocking system call.

WAITING: indicates the connection is waiting for some request to be completed.

TRANSITION: indicates the connection is waiting for the resources necessary for execution.

TERMINATED: indicates the connection has finished execution.

CMDBThreadMgr schedules each child database connection (thread) based on a multi-level priority registry table checked by the CMDBGeneralMgr. It includes all child database name, all table names in the child database and thread ID and starts at the highest and goes to the lowest level depends on database connection activates. CMDBThreadMgr traverses the priority registry table searching for all database connection (thread) information (such as a list of all valid SQL table names) that is needed in the SQL statement, when one is found, CMDBThreadMgr must determine whether there is a valid connection available, if no connection is currently available, but the connection (thread) with a higher priority will preempt the one with lowest priority and begin execution on that database. If it is unable to preempt a connection it will be skipped, and CMDBThreadMgr will continue its traversal.

Based on CMDBSQLThreadTmp, CMDBThreadMgr is able to automatically locate database connection threading issues, such as race conditions, stalls, and deadlocks. Therefore,

CMDBThreadMgr made the multi-database system suitable for high transaction rates and large volumes of data and possesses outstanding security and availability characteristics.

Class Name: CMDBThreadQryMgr

Member Functions: Figure 3.8 (Appendix A)

Member Variables: Figure 3.8 (Appendix A)

Class Description:

Class CMDBThreadQryMgr presents an implementation of Multi-database Thread Query Module. The goal of CMDBThreadQryMgr is to provide Virtual Database Access Control Module an interface to retrieve data, filter data, and manipulate result sets from multi-database by passing transact local child SQL queries and child database connection in thread mode (CMDBSQLThreadTmp). During the time of processing queries, if the final result sets is the inter-join results of different child databases, CMDBThreadQryMgr will regroup and re-summarize data by reloading all child-query results into a temporary database. Figure 3.1 describes how multi-database queries are processed with inter-join among the multi-database tables.

CMDBThreadQryMgr provides Multi-database Thread Query Module three types for executing SQL queries; “sqlquery”, “sqltable” and ‘sqloutput”. All three connect to multi-database and execute SQL query statements; the only difference between them is how the result from the query is handled.

CMDBThreadQryMgr defines the multi-database SQL query interface. It consists of two major functional parts:

First, CMDBThreadQryMgr call function 'Init ()' to detect connection to each child database engine; if there is connection failed it raised a CMDBException warning else, upon success, it sets global variables in the child SQL name space to valid function names from each SQL statement. The global variable represent the multi-database handle, it make CMDBThreadQryMgr possible to access the database handle to execute more specific requests of the database. The parameter passed to connect specifies which database module to use (ODBC) and the registered name of the child database to open. If multi-database was password protected then CMDBThreadQryMgr need to supply those additional parameters as well;

Second, CMDBThreadQryMgr defines the child SQL statement query schema; processes and manages all local SQL statements that are passed by Virtual Database Access Control Module. CMDBThreadQryMgr executes the each SQL query statement and evaluates its results; generates a list of summary names of the field extracted from the rows of the result of the query; reorganizes summary data for the columns from different child recordsets by using aggregate functions for inter-join if needed; generates control-break reports and returns final query results. CMDBThreadQryMgr assumes all necessary multi-databases to be required.

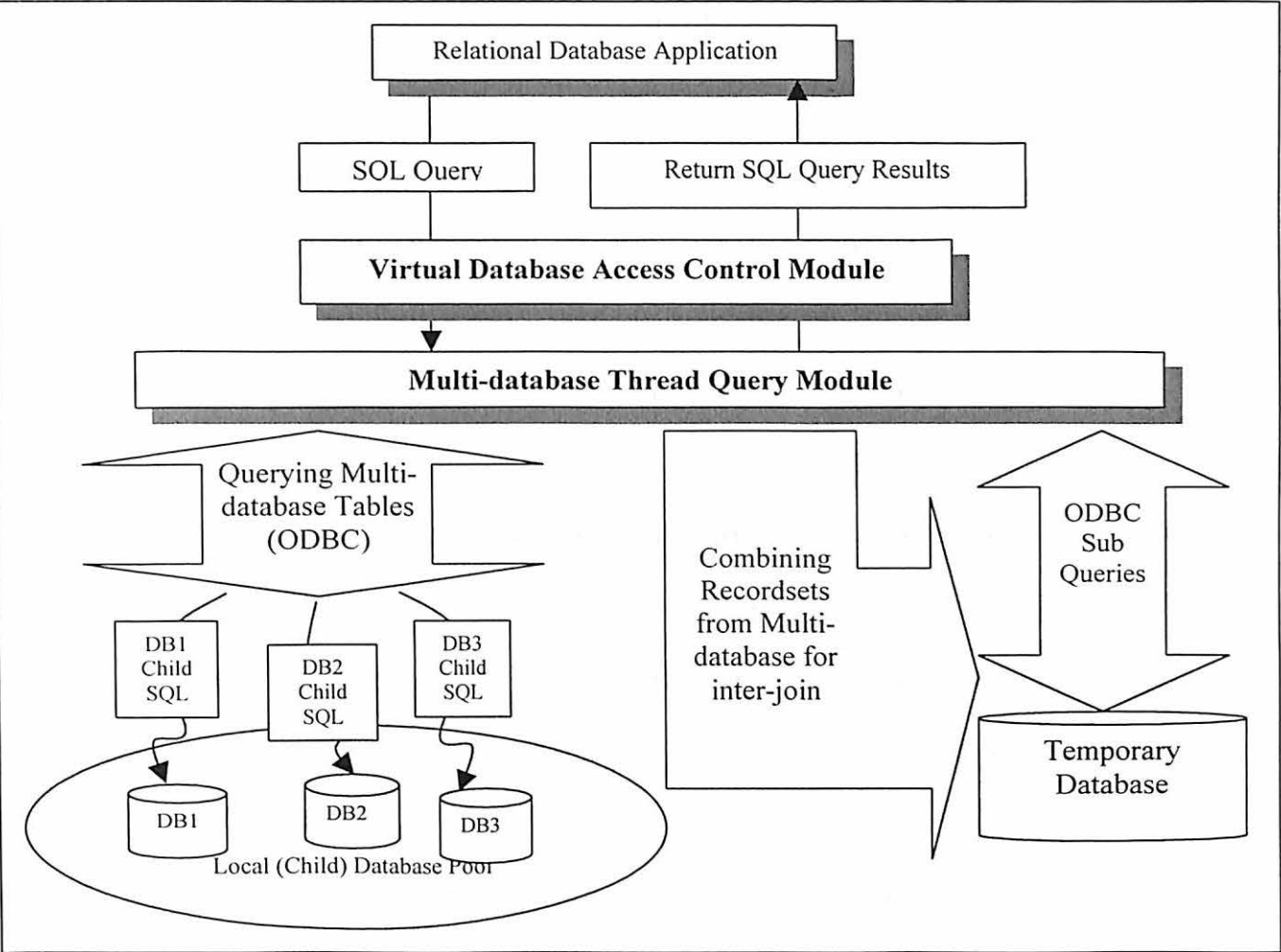


Figure 3.9 Multi-database Query Processing

CHAPTER IV

PERFORMANCE MEASUREMENTS

This chapter describes a methodology for evaluating the performance of thread retrieval approach to a relational multi-database system in a multi-user environment. Two main factors that affect multi-database transaction throughput in a multi-user environment are identified:

1. Access time in multi-database views;
2. Degree of data sharing among simultaneously executing transactions;

Several typical SQL queries will be employed to construct a benchmark program that will evaluate the overall system performance under a wide variety of workloads. Finally, This chapter will present the results of applying thread retrieval approach to the test database system.

4.1 Schema and data conflicts in multi-database system

The test multi-database system and hardware are provided by Quality PC Company, which located at Tulsa, Oklahoma. This system is a general information management system in a medical school, because of offices autonomy, the system consists of two MS SQL 2000 and one MS Access 2000 that located in three campuses: South Campus, North Campus and Main Campus. For the sake of reorganization, the school wishes to combine information across the different campuses.

This chapter will discuss general classification of schematic conflicts that arise in the test multi-database system, and how the thread retrieval approach is applied on this database system using a benchmark application described in sub chapter 4.2.

