DESIGN AND IMPLEMENTATION OF AN OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEM FOR AQUIRE DATABASE

By

HUA LIOU

Bachelor of Science

Zhongshan University

Guangzhou, P. R. China

1990

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
December, 1995

DESIGN AND IMPLEMENTATION OF AN OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEM FOR AQUIRE DATABASE

Thesis Approved:

H. Lu

Thesis Adviser

Blayne E. Mayfield

Michael J. Folk

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. Huizhu Lu for her intelligent supervision, constructive guidance, encouragement and patience. My sincere appreciation also extends to my other committee members Dr. Blayne E. Mayfield and Dr. Mitchell Neilsen. Their guidance, assistance, encouragement, and friendships are very helpful and invaluable throughout the research.

Special thanks go to Dr. Burk for providing AQUIRE data files, and Dr. Clement Ward, Mr. Roland Stolfa and Mr. Mark J. Vasoll for their assistant with the research and thesis preparation.

I would also like to give my special appreciation to my parents, Jinkun Liu and Meiji Fan, and my husband, Mingwu Su, for their strong encouragement, love and understanding throughout this research.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Object-Orientation Concepts

Although the concepts of object-oriented programming [Rine 1992, Wagner 1992] were developed three decades ago, it is only in the last decade that the concepts of objected-oriented programming (such as classes of objects with methods and inheritance) and languages (such as Smalltalk, C++, Eiffel and Common LISP Object System (CLOS)) have become popular. Thereafter, more and more researchers express interest in the applications of object-orientation concepts and approaches, such as the design and implementation of the relational database system utilizing the object-oriented analysis and design methodologies, graphic user interface and office information systems (OIS) design and implementation, computer aided design (CAD), etc. [Alagic 1989, Bertino 1991, Fichman 1992, Pappas 1994, Rine 1994, Weiser 1989, Wells 1992].

The object-orientation concepts have evolved in three different disciplines: first in programming languages, then in artificial intelligence, and then in database. The basic concepts of object-orientation are objects, classes, abstract data type, data encapsulation, and inheritance. An object-oriented approach to programming is based on the concepts of encapsulation and extendibility. Encapsulation means to shape some of the data from the user in the programming, while extendibility refers to the ability to extend an existing system without introducing change to it. Based on the object-oriented approach, the object-oriented analysis (OOA) is an analysis model developed to describe the functionality of the system, it represents the domain of real world problems. After the object-oriented analysis is done, one can utilize the object-oriented

design (OOD) methodology to design and implement the simulation or solution of a problem.

Object-Oriented Database Management Systems

Object-oriented Database Management Systems (OODBMS) are one of the major application areas of object-orientation concepts. Several OODBMS have been developed to date [Mariani 1993, Hurson 1993, Bertino 1989, Weinreb 1988, Velez 1989, Weiser 1989]. The advantages of them are listed as follows:

(a) data structures are flexible;

(b) there are many facilities for describing data;

(c) data values can be inherited;

(d) complex models can be built;

(e) source codes are reusable.

Graphic User Interface

Graphic User Interface (GUI) is one of the most popular techniques used in the user interface design. The GUI usually can provide users with several convenient data input and output formats, such as windows, menus, mice, image, and voice. With GUI, the system is more user-friendly, easy-to-learn and easy-to-use. This is very important for non-professional users.

Microsoft Visual C++

Microsoft Visual C++ is a complete Windows software development kit (SDK) product. MS Visual C++ Version 1.5 or later contains a lot of windows programming and user interface design tools, and many pre-defined basic classes and functions, such as, AppWizard, ClassWizard, Resource Editor, Microsoft Foundation Class Library

(MFC) and MFC dynamic link library (DLL). MFC version 2.5 supports Open Database Connectivity (ODBC) which allows the application to access and update data stored in many popular databases, like Microsoft Access, FoxPro, and SQL Server. MFC version 2.5 also supports Object Linking and Embedding (OLE) which supports for in-place editing, linking, drag-and-drop, and OLE Automation. Moreover, the application developed under MS Visual C++ development environment contains reusable and modular components, which are very useful for programmers to modify the application in the future. The applications can also provide users with standard graphic user interface and multiple data input and output methods.

AQUIRE Database

The Aquatic Toxicity Information Retrieval (AQUIRE) Database was firstly created by the United States Environmental Protection Agency Office of Pesticides and Toxic Substances in 1981 and updated annually. The AQUIRE database contains various information of chemical toxicity related to the different aquatic organisms and plants. The information includes: the citation of scientific articles published nationally and internationally in aquatic toxicity area; the chemical name and its Chemical Abstract Service (CAS) registry number of each chemical material presented in the database; and the latin and common name of each aquatic organism and plant, etc. The information is organized as one main information data file which contains up to 105,394 records with eight associative information data files. The AQUIRE database used in this search is the version created in 1989 [Kulpaiboon 1993].

The Objective of the Thesis

The researcher will apply object-oriented analysis (OOA/Coad-Yourdon) and object-oriented design (OOD/Booch) methodologies to analyze and design the database

management system for the AQUIRE database. This system is expected to be a complete, efficient, and user-friendly database management system for the Aquatic Toxicity Database.

The original AQUIRE database contains the aquatic toxicity information stored in the VAX FORTRAN format without any attached software to retrieve the information and generate the query reports. Currently, there is a commercial company that can provide a set of reports. However, it is expensive for a user who needs a lot of information for his/her research. Moreover, since PCs are used widely than Mini-computer and IBM mainframe machines by users, and a database management system running on PC will be an inexpensive and very convenient solution for users to retrieve the information as often as he/she needs.

The objectives of the research are: (1) design and implement an Object-Oriented Database Management System for the AQUIRE database; (2) provide the Graphic User Interface to users by utilizing the graphic user interface design tools of Microsoft Visual C++ ; (3) make the system user-friendly to both professional and non-professional users; (4) present a method of how to analyze, design, implement and test an object-oriented database management system for an existing very large relational database.

The Organization of the Thesis

The thesis presents the introduction and literature reviews of the research, the detail steps of system analysis, and how it leads to the system design and implementation. The thesis also describes how to utilize the OOA/Code-Yourdon and OOD/Booch methodologies in practical project development and implementation.

Chapter I, which is the current chapter, is an introduction that describes the object-orientation concepts, object-oriented database management systems, graphic user

interface, and Microsoft Visual C++ as the background of the research. The introduction of the original AQUIRE database, the objective and organization of the thesis are also presented in this chapter.