The following schemas define the architectures of all component databases:

Database (I): Name: NorthCampus; Type: MS SQL 2000

Table Name: Department (Primary key: ID; Foreign key: NONE)		
Field Name	Data Type	Description
ID	Number	ID number
Name	Text	Department name

Table 4.1 Definitions of Database (I) Table Department

Table Name: Staff (Primary key: ID; Foreign key: Dept)		
Field Name	Data Type	Description
ID	Number	ID number
Fname	Text	First name
Mname	Text	Mid name
Lname	Text	Last name
Add1	Text	Home Address 1
Add2	Text	Home Address 2
City	Text	City name
State	Text	State name (US)
Zip	Text	Zip code
Phone1	Text	Phone number (O)
Phone2	Text	Phone number (H)
Fax	Text	Fax number
CellNumber	Text	Mobil
Pager	Text	Pager number
EMail	Text	Contact Email
WebSite	Text	Personal homepage
Login	Text	Login name
Pwd	Text	Login password
Dept	Number	Department ID
Active	Boolean	Retire Status

Table 4.2 Definitions of Database (I) Table Staff

Table Name: Grade (Primary key: NONE; Foreign key: CID, SID, STID)		
Field Name	Data Type	Description
STID	Number	Student ID
SID	Number	Staff ID
CID	Number	Course ID
Grade	Text	A~E
GradeDate	Date/Time	Date and Time of Grade

Table 4.3 Definitions of Database (I) Table Grade

Table Name: Course (Primary key: ID; Foreign key: Dept, SID)		
Field Name	Data Type	Description
ID	Number	Course ID
Name	Text	Course Name
Location	Text	Course Location
MON	Text	Monday
TUE	Text	Tuesday
WED	Text	Wednesday
THU	Text	Thursday
FRI	Text	Friday
SAT	Text	Saturday
SUN	Text	Sunday
Dept	Number	Department ID
SID	Number	Staff ID

Table 4.4 Definitions of Database (I) Table Course

Table Name: Student (Primary key: ID; Foreign key: Dept)		
Field Name	Data Type	Description
ID	Number	ID number
FName	Text	First name
MName	Double	Middle name
LName	Number	Last name
EMail	Text	Student Email
Dept	Number	Department ID

Table 4.5 Definitions of Database (I) Table Student

Table Name: Enroll (Primary key: NONE; Foreign key: CID, STID)		
Field Name	Data Type	Description
STID	Number	Student ID
CID	Number	Course ID
Credits	Number	Credits
EnDate	Date/time	Enroll Date
EnMemo	Text	Enroll Memo

Table 4.6 Definitions of Database (I) Table Enroll

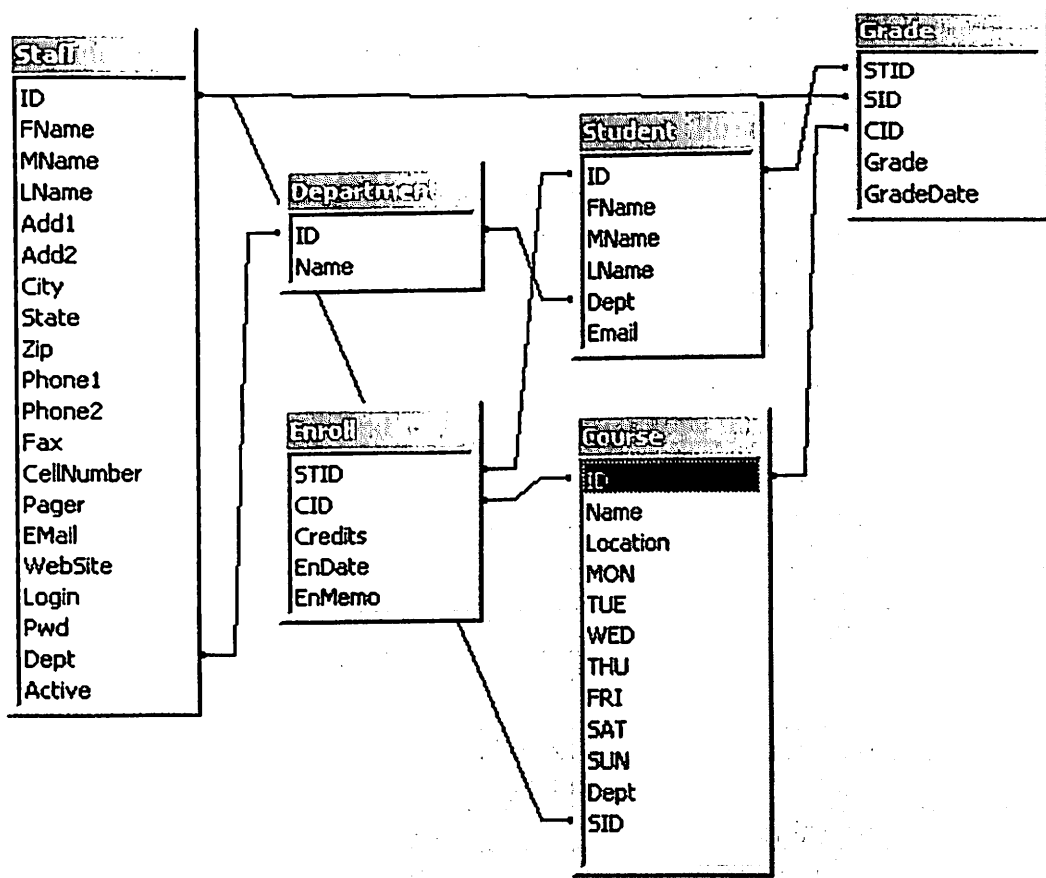


Figure 4.1 Relationships of Database (I) Tables

Database (II): Name: SouthCampus; Type: MS ACCESS 2000

Table Name: Department (Primary key: ID; Foreign key: NONE)		
Field Name	Data Type	Description
ID	Number	Department ID
Name	Text	Department name

Table 4.7 Definitions of Database (II) Table Department

Table Name: Enroll (Primary key: NONE; Foreign key: CID, STID)		
Field Name	Data Type	Description
STID	Number	Student ID
CID	Number	Course ID
Credits	Number	Credits
EnDate	Date/time	Enroll Date
EnMemo	Text	Enroll Memo

Table 4.8 Definitions of Database (II) Table Enroll

Table Name: Advisor (Primary key: ID; Foreign key: Dept)		
Field Name	Data Type	Description
ID	Number	ID number
FName	Text	First name
MName	Text	Mid name
LName	Text	Last name
Add	Text	Home address
City	Text	City name
State	Text	State name (US)
Zip	Text	Zip code
Phone	Text	Phone number (O/H)
EMail	Text	Email address
Login	Text	Login name
Pword	Text	Login password
Dept	Number	Department ID
Active	Boolean	Retire Status

Table 4.9 Definitions of Database (II) Table Advisor

Table Name: Grade (Primary key: NONE; Foreign key: CID, AdvrID, STID)		
Field Name	Data Type	Description
STID	Number	Student ID
AdvrID	Number	Advisor ID
CID	Number	Course ID
Grade	Text	1~5
GradeDate	Date/Time	Date and Time of Grade
Description	Memo	Memo

Table 4.10 Definitions of Database (II) Table Grade

Table Name: Student (Primary key: ID; Foreign key: Dept)		
Field Name	Data Type	Description
ID	Number	ID
FName	Text	First name
MName	Double	Middle name
LName	Number	Last name
EMail	Text	Email address
Dept	Number	Department ID
Description	Memo	Memo

Table 4.11 Definitions of Database (II) Table Student

Table Name: Course (Primary key: ID; Foreign key: Dept, AdvrID)		
Field Name	Data Type	Description
ID	Number	Course ID
Name	Text	Course Name
Location	Text	Course Location
MON	Text	Monday
TUE	Text	Tuesday
WED	Text	Wednesday
THU	Text	Thursday
FRI	Text	Friday
SAT	Text	Saturday
SUN	Text	Sunday
Dept	Number	Department ID
AdvrID	Number	Advisor ID
Description	Memo	Memo