Chapter II is the literature reviews for the research. It discusses the object-oriented model in detail, such as objects, classes, inheritance, encapsulation, and polymorphism, etc. The object-oriented database management system is also described with the reviews of some related OODBMS.

Chapter III describes the system analysis which includes the analysis of the original AQUIRE database, OOA/Coad-Yourdon methodology and how to utilize the methodology in the practical research.

Chapter IV discusses how the system analysis leads to the system design by utilizing OOD/Booch methodology. It presents the design of the database retrieval strategy, indexes, database classes and objects, and user interface classes and objects. The detail design of each class and its data members and function members are described also.

Chapter V presents the implementation of the system with Microsoft Visual C++ Version 2.0 environment under Windows NT 3.5. Firstly, some of the basic components of MS Visual C++ are introduced, then the implementation of the system with the complete architecture of the classes and objects is described. Secondly, the detail information of how to utilize MS Visual C++ in implementing the design of the system is illustrated. Finally, the three kinds of system testing technologies which are used to test the system are discussed.

Chapter VI is the summary and conclusions of the thesis and the suggestion of the future modification for the system. It followed by references, Appendix A, B, C and D. Appendix A discusses the data representation of the VAX FORTRAN and the INTEL 80486 formats, and the data conversion methods between two machines for four different data types. Appendix B, C, D are the system installation guides, user's

manual and sample output respectively. Appendix E describes a procedure for printing results.

CHAPTER II

LITERATURE REVIEW

Relational Database Model

A database is a collection of persistent data used by an application program [Date 1990]. A database management system (DBMS) is a proprietary software for handing the storage and retrieval of data. Generally, a DBMS provides the following advantages: (1) data independence; (2) data integrity; (3) data concurrence and consistency; (4) recovery; (5) access control; (6) controlled data redundancy; (7) centralized control; (8) data maintenance.

The overview of the architecture of a DBMS is separated into three logical views: internal view, global view, and external views. The internal, global, and external views are described using a Data Definition Language (DDL). Users and applications access data through an external view using a DBMS specific languages called Data Manipulation Language (DML).

A relational DBMS is a type of DBMS, which is perceived to hold data in a series of two dimensional tables. Each table consists of a number of rows (called tuples) and columns (called attributes). Relationships between rows in the different tables are represented by the storage of foreign keys within one table. The relational DBMS provides several relational algebra among the tables to generate new tables. These algebra include: Restrict, Project, Product, Union, Intersection, Difference, (Natural) Join, and Divide.

Currently, there are several relational DBMS products available, such as DB2, SQL/DS, OS/2 Extended Edition Database Manager, SQL/400 DB Manager, ORACLE, Sybase, Informix, Foxpro, Paradox, dBase IV, etc.

Basic Concepts of Object-Oriented Programming

Object-oriented programming is a modeling paradigm which supplies the raw modeling power of objects with the management flexibility of classes and inheritance. In the object-oriented paradigm, objects are the atomic units of encapsulation; classes manage the collections of objects; inheritance structures the collection of objects and the inheritance structures of classes.

Objects

Complex objects are built from simpler ones by applying constructors to them. The simplest objects are integers, characters, byte strings of any length, Boolean, and floats (one might add other atomic types). Objects may be attached to or communicate to another by way of a well-defined user interface; also, objects maybe classified according to common behavior and other characteristics, such as: (1) Abstract data type; (2) Data encapsulation; (3) Inheritance. The object constructors must be orthogonal to the objects; that is, any constructor should apply to any object.

Classes

Classes specify the behavior of a collection of objects with common operation. They correspond to the notion of an abstract data type. They have two parts: the interface and the implementation. Only the interface part is visible to the users of the classes; the implementation part of the object is seen only by the class designer. The interface consists of a list of operations together with their signatures (i.e. the type of input parameters and the type of the result).

Implementation consists of a data part and an operation part. The data part which describes the structures of this data part can be more or less complex. The

operation part consists of procedures which implement the operations of the interface part.

Inheritance and Class Hierarchies

The concept of inheritance is a second reusability mechanism. It lets a class, called a subclass, be defined starting from the definition of other classes and messages. In addition, a subclass can have specific attributes, methods, and messages that are not inherited. Moreover, the subclass can override the definition of the super class's attributes and methods.

A class can have several subclasses. Some systems let a class have several super classes (multiple inherence), while others impose the restriction of a single super class. Based on inheritance, the set of classes in the schema can be organized in an inheritance graph. The inheritance graph is a tree when the model does not provide multiple inheritance.

Encapsulation

The idea of encapsulation comes from the need for a clear distinction between the specification and the implementation of an operation, and the need for modularity. Modularity is necessary to structure complex applications designed and implemented by a team of programmers. It is also necessary as a tool for protection and authorization.

The idea of encapsulation in programming language comes from the abstract data types. An object has operations that can be performed on the object. It is the only visible part of the object. The implementation part has a data part and a operation part. The data part is the representation or state of the object, and the operation part describes the implementation of each operation.

In an object-oriented database system, for example, we define Employee as an object that has a data part, probably very similar to the record that was defined in the relational system, which consists of salary raises and termination of the Employee. The operation part of the Employee object consists of some operations, such as to store the information to the data part, and to retrieve the information from the data part, etc.

Encapsulation provides a form of "logical data independence"; we can change the implementation of a class without changing any of the programs using that class. Thus, the application programs are protected from implementation changes in the lower layer of the system.

Object-Oriented Database Systems (OODBS)

Comparison of Conventional Database System and OODBS

Database systems have long been successful for business applications, like an office information system (OIS), Computer Aided Design (CAD), and transaction mechanics. Conventional database systems provide database languages to allow application programmers or end-users to define and manipulate the database. A conventional database consists of three components (or sub-languages): a data definition language, a data manipulation language, and a data control language.

A data model that captures object-orientation concepts is an object-oriented data model. An object-oriented database is a collection of objects where the behavior and state, and the relationships are defined in accordance with an object-oriented data model. An object-oriented database system is a system which allows the definition and manipulation of an object-oriented database system.

Object-oriented database systems [Sun 1992, Mariani 1993, Hurson 1993] integrate techniques from database systems and object-oriented paradigms, therefore,

they have been regarded as the next generation of commercial database systems. An object-oriented database system is both a database system and an object-oriented system. As a database system, it provides support for accessing and updating large amounts of persistent, reliable, and shared data. As an object-oriented system, it supports features such as complex objects with identity, inheritance (of classes or types), encapsulation (of an object state by the methods defined on its class), overriding (redefining methods in classes), and run-time binding of methods to objects. The difference between conventional relational database and object-oriented database is shown as Figure 1.

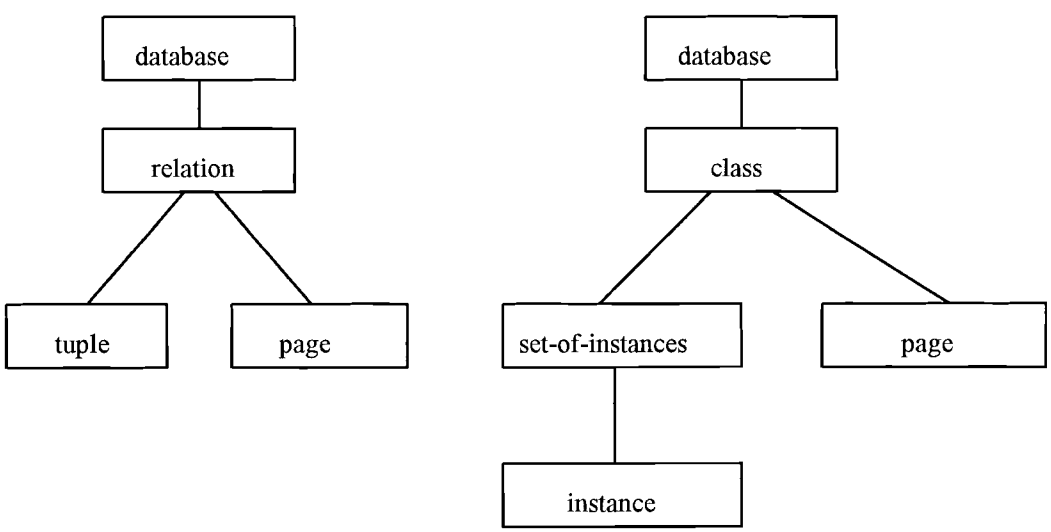


Figure 1. The difference between conventional relational database and object-oriented database

Survey of Current OODBS

Currently, few commercial object-oriented database systems are available. Also there exists no single, widely agreed upon object data model in the sense of the

relational model. Generally, most of the OODB system designers have followed the three principal approaches. These are:

(1) A programming-language-neutral object model extends relations or semantic data models with object-oriented features.

(2) With a database programming language, a new programming language is defined from the group to support database capabilities.

(3) In the programming-language-specific approach, the object data model is equivalent to the type of system of an existing programming language, with the objective of eliminating the mismatch between the programming language and the database.

Following, more than ten representative operational object-oriented database systems are described, most of them are designed and implemented by using one of the three approaches described above.

Gemstone

GemStone [Bertino 1989] supports a client/server architecture, which developed by extending the Smalltalk-80 into database system. Although the Gemstone database model is an extension of Smalltalk-80, it only supports single inheritance. The design approach of GemStone is rather different from most other systems with respect to persistence. In sum, GemStone is one of the more visible object-oriented database systems available today, both in term of the database features provided and contributions to research into object-oriented database architecture.

VBASE

VBASE [Andrews 1987] is implemented in C, and runs on SUN workstations under a 3.2 UNIX operating system. It is implemented in a client/server architecture

and it provides two programmatic interfaces: Type Definition Language and C Object Process. All objects known to VBASE become persistent and they persist until users explicitly delete them. Also noteworthy, most parts of the system are designed and implemented by taking advantage of the separation of the type specifications and their implementation.

Statice

Statice [Weinreb 1988] is implemented in Common LISP to run on the Symbolic LISP machine. It uses a client/server architecture, and can support multiple servers. The Statice data model is largely based on the DAPLEX functional data model. Statice provides a fairly rich set of database features, including transactions, concurrence control, recovery, a query language, query optimization, indexing, and multimedia management. The Statice query language is a rather natural extension of the LISP syntax.

IRIS

The IRIS [Fishman 1987] system is implemented in C and PASCAL, and runs on the HP68000 workstation. IRIS provides object management services to C programs and LISP programs through a C-interface and a LISP-interface to IRIS. The data model of IRIS is based on the DAPLEX functional data model and it allows explicit deletion of persistent objects. The elegant aspect of IRIS is the consistent use of a function formalism in its data modeling and novel aspect of IRIS is the extendibility built into its architecture.

O2

O2 [Velez 1989] is implemented in C and runs on SUN workstations under SUN OS 4.0. O2 is designed to support multiple object-oriented paradigms under a client/server environment. The O2 programmatic interface is embedded in C and Basic, and the object manager of O2 has been implemented on top of the Wisconsin Storage System (WiSS), which is a simplified version of the RSS storage system in SQL/DS. Also, the O2 data model supports multiple inheritance and is designed to run differently in a development mode and a production mode.

Jasmin

Jasmin [Makinouchi 1988] has been developed in Fujitsu, Ltd. Japan. The data model of Jasmin is based on the DAPLEX functional data model and is similar to the IRIS and Statice data models. It also includes multiple inheritance. The programmatic interface is called Jasmin/C and is an object-oriented extension of C. One of the goals of Jasmin is the seamless integration of a programming language and a database system.

ENCORE/ObServer

ENCORE/ObServer [Hornick 1987] is implemented in C. ENCORE is the front-end object manager and Observer is the back-end object server. ObServer runs on SUN workstations under UNIX while ENCORE runs on SUN workstations and DEC machines under VMS. It is designed for a cooperative computer-aided design environments. The ENCORE data model includes multiple inheritance and multi-valued attributes. The ObServer stores multimedia data and supports locking-based concurrence control, log-based recovery and transactions.

POSTGRES

POSTGRES [Rowe 1987] is a sequel to the INGRES relational database system, which along with System R from IBM's San Jose Research Lab, played a major role in ushering in the area of relational databases. The POSTGRES data model includes some basic object-oriented concepts. One of the noteworthy aspects of the POSTGRES is the proposed support for historical data besides data versions. Another novel aspect of the POSTGRES design is the incorporation of advances in hardware technology into the database system architecture.

AVANCE

AVANCE [Bjoninerstedt 1988] is designed for use as a tool for office applications development. The user of the AVANCE program uses the PAL language which is a strongly typed programming language largely influenced by Simula, Smalltalk, and CLU. AVANCE is designed to be a fully distributed object-oriented database system. It is the only object-oriented database system that supports nested transactions. One of the aspects of AVANVCE is its use of a multi-version protocol for concurrence control.

OZ+

OZ+ [Weiser 1989] is implemented in C and Turing Plus and runs on SUN workstations under UNIX. The system, like AVANCE, is designed to facilitate the implementation of office applications and the modeling of office activities. The OZ+ data model is noteworthy in its combination of the complex-object representation of objects and the actor model of communication among active objects. The data model supports single inheritance.