Table 4.12 Definitions of Database (II) Table Course

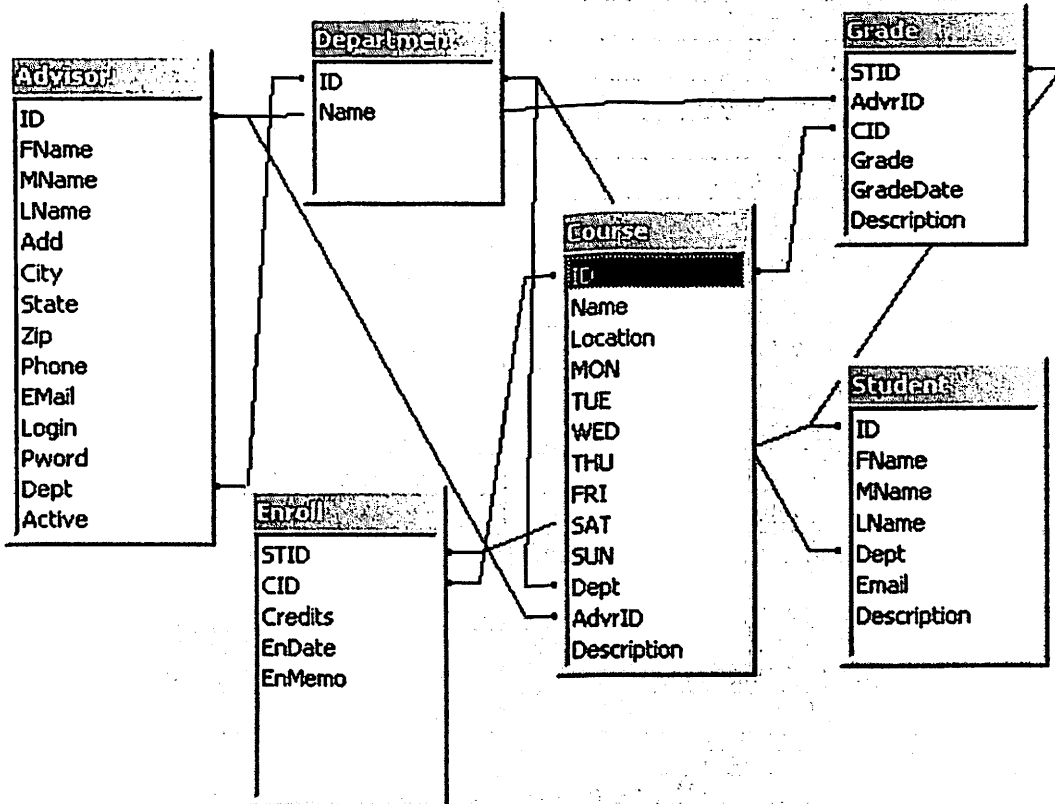


Figure 4.2 Relationships of Database (II) Tables

Database (III): Name: MainCampus; Type: MS SQL 2000

Table Name: Administrator (Primary key: ID; Foreign key: NONE)		
Field Name	Data Type	Description
ID	Number	Administrator ID
FN	Text	First name
MN	Text	Middle name
LN	Text	Last name
Phone	Text	Work phone
Mobil	Text	Mobil Phone Number
EMail	Text	Email address
Login	Text	System Login Name
Pwd	Text	System Login Password

Table 4.13 Definitions of Database (III) Table Administrator

Table Name: Student (Primary key: ID; Foreign key: Dept)		
Field Name	Data Type	Description
ID	Number	ID
FName	Text	First name
MName	Text	Middle name
LName	Text	Last name
Address	Text	Home address
City	Text	City
State	Text	State Name
Zip	Text	Zip code
Phone	Text	Home Phone
DOB	Date	Date of Birth
Sex	Text	Male/Fmale
Email	Text	Email address
Dept	Number	Department ID
Active	Boolean	Registry Status
Description	Text	Memo

Table 4.14 Definitions of Database (III) Table Student

Table Name: Department (Primary key: ID; Foreign key: NONE)		
Field Name	Data Type	Description
ID	Number	ID
Name	Text	Department Name

Table 4.15 Definitions of Database (III) Table Department

Table Name: Employee (Primary key: ID; Foreign key: Dept)		
Field Name	Data Type	Description
ID	Number	Employee ID
FN	Text	First name
MN	Text	Middle name
LN	Text	Last name
Address 1	Text	Home Address
Address 2	Text	Work Address
City 1	Text	City of Home
City 2	Text	City of Job
State 1	Text	State of Home
State 2	Text	State of job
Zip 1	Text	Zip code of Home
Zip 2	Text	Zip code of Job
Phone 1	Text	Home Phone
Phone 2	Text	Work Phone
EMail	Text	Email address
Dept	Number	Department ID
Active	Boolean	Job Status

Table 4.16 Definitions of Database (III) Table Employee

Table Name: Course (Primary key: ID; Foreign key: Dept, EMPID)		
Field Name	Data Type	Description
ID	Number	Course ID
Name	Text	Course Name
Location	Text	Course Location
MON	Text	Monday
TUE	Text	Tuesday
WED	Text	Wednesday
THU	Text	Thursday
FRI	Text	Friday
SAT	Text	Saturday
SUN	Text	Sunday
Dept	Number	Department ID
EMPID	Number	Employee ID
Comments	Text	Comments

Table 4.17 Definitions of Database (III) Table Course

Table Name: Enroll (Primary key: NONE; Foreign key: CID, STID)		
Field Name	Data Type	Description
STID	Number	Student ID
CID	Number	Course ID
Credits	Number	Credits
EnDate	Date/time	Enroll Date
EnMemo	Text	Enroll Memo
Comments	Text	Comments

Table 4.18 Definitions of Database (III) Table Enroll

Table Name: Grade (Primary key: NONE; Foreign key: CID, EMPID, STID)		
Field Name	Data Type	Description
STID	Number	Student ID
EMPID	Number	Advisor ID
CID	Number	Course ID
Grade	Text	1~5
GDate	Date/Time	Date and Time of Grade
Comments	Memo	Memo

Table 4.19 Definitions of Database (III) Table Grade

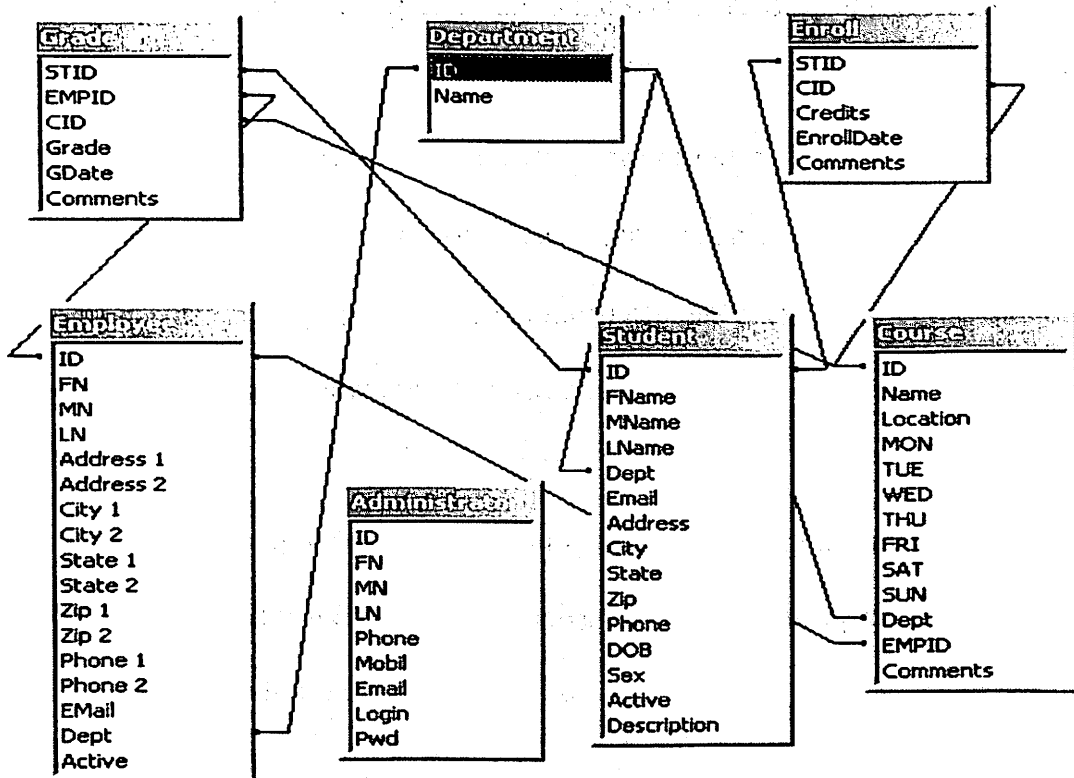


Figure 4.3 Relationships of Database (III) Tables

All the data in table 4.1~4.19 are modeled in the relational data model. However, it is semantically heterogeneous. This is due to the databases (office) autonomy and because of the following properties:

- The databases use separate codes to denote distinct grades. For example, the Database (I) takes the values A, B, C, D and E, but Database (II) takes the values 1,2,3,4 and 5;
- Types or domains of semantically equivalent attributes may be different;
- The table has similar structure is entitled with different name in different databases (Database (I).Staff, Database (II).Advisor and Database (III).Employee, etc);
- Some employees may appear in Database (I).Staff, Database (II).Advisor and Database (III).Employee, but not all employees are stored in all of the databases;
- Despite the same names, primary key values modeling the same object in different databases are independent;
- The databases may disagree on the values of vary attributes;
- In contrast, the databases always agree on a employee name and the corresponding number;
- The databases disagree upon the choice of attributes that should model the school of employees and the names modeling the same concepts;

Similar properties will be typical of the general multi-database environment. Multi-databases relative to this school will usually resemble each other, but will also present numerous semantic differences like those mentioned above.

4.2 Performance Benchmark Application

The benchmark application (Figure 4.4) was developed using Microsoft Visual C++ 6.0 based on the class definitions described in Chapter 3. The benchmark application will build a multi-thread query interface to the test databases via ODBC, test SQL queries, display system information, query results and evaluate performance. Each database engine's query component is parallel in order to run various SQL queries and retrieve results.