EXTRA

The EXTRA object manager [Carey 1988] is being contracted on top of the EXODUS extensible storage system. The data model of EXTRA is a rich amalgamation and extension of the data models of O2, GemStone, POSTGRES, etc. It includes multiple inheritance composite objects, collections and set and array type constructors. For the implemented of EXODUS, a new programming language called E was designed and implemented by extending C++. EXTRA contains a number of features appropriate for the implementation of a database system.

CHAPTER III

ANALYSIS OF THE AQUIRE DATABASE

The original AQUatic toxicity Information and REtrieval database (AQUIRE) and the description files are stored in one source tape. The AQUIRE database is implemented by VAX FORTRAN language in binary format running on the VMS operating system on a DEC VAX 11/785 computer. No retrieval and manipulation software is contained in the source tape.

The completed AQUIRE database contains files 10 to 18 as text or binary data files and text files 1 to 9 as corresponding description files. File 11 consists of the main information for each record in the AQUIRE database and files 10, 12 to 18 consist of additional information in the different areas for the records in file 11. File 1 describes the general information for the AQUIRE database and the first data file which is the Chemical Abstract Service (CAS) registry numbers and the chemical names for all the chemicals in the AQUIRE database. Files 2 to 9 are in FORTRAN COMMON block descriptions showing the records format for data files 11 to 18 respectively. The purity / chemical characteristic data file 16 appends by trailing one blank character per record for blocking purposes. Table 1 shows the files names, characters per line for all the original text files in the AQUIRE database. Table 2 shows the file names, maximum record number, record sizes and their contents for all the original data files in the AQUIRE database.

RELATIONSHIPS AMONG THE AQUIRE DATABASE FILES

The AQUIRE database consists of nine fixed-size record data files. File 11 is the main data file which contains 6 external record pointers to files 13 to 18. One foreign key matches the candidate key in file 12. Each record in files 14 to 18 contains

TABLE 1

LIST OF THE ORIGINAL AQUIRE TEXT FILES

File name	Char per line (Byte)	File size(Byte)	File Contents
1	80	4640	Description file for the database and file10
2	80	5280	Description file for file 11
3	80	2880	Description file for file 12
4	80	1680	Description file for file 13
5	80	2560	Description file for file 14
6	80	2080	Description file for file 15
7	80	1760	Description file for file 16
8	80	1680	Description file for file 17
9	80	1920	Description file for file 18

TABLE 2

LIST OF THE ORIGINAL AQUIRE DATA FILES

File Name	Record size (Byte)	Record no.	Files Size (Byte)	File Contents
10	136	5,392	733,312	CAS registry no. and chemical name
11	160	105,392	16,863,040	AQTOX main information
12	544	9,952	5,413,888	Citation Information
13	68	2,184	191,420	Species Information
14	24	117,103	2,2810,472	CAS number in each test
15	20	369,003	7,380,060	Concentration-Conf. int-BCF
16	45	117,100	5,269,500	Purity / Chemical characteristics
17	72	260,816	18,778,752	Remark Information
18	16	777,825	12,445,200	Temp-Hardness-Alk-D.O.-pH

one pointer field to the corresponding record in the main data file 11. The CAS number is the primary key in files 10 and foreign key in file 14.

Besides the external record pointers to other corresponding files, some of the files also contain the internal record pointers to the other records in the same files. File 14 contains one internal record pointer to the next chemical in the same test, two internal record pointers to the predecessor and successor locations with the same CAS hash #. File 15 contains one internal record pointer to the next Concentration-Conf. int-BCF record which can link to up to 12 records in this file to make up the complete information. Files 16 to 18 contain the similar internal record pointers which can link to up to 6, 7 and 15 records in the same file respectively.

There are three types of relationships among these nine data files, such as many-to-one, one-to-many and one-to-one relationships. For example, there is a one-to-one relationship between files 11 and 13, and there are one-to-many relationships between files 10 and 14, files 11 and 12, and files 11 and 15 to 18. Table 3 shows the list of the external/internal record pointer fields and other fields in the data files.

OBJECT-ORIENTED ANALYSIS MODEL (OOA / Coad-Yourdon)

Object-oriented analysis is an analysis model developed to describe the functionality of the system. Coad and Yourdon extended this model with respect to processes, human interfaces and DBMS issues. Coad and Yourdon view their OOA methodology as building upon the best concept from information modeling, object-oriented programming languages, and knowledge, as a system. OOA results in a five-layer model of the problem domain, where each layer builds on the previous layer. The layer model is constructed using a five-step procedure.

TABLE 3

LIST OF THE EXTERNAL/INTERNAL RECORD POINTERS AND OTHER FIELDS IN NINE DATA FILES

File names	Ext. record pointers	Int. record pointers	Other field names
10	None	None	CASNo., Chemical Name
11	zcasptr, zchrptr, zispec, zccbptr, zthaptr, zremptr	None	zflag, zupdate, zrefnum, zrploc, zrsloc, zauthor, zyear, zaploc, zasloc, ztest1, ztest2, zrevur, zrcode, zlfield, zfwsw, zstudy, zsploc, zssloc, zlifstg, zcontrol, zeffect, zresvrd, zcctyp1, zcctyp2, zbctyp1, zbctyp2, zbtime1, zrtime1, zbtime2, zrtime2, ztimeun, zexpty, zconc, zfill
12	None	None	author, year, yeara, title, source, refnum, reftyp, aploc, asloc, rploc, rsloc, mi, di, yi, mu, du, yu, litev, garmark
13	None	None	latin, common, major, minor, spref
14	casaqloc	casploc, cassloc, nextcas	casnum, castype
15	ccbaploc	nextccb	ccbcf, ccbfield
16	chraploc	nextchr	purity, chrtype, chrfield, chrfill
17	remaploc	nextrem	remark, remfield, remfill
18	thaaploc	nexttha	thadp, thafield

(1) Define object and classes

(2) Define structure

(3) Define subject areas

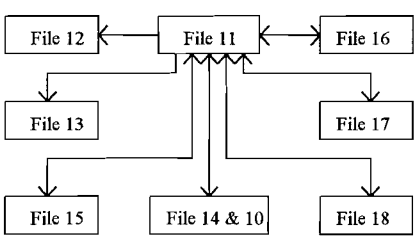
(4) Define attributes

(5) Define services

The primary tools for Coad and Yourdon OOA are class and object diagrams and series charts. Coad and Yourdon explicitly support each of the essential principles

of object orientation. In addition, encapsulation of objects is modeled through the concept of exclusive series.