The performance benchmark application assumed that the tester has a basic working knowledge of running applications in a networked multi-user environment with Win32 client(s) and drive sharing. Test was done with Windows 2000 Server as the Win32 operating system environment. It is further assumed that the tester understands the basic concepts of relational database, multi-thread and functions of caching, logging, etc. and only compares results run with identical settings under identical circumstances. Measurements are presented on the GUI interface, which will also be presented in diagrams in this chapter. Benchmark application was run in the following multi-database environments:

- Simulates the deployments to sites where only a single-user and a single computer exist;
- Simulates the most common application environment for non-client/server deployments;
- Simulates the application environment where client/server are desired for best multi-user security, stability, and performance;

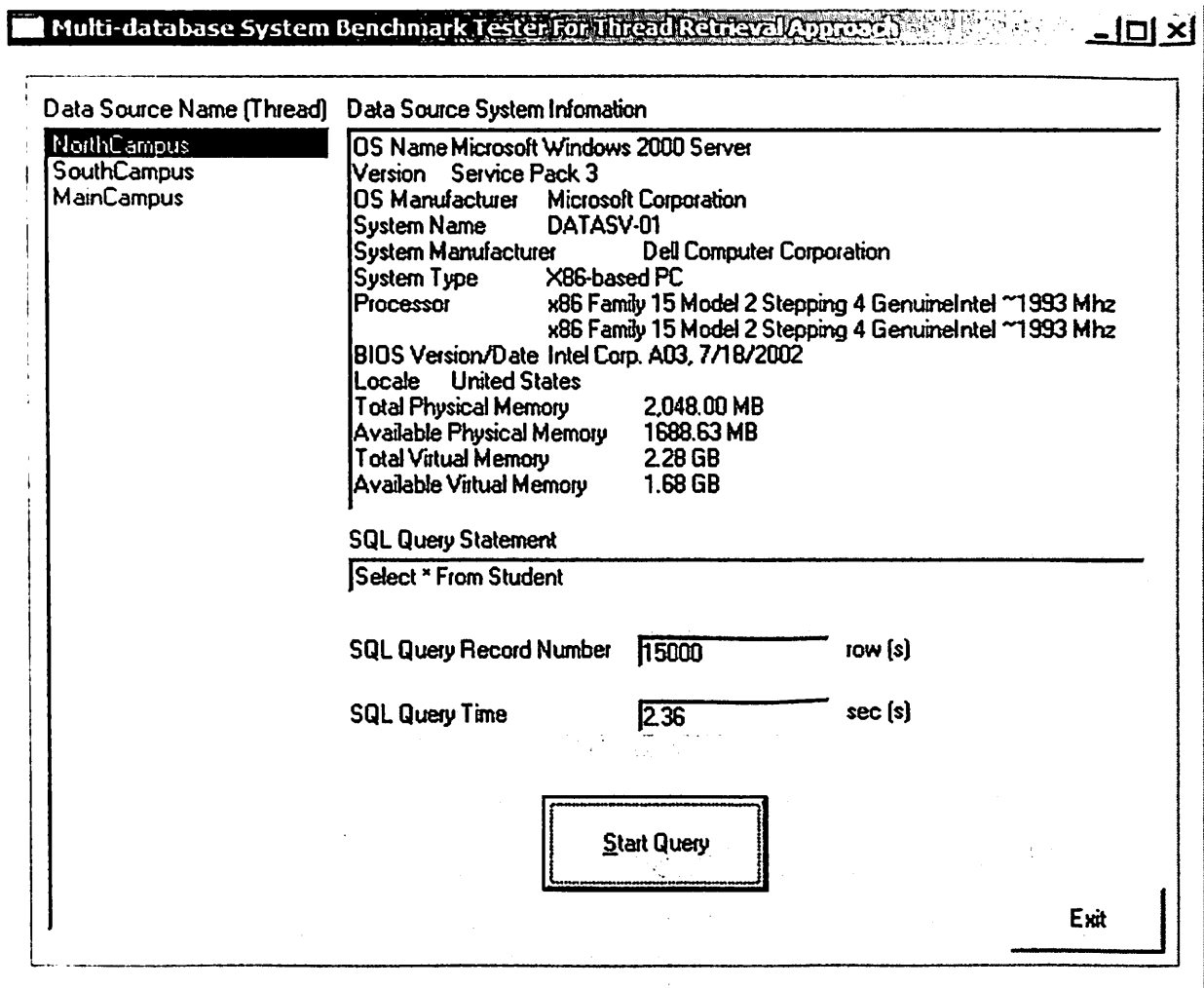


Figure 4.4 Screen Shot of Benchmark Application

4.3 Performance Measurements and Summary

4.3.1 Performance Measurements

The benchmark test has been focused on the execution time for 15 individual queries. The queries retrieve data using various SELECT, UPDATE, INSERT, DELETE, and ALTER TABLE statements in multi-database views with columns representative of typical data, such as address and student grade. In simple terms, each test performs the same operations against identical data but varies the type of databases and the size of the database. The tests were run using two Pentium III 500MHz servers with 512 MB of RAM on each, and a Pentium4 2.0 GHz server with 2 GB of

RAM across 100Mb Ethernet network. SELECT result sets were scanned for the row count, while the rows affected by a non-SELECT query was obtained through the application interface. All test results are listed as following:

Query 1 – SELECT * FROM STUDENT

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	321	4698				
SouthCampus			858	7574		
MainCampus					342	13544

Table 4.20 Query Performance of Query 1

Query 2 – SELECT Grade FROM GRADE WHERE CID < 5000

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	345	121				
SouthCampus			247	574		
MainCampus					148	1544

Table 4.21 Query Performance of Query 2

Query 3 –SELECT FNAME, MNAME, LNAME, COURSE.NAME, CREDITS FROM STUDENT, ENROLL, COURSE WHERE ENROLL.STID=STUDENT.ID AND ENROLL.CID=COURSE.ID ENROLLDATE BETWEEN #01/01/1995# AND #01/01/2002# ORDER BY FNAME

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	385	1248				
SouthCampus			268	471		
MainCampus					145	8348

Table 4.22 Query Performance of Query 3

Query 4 – SELECT * FROM STUDENT WHERE FNAME LIKE ' PAULA%'

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	227	535				
SouthCampus			228	276		
MainCampus					136	451

Table 4.23 Query Performance of Query 4

Query 5 – SELECT * FROM STUDENT WHERE LNAME IN ('JOHANNESSEN', 'MORGAN', 'DAVIS', 'SMITH')

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	313	675				
SouthCampus			255	338		
MainCampus					281	959

Table 4.24 Query Performance of Query 5

Query 6 – SELECT AVG (GRADE), LNAME, MNAME, FNAME FROM STUDENT, GRADE WHERE STUDENT.ID = STID AND STID BETWEEN 10000000 AND 50000000

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	280	3471				
SouthCampus			761	5466		
MainCampus					159	11279

Table 4.25 Query Performance of Query 6

Query 7 - SELECT STID, SUM(CREDITS) FROM ENROLL WHERE (ENDATE BETWEEN #09/01/1995# AND #09/01/2002#) GROUP BY CID

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	159	483				
SouthCampus			4681	1248		
MainCampus					277	12885

Table 4.26 Query Performance of Query 7

Query 8 - SELECT DEPARTMENT.NAME, COURSE.ID, COURSE.NAME, COURSE.LOCATION FROM COURSE, DEPARTMENT WHERE COURSE.DEPT = DEPARTMENT.ID ORDER BY DEPARTMENT.NAME

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	137	53				
SouthCampus			118	37		
MainCampus					32	88

Table 4.27 Query Performance of Query 8

Query 9 -SELECT STAFF.FNAME, STAFF.MNAME, STAFF.LNAME, ADVISOR.FNAME, ADVISOR.MNAME, ADVISOR.LNAME, EMPLOYEE.FN, EMPLOYEE.MN, EMPLOYEE.LN, DEPARTMENT.NAME FROM STAFF, ADVISOR, EMPLOYEE SELECT LEFT OUTER JOIN DEPARTMENT ON (STAFF.DEPT = DEPARTMENT.ID AND ADVISOR.DEPT = DEPARTMENT.ID AND EMPLOYEE.DEPT = DEPARTMENT.ID) WHERE (DEPARTMENT.ID BETWEEN 5 AND 15)

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	244	671				
SouthCampus			273	455		
MainCampus					145	793

Table 4.28 Query Performance of Query 9

Query 10 - UPDATE STUDENT SET DEPT = 23 WHERE ID BETWEEN 100070000 AND 350000000

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	85	113				
SouthCampus			108	251		
MainCampus					55	339

Table 4.29 Query Performance of Query 10

Query 11 – DELETE FROM GRADE WHERE CID IN (37, 45, 68, 12)

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	205	855				
SouthCampus			337	1771		
MainCampus					128	10634

Table 4.30 Query Performance of Query 11

Query 12 – SELECT (SELECT * FROM STUDENT) INTO SAMPLETEST

Database Name	Server I		Server II		Server III	
	Time (ms)	Rows	Time (ms)	Rows	Time (ms)	Rows
NorthCampus	485	5708				
SouthCampus			900	8250		
MainCampus					450	13578