By applying OOA/Coad-Yourdon methodology, the Object-Oriented Aquatic Toxicity Database Management System (OOATDBMS) is defined as two modules of objects, such as database and retrieval system objects and user interface objects. The objects in the database and retrieval system are redefined as eight objects, files 11, 12, 13, 15, 16, 17 and 18 are defined as seven objects respectively, files 10 and 14 are defined as one object due to the closed relationship of the CAS registry number. These objects and the associated manipulation procedures are defined as classes. Each class consists of its data members which are the data fields in each data file; and function members which are the procedures used to manipulate its data members. There are message passing relationships between these eight classes which are represented as the external pointers relationships among the original AQUIRE data files. Figure 2 shows the eight database classes and their relationships.



Note: A ⟶ B means A contain an external record pointer to B

A ⟷ B means A and B contain external record pointers to each other

Figure 2 The eight database classes and their relationships

The user interface objects consist of a set of visual objects associated with the windows, views, menus, icons, dialog boxes, etc., which are used to display the information on the screen after the information has been retrieved from the AQUIRE database. These visual objects and their relationships are predefined in Microsoft Visual C++ object-oriented development environment for all the MS Visual C++ Windows

applications. The OOATDBMS needs to derive its own user interface objects and define the interactive relationships between themselves and between the user interface objects and database and retrieval system objects.

CHAPTER IV

DESIGN OF THE OBJECT-ORIENTED AQUATIC TOXICITY DATABASE MANAGEMENT SYSTEM (OOATDBMS)

Design is the process of mapping system requirements defined during analysis to an abstract representation of a specific system-based implementation, meeting cost and performance constraints. Compared to OOD, none of the conventional design methodologies supports the definitions of classes, inheritance, methods or message protocols, even though both of the methodologies provide the tools that define a hierarchy of modules.

Design Database Retrieval Strategy

The original AQUIRE Database consists of nine data files, some of the them already contain external and / or internal record pointers to other files or other records in the same files. For example, file 11 contains the external record pointers to files 13 to 18. Files 14 to 18 contain the external record pointers to file 11. File 14 contains internal record pointers also. However, some of the files do not contain any external or internal record pointer indicating the relationships between each other and some of them only have the key matching relationships. For example, there is no pointer to each other among files 12 to 18. The relationship between file 11 and file 13 is established by matching the *zispec* field in file 11 with the record no. of file 13, and the relationship between file 11 and file 14 is established by matching the *ziref* field in file 11 with the *zrefnum* field in file 14.

The goal of designing an effective database retrieval strategy is to provide users with accurate information by multiple data retrieval methods and fast retrieval response time. The original AQUIRE Database is a very large database since the total database occupies about 80 MB. The maximum data records in one file is 777,825 and up to six

files contain more than 100,000 data records in each file. In this case, the database retrieval strategy for Object-Oriented Aquatic Toxicity Database Retrieval System (OOATDBMS) is designed as follows.

Multiple Data Retrieval Methods

(1) To retrieve main information of a record in AQUIRE database, users may enter either one of the following fields on the AQUIRE main window: its record number, Citation reference number, CAS file record number, Species record number, purity /chemical record number, conf.-int-BCF record number, temp-hardness record number; or click the Aquire push button on other windows;

(2) To retrieve Citation information of a record, users may enter the reference number on citation information window; or click the Citation push button on other windows;

(3) To retrieve Species information of a record, users may enter either one of the following fields on the Species windows: latin name, common name, major code, minor code, species reference number, its record number; or click the Species push button on other windows;

(4) To retrieve CAS information of a record, users may enter its record no. or click the Previous or Next push button to retrieve the previous or next record with the same CAS # on CAS window; or click the CAS push button on other windows;

(5) To retrieve Concentration conf.-int-BCF information of a record, users may enter the Refer push button on a window other than concentration conf.-int-BCF information;

(6) To retrieve Purity/chem characteristic information of a record, users may click the Purity push button on a window other than purity/chem characteristic information;

(7) To retrieve Remark information of a record, users may click the Remark push button on a window other than remark information;

(8) To retrieve Temp-Hardness-Alk-D.O.-pH information of a record, users may click the Temp push button on a window other than Temp-Hardness-Alk-D.O.-pH information.

Fast Retrieval Response Time

(1) retrieve the data record by directly accessing the file using its record no.;

(2) create indexes for the files;

(3) by using file 11 as an intermediate bridge, visually establish the relationships among files 12 to 18 if they do not contain any record pointers to each other.

Design Indexes

An index is a special kind of sorted file, which is generally used to speed up the retrieval. The file which has an index associated with it is called an indexed file. There are two types of indexes: one is called primary index where the index key is the primary key field in the indexed file; another is called secondary index where the index key is the field other than the primary key field in the indexed file. In OOATDBMS, there are three files which need to create the indexes.

File 11 (AQUIRE main information) contains 105,392 records with the fields *ziref* and *zispec* unsorted. In order to quickly retrieve file 11 by a given *ziref* or *zispec*, it is necessary to create two secondary indexes, one for mapping the *ziref* value to record number, another for mapping the *zispec* value to the record number.

File 12 (Citation information) contains 9,952 records with the citation reference numbers unsorted. In order to quickly retrieve file 12 by a given *zrefnum*, it is

necessary to create one secondary index for mapping the *zrefnum* value to the record no.

File 13 (Species information) contains 2,815 records with latin name, common name, major code, minor code and species reference number unsorted. In order to quickly retrieve species information, it is necessary to create one primary index for latin name and four secondary indexes for common name, major code, minor code and species reference number individually. Each index only contains two fields, i.e. the corresponding indexed field and the record no. in file 13. Table 4 shows the detail information of eight indexes designed for the OOATDBMS.

TABLE 4

LIST OF INDEXES IN THE OOATDBMS

Index file	Max record no.	Search key	Associated file
aqziref.idx	105,392	ziref	File 11
aqzispec.idx	105,392	zispec	File 11
retoxref.idx	9,952	zrefnum	File 12
latin.idx	2,814	latin	File 13
common.idx	2,814	common	File 13
major.idx	2,814	major	File 13
minor.idx	2,814	minor	File 13
spref.idx	2,814	spref	File 13

The procedures used to create indexes are listed as follows:

(1) read the search fields from the indexed file and write them with the record no. into a temporary file;

(2) sort the temporary file using a random disk access sorting algorithm [Schildt 1990] because the file size is too large to be read into the memory at one time;

(3) save the sorted file as an index;

(4) repeat steps (1) to (3) until eight indexes have been created.

After eight sorted indexes have been created, searching the indexes by applying the binary search algorithm will achieve very good performance because the time complexity of binary search algorithm is pretty good. The number of comparisons in the best case is one, the number of comparisons in the worse case is $\lfloor \log_2 N \rfloor + 1$ for $N \geq 1$, and the number of comparisons in the average case is approximately $\lfloor \log_2 N \rfloor + \frac{1}{2}$ for $N \geq 1$.

With the pre-existed external and internal record pointers and eight indexes, all the data files in the AQUIRE Database are visually linked to each other and can be retrieved by a direct access method.

These two sections discuss the database retrieval strategy and the procedures for design indexes, which are the basic steps of the database retrieval system design. In the next section, the design of OOATDBMS is discussed using the object-oriented design (OOD/Booch) method, which is intended to implement the above retrieval strategy.

Object-Oriented Design (OOD/Booch)

Object-oriented design (OOD/Booch) method is very similar to the OOA/Coad-Yourdon method of finding objects, but the OOD/Booch method focuses more on establishing a grounding for the implementation. Booch employs a detailed precedence (where appropriate) for designing the data encapsulated within objects. He delineates four major steps that must be performed during the course of the OOD:

(1) Identify the class and objects;

(2) Identify the semantics of classes and objects;

(3) Identify relationships between classes and objects;

(4) Implement classes and objects.

The primary tools used during the OOD / Booch method are:

- Class diagram and class template
- State diagram and timing diagrams
- State-transition diagram
- Operation templates
- Module diagrams and templates
- Process diagrams and templates

Design Database Classes and Objects

By applying the OOD / Booch method, the system design of the Object-Oriented Aquatic Toxicity Database Management System is identified as two major subsystems, one is a database retrieval subsystem and another is a user interface subsystem. The database retrieval subsystem consists of eight classes which are designed for data files 11 to 18. The data members of each class are the fields in each file plus one index field called thisRecordNo, which is used to indicate the current record no. in each file. The function members of each class are the constructor, destructor and some special functions associated with each class. Table 5 shows the database classes names, their data member number, function member number and their relationships with the original AQUIRE data files.

The detail information of each class are listed as follows:

Class name: CAqtoxClass

Description: This class is designed for data file 11, which is the AQUIRE main information file. This class is used to store the current AQUIRE main information for the record which has just been retrieved from file 11. The class provides three associated functions besides the constructor and destructor. There are functions to return the

TABLE 5

LIST OF THE DATABASE CLASSES AND OBJECTS

Classes names	Data member no.	Function member no.	Related to data files
CAqtoxClass	41	5	11
CRetoxClass	20	4	12
CSpeciesClass	6	11	13
CStocasClass	8	5	14
CStoccbClass	18	4	15
CStochrClass	14	4	16
CStoremClass	9	4	17
CStothaClass	17	4	18

current record no., to retrieve the record by a given record no. and to search the citation record no. by a given citation reference number (*ziref*).

Data members: thisRecdNo, zflag, zupdate, zcasptr, zchrptr, ziref, zrploc, zrsloc, zauthor, zyear, zaploc, zasloc, ztest1, ztest2, zrevur, zrcode, zlfield, zafwsw, zstudy, zispec, zsploc, zssloc, zlifstg, zcontrl, zeffect, zresvrd, zccbptr, zcctyp1, zcctyp2, zbctyp1, zbctyp2, zbtime1, zrtime1, zbtime2, zrtime2, ztimeun, zexpty, zconc, zthaptr, zremptr,zfill;

Function members: aqtoxClass(), ~aqtoxClass(), long getRecdNo(), ItemStruct searchRecdNum(ifstream& f, long l), int getRetoxRecdNo(ifstream& f, long keyRefNum);

Class name: CRetoxClass

Description: This class is designed for data file 12, which is the citation information file. This class is used to store the current citation

information for the record which has just been retrieved from file 12. The class provides two associated functions besides the constructor and destructor. There are functions to retrieve the record by a given record no. and to search the aqtox record no. by a given citation reference number.

Data members: thisRecdNo, author, year, yeara, title, source, refnum, reftyp, aploc, asloc, rploc, rsloc, mi, di, yi, mu, du, yu, litrev, garmark;

Function members: CRetoxClass(), ~CRetoxClass(),
RetoxStruct searchRecdNum(long i),
long getAqtoxRecdNo(long keyRefNum);

Class name: CSpeciesClass

Description: This class is designed for data file 13, which is the species information file. This class is used to store the current species information for the record which has just been retrieved from file 13. The class provides eight associated functions besides two constructors and one destructor. There are functions to return the current record no., to retrieve the record by a given record no., to search the aqtox record no. by a given species record number, to search the species record no. by a given latin name, or common name, or major code, or minor code, or species reference number.

Data members: thisRecdNo, Latin, Common, Major, Minor, Spref;

Function members: CSpecies(), ~CSpecies(), CSpecies(specStruct sp),
long getRecordNo(),
specStruct searchRecordNo(ifstream& f, long i),
long getAqtoxRecdNo(ifstream& f, long keySpec),
long searchByLatin(ifstream& f, CString latin),

long searchByCommon(ifstream& f, CString common),
long searchByMajor(ifstream& f, CString major),
long searchByMinor(ifstream& f, CString minor),
long searchBySpref(ifstream& f, short spref);

Class name: CStocasClass

Description: This class is designed for data files 10 and 14, which is the chemical abstract service (CAS) registry number and chemical name. This class is used to store the current CAS information such as CAS number, its chemical name and type for the record which has just been retrieved from files 10 and 14. The class provides three associated functions besides the constructor and destructor. There are functions to return the current record no., to retrieve the record by a given record no., to search the CAS chemical name information from file 10 by a given CAS number.

Data members: thisRecdNo, nextcas, casnum, casploc, cassloc, casaqloc, castype, chemName;

Function members: CStocasClass(), ~CStocasClass(), long getRecordNo(),
stocasStruct searchRecdNum(long l),
char* searchCASRecdNo(long casNum);

Class name: CStoccbClass

Description: This class is designed for data file 15, which is the Concentration-Conf int-BCF information file. This class is used to store the current concentration-Conf int-BCF information for the record which has just been retrieved from file 15. The class provides two associated functions besides the constructor and destructor. There

are functions to return the current record no. and to retrieve the record information by a given record no.