Table 4.31 Query Performance of Query 12

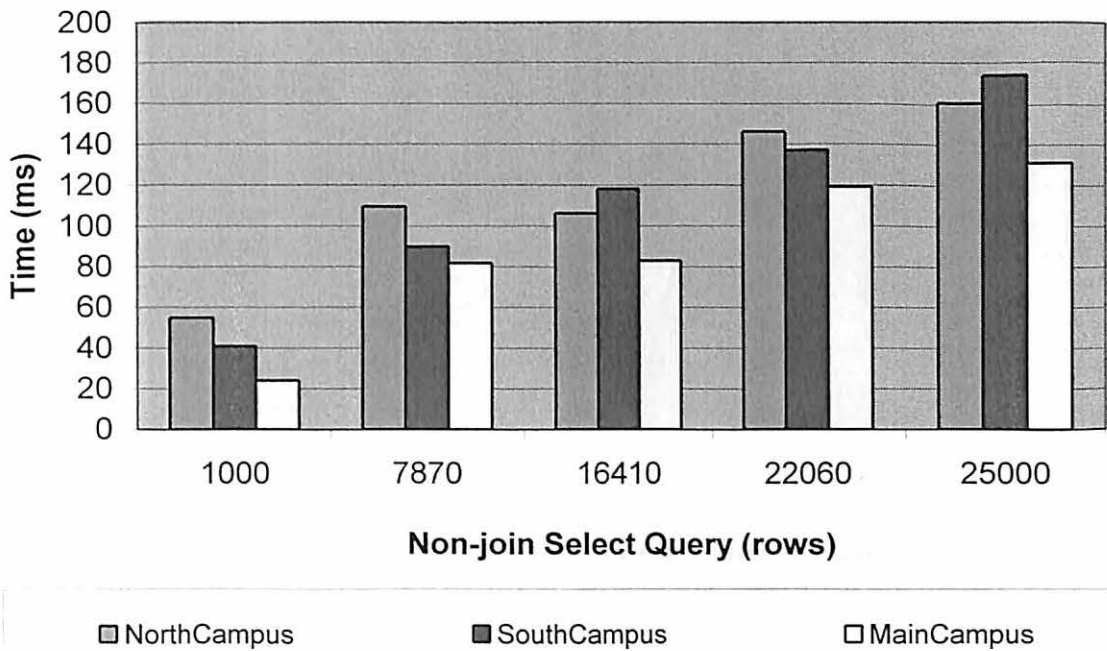


Figure 4.5 Non-join Query Analyses

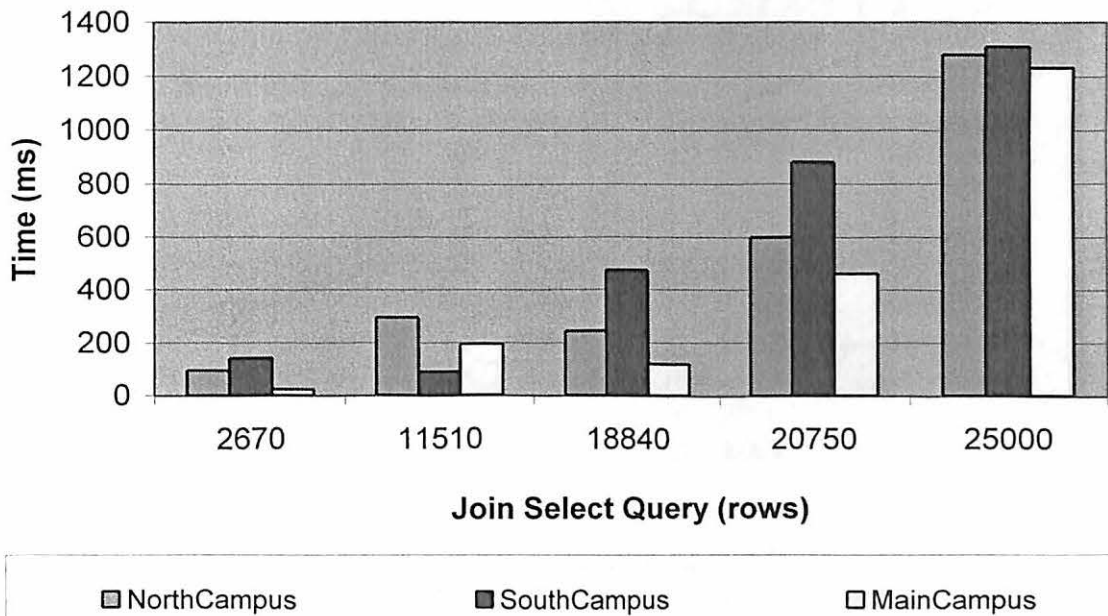


Figure 4.6 Join Query Analyses

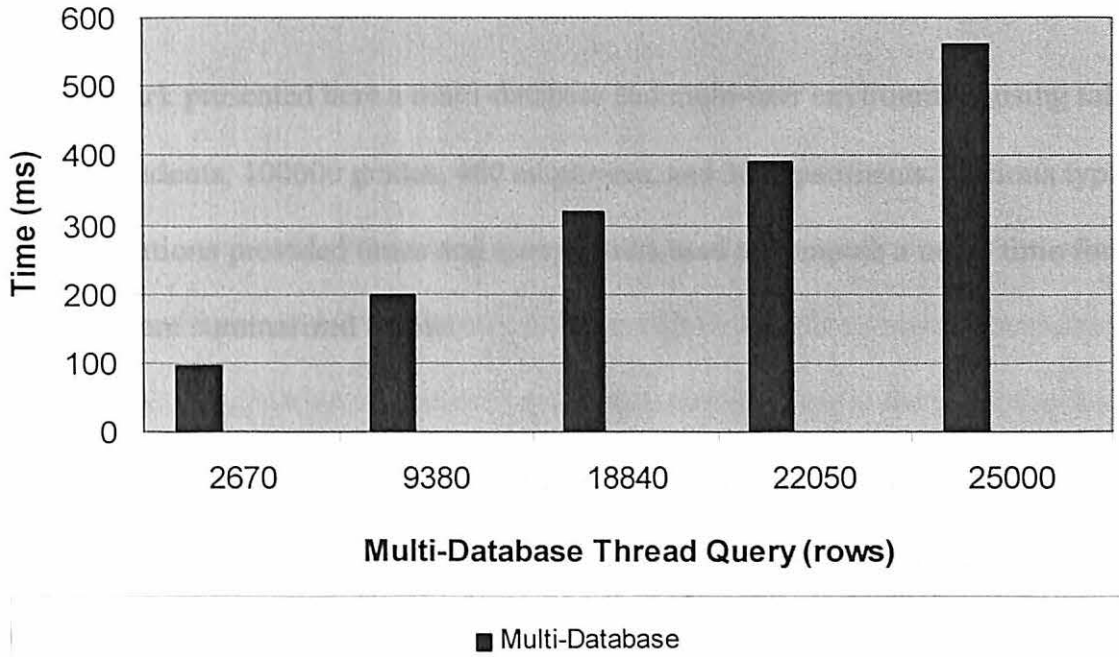


Figure 4.7 Multi-Database Thread Query Analysis

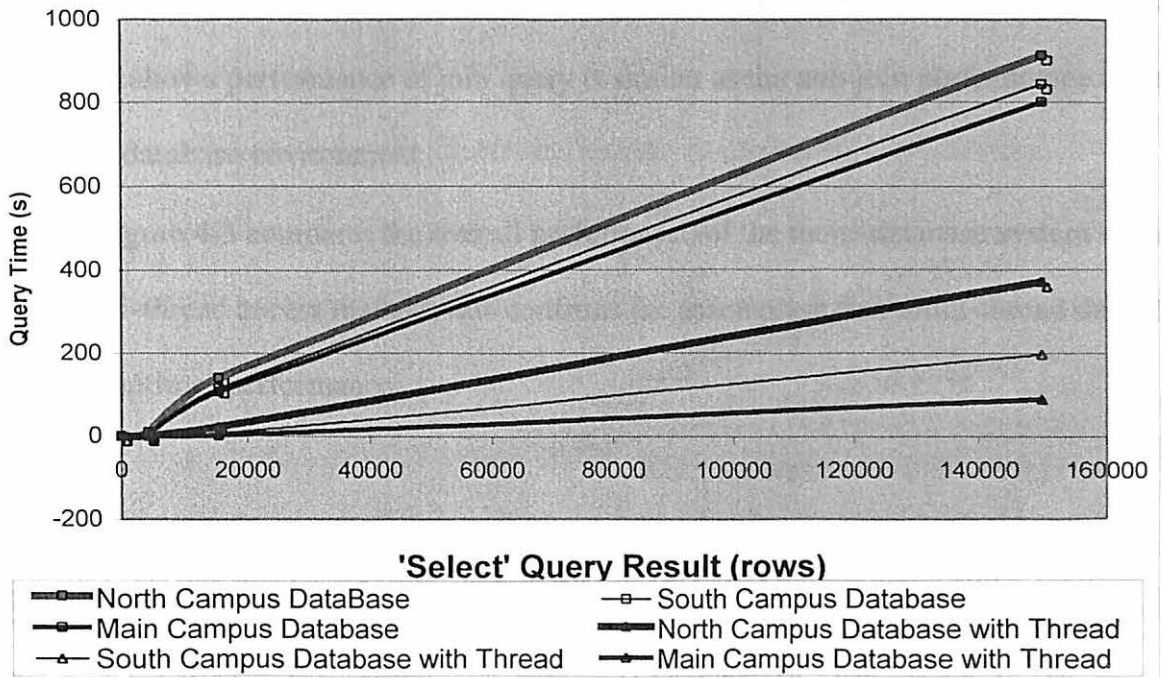


Figure 4. 8 Multi-Database Thread Overall Query Compare

4.3.2 Performance Test Summary

The benchmark presented here a multi-database and multi-user environment using tables populated 50,000 Students, 100000 grades, 400 employees, and 30 departments. Various typical SQL statement executions provided times and query results used to compute a mean time for each test. The test results are summarized below:

- Figure 4.1 and Figure 4.2 show the execution time to access SQL 2000 and MS Access individually, Figure 4.3 shows the time to execute multi-database queries using multi-thread access approach.
- Figure 4.1 demonstrates measurements permit to investigate how the data access cost varies for different size of non-join query to each single relational database. The result shows there are not considerable differences between accessing MS SQL 2000 versus MS Access.
- Figure 4.2 shows performance of join query is similar as the non-join performance under the single database environment.
- Finally, Figure 4.3 compares the overall performance of the multi-database system obtained from multi-thread access method; this confirms the assumption that multi-thread database access is optimal performance.

CHAPTER V

CONCLUSIONS

This paper presented strategies to access multiple, heterogeneous and distributed relational databases using multi-thread and ODBC capabilities. This strategy is an abstract and extended model to the MDBS architecture that provides the necessary support for the reliability in the information transaction of an object-oriented application system.

5.1 The Key Features Of The Multi-Thread Database Access Strategy

- It can reduce the frequency of global transaction using the ODBC standard and OS thread concepts onto any relational DBMSs for which a driver is available can be accessed.
- The solution of the relational DBMSs interoperability problem involves constructing multi-database views in homogeneous interfaces from the desired set of data sources.
- Multi-thread solution provided the ability to quickly and easily modify applications in this manner allows them to be rapidly tuned and redeployed for a distributed multi-database system.
- The flexibility and reuse of multi-thread ODBC settings hides all internal synchronization mechanisms from the multi-database application development so that it is able to deal with extensibility using the standard SQL interface to access the any kind of ODBC-compatible data sources.

5.2 Future Work

Based on the performance analysis, this approach demonstrated the feasibility of how the benefits from all the traditional features supported by commercial database systems, namely, features such as recovery, data integrity constraints, concurrency, optimization algorithms and implementations, are all available to the information retrieval application without additional software development. Because modern information retrieval systems need to sustain a high degree of accuracy and scale in terms of the volume of object-oriented data, therefore, the future work can be combined with an object oriented view mechanism to build an abstraction of the data retrieved with multi-thread interface. This abstraction is capable of tackling schematic discrepancies among the heterogeneous databases in an object-oriented system. The improvements that also need to be considered for multi-threading ODBC include:

- ODBC-thread specific object-oriented data type
- ODBC-thread cancellation and error handling
- ODBC-thread pool

APPENDIX A

CLASS SCHEMA DEFINITION OF MULTI-THREAD RETRIEVAL APPROACH TO A RELATIONAL MULTI-DATABASE SYSTEM

```
struct CMDBFieldInfo
```

```
{  
    char m_sName[50];  
    short m_nType;  
    long m_lSize;  
    long m_lDefinedSize;  
    long m_lAttributes;  
    short m_nOrdinalPosition;  
    BOOL m_bRequired;  
    BOOL m_bAllowZeroLength;  
    long m_lCollatingOrder;  
};
```

```
Class CMDBDatabase: Public CDatabase
```

```
{  
public:  
  
    CMDBDatabase()  
    {  
        m_pConnection = NULL;  
        m_sConnection = _T("");  
        m_sLastError = _T("");  
        m_dwLastError = 0;  
        m_nRecordsAffected = 0;  
        m_nConnectionTimeout = 0;  
    }  
  
    virtual ~CMDBDatabase()  
    {  
        Close();  
  
        m_pConnection = NULL;  
        m_sConnection = _T("");  
        m_sLastError = _T("");  
        m_dwLastError = 0;  
    }  
};
```

```

    BOOL open(LPCTSTR lpstrConnection = _T(""), LPCTSTR lpstrUserID = _T(""), LPCTSTR
lpstrPassword = _T(""));

    _ConnectionPtr GetActiveConnection()
        {return m_pConnection;};

    BOOL Execute(LPCTSTR lpstrExec);
    DWORD get_RecordCount(_RecordsetPtr m_pRs);

    BOOL IsOpen();

    void Close();

    void set_ConnectionString(LPCTSTR lpstrConnection)
        {m_sConnection = lpstrConnection;};

    CString get_ConnectionString()    {return m_sConnection;};

    CString get_LastErrorString()    {return m_sLastError;};

    DWORD get_LastError()    {return m_dwLastError;};

    CString get_ErrorDescription()    {return m_sErrorDescription;};

    void set_ConnectionTimeout(long nConnectionTimeout = 30)
        {m_nConnectionTimeout = nConnectionTimeout;};

protected:
    void dump_com_error(_com_error &e);

public:
    _ConnectionPtr m_pConnection;

protected:

    CString m_sConnection;
    CString m_sLastError;
    CString m_sErrorDescription;
    DWORD m_dwLastError;
    int m_nRecordsAffected;
    long m_nConnectionTimeout;
};