Data members: thisRecdNo, mBcfFrom1, mBcfFrom2, mBcfTo1, mBcfTo2, mBcfType1, mBcfType2, mConcFrom1, mConcFrom2, mConcTo1, mConcTo2, mConcType1, mConcType2, mInterFrom1, mInterFrom2, mInterTo1, mInterTo2, ccbaqloc;

Function members: CStoccbClass(), ~CStoccbClass(), long getRecordNo(), stoccbStruct searchRecordNo(long i);

Class name: CStochrClass

Description: This class is designed for data file 16, which is the purity / chemical characteristic information file. This class is used to store the current purity / chemical characteristic information for the record which has just been retrieved from file 16. The class provides two associated functions besides the constructor and destructor. There are functions to return the current record no. and to retrieve the record information by a given record no.

Data members: thisRecdNo, chrAqloc, mChemchar1, mChemchar2, mChemchar3,mChemchar4, mChemchar5, mChemchar6, mPurity1, mPurity2, mPurity3, mPurity4, mPurity5, mPurity6;

Function members: CStochrClass(), ~CStochrClass(), long getRecordNo(), stochrStruct searchRecordNo(long i);

Class name: CStoremClass

Description: This class is designed for data file 17, which is the remark information file. This class is used to store the current remark information for the record which has just been retrieved from file

17. The class provides two associated functions besides the constructor and destructor. There are functions to return the current record no. and to retrieve the record information by a given record no.

Data members: thisRecdNo, remaqloc, remark[7][61];

Function members: CStoremClass(), ~CStoremClass(), long getRecordNo(), storemStruct searchRecordNo(long l);

Class name: CStothaClass

Description: This class is designed for data file 18, which is the temp-hardness-Alk-D.O.-pH information file. This class is used to store the temp-hardness-Alk-D.O.-pH information for the record which has just been retrieved from file 18. The class provides two associated functions besides the constructor and destructor. There are functions to return the current record no. and to retrieve the record information by a given record no.

Data members: thisRecdNo, mThaaqloc, mFrom[5][7], mTo[5][7], mValue[5][7];

Function members: CStothaClass(), ~CStothaClass(), long getRecordNo(), stothaStruct searchRecordNo(long l);

Design User Interface Classes and Objects

The User interface classes and objects contain windows, menus, views, dialog boxes, and controls such as buttons, edit boxes, etc. They functionally co-operate together to display information on the screen and take users input. Users can open or close windows and views, click on the menus item to activate windows, and click push buttons to activate other views or dialog boxes, etc. All of these operations are parts of

the functions provide by the user interface classes and objects [Collins 1995, Johnsonvaugh 1995, Neilsen 1995].

In the OOATDBMS, the user interface classes and objects are basically consists of one main window with one menu screen, eight derived view classes, 94 edit boxes, and 70 push buttons. Besides the basic menu items included in all standard windows menu screen, the main window menu screen contains one additional menu item called Retrieve with eight sub-menu items. There are: AQUIRE, Citation, Species, CAS, Purity / Chemical, Conf. int-BCF, Remark, Temp-Hardness-Alk-D.O.pH. Each item corresponds to one derived formview window. By selecting one of eight sub-menu items, the corresponding view window will be created if it does not exist, or will be activated if it already exists.

Some of the edit boxes in the eight derived views are the places where users can input the search keys. Each view class object corresponds to one database class object. It contains 4 to 25 edit boxes, 8 to 11 push buttons, and 12 to 15 message handling functions. The edit boxes are used to accept user's search keys and /or hold the information which is needed to be displayed on the screens. The push buttons are used to accept users' control messages, such as transferring control to other views or closing the view, etc. The message handling functions are used to maintain the information changed in the edit boxes and process users' input control messages. Table 6 shows the view names, number of edit boxes, number of push buttons, number of functions and associated database classes names for the view class objects.

The detail information of each view are listed as follows:

View name: CAqtoxView

Description: This view class is designed for CAqtoxClass to display information contained in the CAqtoxClass instance and accept users' input data. The information is displayed in 25 edit boxes. Users can input search keys on eight different fields, i.e., mRecdNum, mRetoxPtr,

TABLE 6

LIST OF EIGHT VIEW OBJECTS AND THEIR INFORMATION

ViewNames	Edit Boxes	Push Buttons	No. of Func.	Associated DB Files
CAqtoxView	25	9	13	CAqtoxClass
CRetoxView	9	9	13	CRetoxClass
CSpeciesView	6	9	13	CSpeciesClass
CStocasView	4	11	15	CStocasClass
CStoccbView	16	8	13	CStoccbClass
CStochrView	12	8	12	CStochrClass
CStoremView	7	8	12	CStoremClass
CStothaView	15	8	12	CStothaClass

mSpecNo, mCasRecd, mCcbPtr, mChrPtr, mRemarkPtr, mTempPtr, and click the Search button on the screen or press ENTER from the keyboard. The retrieval information will be displayed if the record exists, otherwise, an error message will be displayed. If the user clicks buttons other than the Search button, for example, the Citation button, the window control will be transferred to CRetoxView and the CRetoxView with the retrieval citation information, which corresponds to the CAqtoxClass record information, will be displayed if the record exists, otherwise, an error message will be displayed. There are nine push buttons, seven of them for transferring control to other view class objects, two of them for searching the records and closing the current view.

Edit boxes: mAuthor, mCasRecd, mCcbPtr, mChrPtr, mControl, mEffect, mExposure, mLifeStage, mMedia, mMethod, mRetoxPtr,

mReviewer, mCode, mSpecNo, mStudy, mTempPtr, mTestField, mTim1, mTime2, mTotalTest, mYear, mDate, mTestNo, mRecdNum, mRemarkPtr;

Push buttons: mCitationCtrl, mCASCtrl, mTempCtrl, mSpecCtrl, mRemarkCtrl, mRefCtrl, mPurityCtrl, mSearch, mClose;

Functions: void OnClickButton(int bt), int mySearch(), virtual ~CAqtoxVw(), virtual void DoDataExchange(CDataExchange* pDX), void OnClickedAqtoxClose(), void OnClickedAqtoxSearch(), void OnClickedAqtoxCas(), void OnClickedAqtoxCitation(), void OnClickedAqtoxPurity(), void OnClickedAqtoxReference(), void OnClickedAqtoxRemark(), void OnClickedAqtoxSpecies(), void OnClickedAqtoxTempHard();

View name: CRetoxView

Description: This view class is designed for CRetoxClass to display information contained in the CRetoxClass instance and accept users' input data. The information are displayed in 9 edit boxes. Users can input search keys on two different fields, i.e., mRetoxRecdNo, mRetoxRefNum, and click the Search button on the screen or press ENTER from the keyboard. The retrieval information will be displayed if the record exists, otherwise, an error message will be displayed. If the user clicks buttons other than the Search button, for example, the AQUIRE button, the window control will be transferred to CAqtoxView and the CAqtoxView with the retrieval AQUIRE main information, which corresponds to the CRetoxClass record information, will be displayed if the record exists,

otherwise, an error message will be displayed. There are nine push buttons, seven of them for transferring control to other view class objects, two of them for searching the records and closing the current view.