```

Figure 3.2 CMDBDatabase Class Schema Definition

Class CMDBRecordset: Public CRecordset

```
{  
public:
```

```
Enum CMDBSearchEnum
```

```
{  
    searchForward = 1,  
    searchBackward = -1  
};
```

```
BOOL set_FieldValue(int nIndex, int nValue);  
BOOL set_FieldValue(LPCTSTR lpFieldName, int nValue);  
BOOL set_FieldValue(int nIndex, long lValue);  
BOOL set_FieldValue(LPCTSTR lpFieldName, long lValue);  
BOOL set_FieldValue(int nIndex, unsigned long lValue);  
BOOL set_FieldValue(LPCTSTR lpFieldName, unsigned long lValue);  
BOOL set_FieldValue(int nIndex, double dblValue);  
BOOL set_FieldValue(LPCTSTR lpFieldName, double dblValue);  
BOOL set_FieldValue(int nIndex, CString strValue);  
BOOL set_FieldValue(LPCTSTR lpFieldName, CString strValue);  
BOOL set_FieldValue(int nIndex, COleDateTime time);  
BOOL set_FieldValue(LPCTSTR lpFieldName, COleDateTime time);  
BOOL set_FieldValue(int nIndex, bool bValue);  
BOOL set_FieldValue(LPCTSTR lpFieldName, bool bValue);  
BOOL set_FieldValue(int nIndex, COleCurrency cyValue);  
BOOL set_FieldValue(LPCTSTR lpFieldName, COleCurrency cyValue);  
BOOL set_FieldValue(int nIndex, _variant_t vtValue);  
BOOL set_FieldValue(LPCTSTR lpFieldName, _variant_t vtValue);
```

```
BOOL set_FieldEmpty(int nIndex);  
BOOL set_FieldEmpty(LPCTSTR lpFieldName);
```

```
void cancel_Update();
```

```
BOOL Update();  
void Edit();  
BOOL AddNew();
```

```
BOOL Find(LPCTSTR lpFind, int nSearchDirection = CMDBRecordset::searchForward);  
BOOL FindFirst(LPCTSTR lpFind);  
BOOL FindNext();
```

```
CMDBRecordset();
```

```
CMDBRecordset(CMDBDatabase* pAdoDatabase);
```

```
virtual ~CMDBRecordset()
```

```
{
```

```
    Close();
```

```
    if(m_pRecordset) m_pRecordset.Release();
```

```
    if(m_pCmd) m_pCmd.Release();
```

```
    m_pRecordset = NULL;
```

```
    m_pCmd = NULL;
```

```
    m_pRecBinding = NULL;
```

```
    m_sQuery = _T("");
```

```
    m_sLastError = _T("");
```

```
    m_dwLastError = 0;
```

```
    m_nEditStatus = dbEditNone;
```

```
}
```

```
CString get_Query()
```

```
{return m_sQuery;};
```

```
void set_Query(LPCSTR strQuery)
```

```
{m_sQuery = strQuery;};
```

```
DWORD get_RecordCount();
```

```
BOOL IsOpen();
```

```
void Close();
```

```
    BOOL Open(_ConnectionPtr mpdb, LPCTSTR lpstrExec = _T(""), int nOption =  
CMDBRecordset::openUnknown);
```

```
    BOOL Open(LPCTSTR lpstrExec = _T(""), int nOption = CMDBRecordset::openUnknown);
```

```
    BOOL OpenSchema(int nSchema, LPCTSTR SchemaID = _T(""));
```

```
long get_FieldCount()
```

```
{return m_pRecordset->Fields->GetCount();};
```

```
    BOOL get_FieldValue(LPCTSTR lpFieldName, int& nValue);
```

```
    BOOL get_FieldValue(int nIndex, int& nValue);
```

```
    BOOL get_FieldValue(LPCTSTR lpFieldName, long& lValue);
```

```
    BOOL get_FieldValue(int nIndex, long& lValue);
```

```
    BOOL get_FieldValue(LPCTSTR lpFieldName, unsigned long& ulValue);
```

```
    BOOL get_FieldValue(int nIndex, unsigned long& ulValue);
```

```

BOOL get_FieldValue(LPCTSTR lpFieldName, double& dbValue);
BOOL get_FieldValue(int nIndex, double& dbValue);
BOOL get_FieldValue(LPCTSTR lpFieldName, CString& strValue, CString strDateFormat =
_T(""));
BOOL get_FieldValue(int nIndex, CString& strValue, CString strDateFormat = _T(""));
BOOL get_FieldValue(LPCTSTR lpFieldName, COleDateTime& time);
BOOL get_FieldValue(int nIndex, COleDateTime& time);
BOOL get_FieldValue(int nIndex, bool& bValue);
BOOL get_FieldValue(LPCTSTR lpFieldName, bool& bValue);
BOOL get_FieldValue(int nIndex, COleCurrency& cyValue);
BOOL get_FieldValue(LPCTSTR lpFieldName, COleCurrency& cyValue);
BOOL get_FieldValue(int nIndex, _variant_t& vtValue);
BOOL get_FieldValue(LPCTSTR lpFieldName, _variant_t& vtValue);

BOOL IsFieldNull(LPCTSTR lpFieldName);
BOOL IsFieldNull(int nIndex);
BOOL IsFieldEmpty(LPCTSTR lpFieldName);
BOOL IsFieldEmpty(int nIndex);

bool IsEOF()
    {return m_pRecordset->EndOfFile == VARIANT_TRUE;};

bool IsEOF()
    {return m_pRecordset->EndOfFile == VARIANT_TRUE;};

bool IsBOF()
    {return m_pRecordset->BOF == VARIANT_TRUE;};

bool IsBOF()
    {return m_pRecordset->BOF == VARIANT_TRUE;};

void move_First()
    {m_pRecordset->MoveFirst();};

void move_Next()
    {m_pRecordset->MoveNext();};

void move_Previous()
    {m_pRecordset->MovePrevious();};

void move_Last()
    {m_pRecordset->MoveLast();};

long get_AbsolutePage()
    {return m_pRecordset->GetAbsolutePage();};

```

```

void set_AbsolutePage(int nPage)
    {m_pRecordset->PutAbsolutePage((enum PositionEnum)nPage);};

long get_PageCount()
    {return m_pRecordset->GetPageCount();};

long get_PageSize()
    {return m_pRecordset->GetPageSize();};

void set_PageSize(int nSize)
    {m_pRecordset->PutPageSize(nSize);};

long get_AbsolutePosition()
    {return m_pRecordset->GetAbsolutePosition();};

void set_AbsolutePosition(int nPosition)
    {m_pRecordset->PutAbsolutePosition((enum PositionEnum)nPosition);};

BOOL get_FieldInfo(LPCTSTR lpFieldName, CMDBFieldInfo* fldInfo);
BOOL get_FieldInfo(int nIndex, CMDBFieldInfo* fldInfo);

CString get_LastErrorString()
    {return m_sLastError;};

DWORD get_LastError()
    {return m_dwLastError;};

void GetBookmark()
    {m_varBookmark = m_pRecordset->Bookmark;};

BOOL set_Bookmark();
BOOL Delete();

bool IsConnectionOpen()
    {return m_pConnection != NULL && m_pConnection->GetState() != adStateClosed;};

BOOL set_Filter(LPCTSTR strFilter);
BOOL set_Sort(LPCTSTR lpstrCriteria);

BOOL Execute(CMDBThreadMgr* pCommand);

BOOL Requery();

```


public:

protected:

```
int m_nSearchDirection;  
CString m_sFind;
```

```
int m_nEditStatus;  
CString m_sLastError;  
DWORD m_dwLastError;  
void dump_com_error(_com_error &e);  
CString m_sQuery;
```

protected:

```
BOOL put_FieldValue(LPCTSTR lpFieldName, _variant_t vtFld);  
BOOL put_FieldValue(_variant_t vtIndex, _variant_t vtFld);  
BOOL get_FieldInfo(FieldPtr pField, CMDBFieldInfo* fldInfo);
```

```
};
```

Figure 3.3 CMDBRecordset Class Schema Definition

```

template<typename T>
Class CMDBSQLThreadTmp: Public CWinThread
{
public:

    CMDBSQLThreadTmp(T& thObject, void (T::*pfnOnRunning)(), int nPriority =
THREAD_PRIORITY_NORMAL);

    bool wait_until_terminate(DWORD dwMiliSec = INFINITE);

    bool start();

    bool start_and_wait();

    bool suspend();

    bool resume();

    bool pause();

    bool is_running();

    bool is_terminated();

    bool is_suspend();

    void set_priority(int nLevel);

    int get_Priority();

    void speed_up();

    void slow_down();

    void terminate();

    virtual ~CMDBSQLThreadTmp()
    {
        ::CloseHandle(m_hEvent);
    }

protected:
    static unsigned __stdcall _ThreadProc(LPVOID lpParameter);

    void exit();

```

```

inline void on_running();

private:

hide copy constructor and assignment
CMDBSQLThreadTmp ( const CMDBSQLThreadTmp& t );
CMDBSQLThreadTmp& operator= ( const CMDBSQLThreadTmp& t );

CString GetLastError();

bool copy_handle ( HANDLE h )
{
    BOOL b = ::DuplicateHandle (
        ::GetCurrentProcess(), h,
        ::GetCurrentProcess(), &m_hThread,
        0, FALSE, DUPLICATE_SAME_ACCESS
    );
    return (b != FALSE);
}

protected:

T& m_thObject;
void (T::*m_pfnOnRunning)();
HANDLE m_hThread, m_hEvent;
int m_nInitPriority;
unsigned int m_dwThreadID;
bool m_bTerminate, m_bSuspend, m_bIsRunning;
};

```

Figure 3.4 CMDBSQLThreadTmp Class Schema Definition

```

Class CMDBException: Public CException
{
public:
    enum
    {
        noError, ' no error
        Unknown, ' unknown error
    };

    DECLARE_DYNAMIC(CMDBException);

    CMDBException();
    CMDBException(const CString& sMessage);
    ~CMDBException();

public:
    CString get_Description() const;
    void set_Message(const String& sMessage);
    virtual CString get_ExceptionType() const;
    virtual CString get_Message() const;
    static int get_Error(int nADOError);
    int m_nCause;
    CString m_sErrorString;

protected:
    virtual CString to_String() const;
    virtual CString get_LocalizedDescription() const;
}

```

Figure 3.5 CMDBException Class Schema Definition

Class CMDBGeneralMgr

```
{  
public:  
  
    CMDBGeneralMgr();  
  
    virtual ~CMDBGeneralMgr()  
    {  
        m_pParameter.Release();  
        m_pParameter = NULL;  
        m_sName = _T("");  
    }  
  
    BOOL create_MDBThreadMgr();  
  
    BOOL create_MDBThreadQryMgr();  
  
    BOOL destroy_MDBThreadMgr();  
  
    BOOL destroy_MDBThreadQryMgr();  
  
    void analysis_SQL(CString& sSQLStatements);  
  
    void create_LocalSQL();  
  
    CMDBRecordset* query_LocalSQL();  
  
protected:  
    void dump_com_error(_com_error &e);  
  
protected:  
  
    CMDBThreadMgr *pMDBThreadMgr;  
    CMDBThreadQryMgr *pMDBThreadQryMgr;  
    CArray *pMDBThreadRegTable;  
  
    CString m_sName;  
  
    CArray m_aryLocalSQL;  
  
    CString m_sLastError;
```

```
DWORD m_dwLastError;
```

```
private:
```

```
    //gives back the sql table name of criterion with a counter e.g. forder  
    CString fordertTableSQL();
```

```
    //gives back the sql table name of student quality with a counter e.  
    CString hatTableSQL();
```

```
    //gives back an sql string representing the matching criterion criterion  
    CString matchCriterionSQL();
```

```
    //return specific matching criterion implemented by subclasses  
    CString specMatchSQL();
```

```
};
```

Figure 3.6 CMDBGeneralMgr Class Schema Definition

Class CMDBThreadMgr

```
{  
public:  
  
    CMDBThreadMgr();  
  
    virtual ~CMDBThreadMgr()  
    {  
        m_pCommand.Release();  
        m_pCommand = NULL;  
        m_sCommandText = _T("");  
    }  
  
    void set_TimeOut(long nTimeOut)  
        {m_pCommand->PutCommandTimeout(nTimeOut);};  
  
    void allocate_MDBThread(LPCTSTR lpstrConnection = _T(""), LPCTSTR lpstrUserID =  
_T(""), LPCTSTR lpstrPassword = _T(""), int nPriority = THREAD_PRIORITY_NORMAL)  
    {  
  
        CMDBDatabase* pDB;  
        pDB = new CMDBDatabase();  
  
        CMDBSQLThreadTmp<CMDBDatabase>* pMDBTrd = new  
CMDBSQLThreadTmp<CMDBDatabase>(*pDB,&CMDBDatabase::open(lpstrConnection,lpstrU  
serID, lpstrPassword), nPriority);  
        pMDBTrd ->start_and_wait();  
        pMDBTrd ->wait_until_terminate();  
  
        aryMDBThreadRegTable.Add(pMDBTrd);  
  
    }  
  
    CMDBDatabase* get_MDB_by_Thread(LPCTSTR lpstrConnection);  
  
    BOOL deallocate_MDBThread(LPCTSTR lpstrConnection);  
  
    CArray* get_MDBThreadRegTable();  
  
    // Retrieves the thread ID of the calling thread  
    DWORD get_CurrentThreadId();  
  
    int GetRecordsAffected()  
        {return m_nRecordsAffected;};  
};
```

```
_CommandPtr GetCommand()  
    {return m_pCommand;};
```

protected:

```
void dump_com_error(_com_error &e);
```

protected:

```
CArray aryMDBThreadRegTable;
```

```
int m_nRecordsAffected;
```

```
CString m_sLastError;
```

```
DWORD m_dwLastError;
```

```
};
```

Figure 3.7 CMDBThreadMgr Class Schema Definition

Class CMDBThreadQryMgr

```
{
public:

    CMDBThreadQryMgr();

    virtual ~CMDBThreadQryMgr()
    {
        m_pCommand.Release();
        m_pCommand = NULL;
        m_sCommandText = _T("");
    }

    CMDBRecordset* process_LocalSQL();

    CMDBRecordset* inter_join_LocalSQL();

    int get_MDBThreadAffected()
        {return m_nRecordsAffected;};

    void resort_MDBThread();

        bool init ();

        void update_evalQuery ();

        CString* get_Fields();

protected:
    void dump_com_error(_com_error &e);

protected:

    CArray m_aryLocalSQL;

    int m_nRecordsAffected;

    CString m_sLastError;

    DWORD m_dwLastError;
};
```

Figure 3.8 CMDBThreadQryMgr Class Schema Definition

REFERENCES

- [1] Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387. Association for Computing Machinery, Inc. 1970
- [2] Codd, E. F. (1979). Extending the Database Relational Model to Capture More Meaning. ACM TODS, Vol.4:4, pp. 177-227.
- [3] Codd, E. F. (1990). The Relational Model for Database Management (2nd Edition). New York, NY: Addison-Wesley Publishing Company, 1990.
- [4] Date, C. J., & Darwen, Hugh (1993). The SQL Standard, Third Edition. New York, NY: Addison-Wesley Publishing Company, 1993.
- [5] Elmasri R., Navathe S. (1989). Fundamentals of Database Systems. New York, NY: Addison-Wesley Publishing Company, Reading, 1989.
- [6] Elmasri, R., Weeldreyer, J., & Hevner, A. (1985). The Category Concept: An Extension to the Entity-Relationship Model. Int'l Journal of Data and Knowledge Engineering, Vol.1:1.
- [7] Elmasri, R. and S.B. Navathe (2004). Fundamentals of Database Systems, 4th ed. Addison Wesley. Extension of the 3rd edition with SQL3 and XML
- [8] Embley, D. W. (1998). Object Database Development - Concepts and Principles. New York, NY: Addison-Wesley Publishing Company, Reading, 1998
- [9] Hurson, A. R., M. W. Bright, and S. H. Pakzad, ed. (1994). Multidatabase Systems: An Advanced Solution for Global Information Sharing. IEEE 1994.
- [10] Khoshafian, S. (1993). Object-oriented Databases. NY: John Wiley & Sons, 1993.
- [11] Litwin et al, W. (1990). MSQL: A Multidatabase language. Information Sciences, 49(1-3): 59--101, October-December 1990. 16.
- [12] Sheth, A. & Klas, W. (Eds). (1998). Multimedia Data Management : Using Metadata to Integrate and Apply Digital Media. McGraw-Hill Series on Data Warehousing and Data Management, 1998.

- [13] Silberschatz. Abraham, Peter Baer Galvin and Greg Gagne (2002). Operating System Concepts. John Wiley & Sons, Inc., 2002

- [14] Simon, Richard J. (1996). Windows 95 multimedia & ODBC API Bible. Corte Madera, CA: Waite Group Press, 1996.

- [15] Ullman, J. (1989). Principles of Database and Knowledge-base Systems (Vol. I & II). Rockville, MD: Computer Science Press, 1989.

- [16] Zaratian, Beck (1998) Microsoft Visual C++ 6.0 Programmer's Guide. Redmond, WA: Microsoft Press, 1998

GLOSSARY

ANSI American National Standards Institute

API Application Programming Interfaces

ARM Annotated Reference Manual

BSI British Standards Institute

class Template from which objects can be created. It is used to specify the behavior and attributes common to all objects of the class.

DAO Data Access Objects

DBMS Database management systems.

encapsulation The facility by which access to data is restricted to legal access. Illegal access is prohibited in an object by encapsulating the data and providing the member functions as the only means of obtaining access to the stored data.

encompass The facility by which access to data is restricted to legal access. Illegal access is prohibited in an object by encapsulating the data and providing the member functions as the only means of obtaining access to the stored data.

ERP Enterprise Resource Planning Project;

GDI Global Defense Information

GUI Graphic User Interface

MDBS Multi-database management systems.

heterogeneous Consisting of dissimilar data structure or parts;

inheritance The mechanism by which new classes are defined from existing classes. Subclasses inherit operations of their parent class. Inheritance is the mechanism by which reusability is facilitated. It is a mechanism for sharing behavior and attributes between

integrity A kind of consistency that guaranteed the existence of all objects referenced. The

consistency of the database can be typically expressed through predicates or conditions on the current state of the database.

inter join A statement is used to combine the data contained in two relational database tables based upon a common attribute.

ISO International Standards Organization

object A combination of data and the collection of operations that are implemented on data; also, a collection of operations that shares a state. An object is used to model a person, place, thing, or event from the real world. It encapsulates data and operations that can be used to manipulate the data and ponds to requests for service.

ODBC Open Database Connectivity

OODBMS Objected-oriented database management system that can be used to store and retrieve objects.

primary key The primary key of a relational table uniquely identifies each record in the table. It can either be a normal attribute that is guaranteed to be unique (such as Social Security Number in a table with no more than one record per person) or it can be generated by the DBMS (such as a globally unique identifier, or GUID, in Microsoft SQL Server).

query An activity that involves selecting objects from implicitly or explicitly identified collections based on a specified predicate.

recordset A set of database record consists of one set of tuples for a given relational table. In a relational database, records correspond to rows in each table.

RTTI Run-Time Type Identification

thread A thread is basically a path of execution through a program. It is also the smallest unit of execution that Win32 schedules. A thread consists of a stack, the state of the CPU registers, and an entry in the execution list of the system scheduler. Each thread shares all of the process's resources.

table The grouping of information in a relational database. Tables are composed of columns and rows.

transaction A sequence of database operations that transforms a consistent state of a database into another consistent state, without necessarily preserving consistency at all intermediate points.

type A predicate defined over value that can be used in a signature to restrict a possible parameter or characterize a possible result.

VITA



Hui Lin

Candidate for the Degree of

Master of Science

Thesis: A MULTI-THREAD RETRIEVAL APPROACH TO A RELATIONAL
MULTI-DATABASE SYSTEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Beijing, P. R. China, April 12, 1962, the only daughter of Mr. Yincheng Lin and Mrs. Qian Zhang

Education: Graduated from the No. 28 High School of Beijing, Beijing, P. R. China, in July 1980; received the Bachelor of Science from Beijing Technology University, Beijing, China, in July 1987; Completed the requirements for the Master of Science at Oklahoma State University in May, 2004.

Professional Experience: Employed by YuYang Trading Corp., Beijing, China, as a inspector, September 1987 to July 1989; Employed by Ocean China Inc., Alberta, Canada, as a market director, January 1991 to January 1995. Employed by Jinxin Telecom Corp. Beijing, China, as a Senior Software Design Engineer from August 1995.