Edit boxes: mRetoxAuthor, mRetoxInsertDate, mRetoxRecdNo, mRetoxRefNum, mRetoxRefType, mRetoxSource, mRetoxTitle, mRetoxUpdateDate, mRetoxYear;

Push buttons: mRetoxTemp, mRetoxSpecies, mRetoxRemark, mRetoxRefer, mRetoxPurity, mRetoxCAS, mRetoxClose, mRetoxSearch, mRetoxAquire;

Functions: void OnClickButton(int bt), int mySearch(), virtual ~CRetoxVw(), virtual void DoDataExchange(CDataExchange* pDX), void OnRetoxSearch(), void OnClickedRetoxAquire(), void OnRetoxClose(), void OnRetoxCas(), void OnRetoxPurity(), void OnRetoxRefer(), void OnRetoxRemark(), void OnRetoxSpecies(), void OnRetoxTemp();

View name: CSpeciesView

Description: This view class is designed for CSpeciesClass to display information contained in the CSpeciesClass instance and accept users' input data. The information is displayed in 6 edit boxes. Users can input search keys on all six fields, i.e., mLatin, mCommon, mMajor, mMinor, mSpref, mSpecNum, and click the Search button on the screen or press ENTER from the keyboard. The retrieval information will be displayed if the record exists,

otherwise, an error message will be displayed. If the user clicks buttons other than the Search button, for example, the AQUIRE button, the window control will be transferred to CAqtoxView and the CAqtoxView with the retrieval AQUIRE main information, which corresponds to the CSpeciesClass record information, will be displayed if the record exists, otherwise, an error message will be displayed. There are nine push buttons, seven of them for transferring control to other view class objects, two of them for searching the records and closing the current view.

Edit boxes: mLatin, mCommon, mMajor, mMinor, mSpref, mSpecNum;

Push buttons: mSpeciesTemp, mSpeciesRetox, mSpeciesRemark, mSpeciesPurity, mSpeciesRefer, mSpeciesCAS, mClose, mSearch, mAquire;

Functions: void OnClickButton(int bt), int mySearch(), virtual ~CSpecView(), virtual void DoDataExchange(CDataExchange* pDX), void OnClickedClose(), void OnClickedAquire(), void OnClickedSearch(), void OnSpeciesCas(), void OnSpeciesPurity(), void OnSpeciesRefer(), void OnSpeciesRemark(), void OnSpeciesRetox(), void OnSpeciesTemp();

View name: CStocasView

Description: This view class is designed for CStocasClass to display information contained in the CStocasClass instance and accept users' input data. The information is displayed in 4 edit boxes. Users can input search keys on one field, i.e., mCASRecordNo and click the

Search button on the screen or press ENTER from the keyboard. The retrieval information will be displayed if the record exists, otherwise, an error message will be displayed. If the user clicks buttons other than the Search button, for example, the AQUIRE button, the window control will be transferred to CAqtoxView and the CAqtoxView with the retrieval AQUIRE main information, which corresponds to the CStocasClass record information, will be displayed if the record exists, otherwise, an error message will be displayed. There are eleven push buttons, seven of them for transferring control to other view class objects, two of them for retrieving previous and next record with the same CAS number and two of them for searching the records and closing the current view.

Edit boxes: mCASNum, mChemName, mChemType, mCASRecordNo;

Push buttons: mStocasSpecies, mStocasTemp, mStocasRemark, mStocasRefer, mStocasPurity, mStocasCitation, mStocasAquire, mPreview, mSearch, mNext, mClose;

Functions: void OnClickButton(int bt), int mySearch(), virtual ~CStocasVw(), virtual void DoDataExchange(CDataExchange* pDX), void OnClickedStocasNext(), void OnClickedStocasSearch(), void OnClickedStocasPreview(), void OnStocasClose(), void OnStocasAquire(), void OnStocasCitation(), void OnStocasPurity(), void OnStocasRefer(), void OnStocasRemark(), void OnStocasSpecies(), void OnStocasTemp();

View name: CStoccbView

Description: This view class is designed for CStoccbClass to display information contained in the CStoccbClass instance in 16 edit boxes. There are eight push buttons, seven of them for transferring control to other view class objects and one of them for closing the current view.

Edit boxes: mStoccbBcfFrom1, mStoccbBcfFrom2, mStoccbBcfTo1, mStoccbBcfTo2, mStoccbBcfType1, mStoccbBcfType2, mStoccbConcFrom1, mStoccbConcFrom2, mStoccbConcTo1, mStoccbConcTo2, mStoccbConcType1, mStoccbConcType2, mStoccberFrom1, mStoccberFrom2, mStoccberTo1, mStoccberTo2;

Push buttons: mStoccbTemp, mStoccbSpecies, mStoccbRemark, mStoccbPurity, mStoccbCitation, mStoccbCAS, mStoccbClose, mStoccccbAquire,

Functions: void OnClickButton(int bt), int mySearch(), virtual ~stoccbVw(), virtual void DoDataExchange(CDataExchange* pDX), void OnClickedStoccbAquire(), void OnStoccbSearch(), void OnStoccbClose(), void OnStoccbCas(), void OnStoccbCitation(), void OnStoccbPurity(), void OnStoccbRemark(), void OnStoccbSpecies(), void OnStoccbTemp();

View name: CStochrView

Description: This view class is designed for CStochrClass to display information contained in the CStochrClass instance in 12 edit boxes. There are eight push buttons, seven of them for transferring control to other view class objects and one of them for closing the current view.

Edit boxes: mStochrChemchar1, mStochrChemchar2, mStochrChemchar3, mStochrChemchar4, mStochrChemchar5, mStochrChemchar6,

mStochrPurity1, mStochrPurity2, mStochrPurity3,
mStochrPurity4, mStochrPurity5, mStochrPurity6;

Push buttons: mStochrCAS, mStochrTemp, mStochrSpecies, mStochrRemark, mStochrRefer, mStochrCitation, mStochrClose, mStochrAquire;

Functions: void OnClickStochrSearch(), int mySearch(), virtual ~CStochrView(), virtual void DoDataExchange(CDataExchange* pDX), void OnClickedStochrAquire(), void OnStochrClose(), void OnStochrCas(), void OnStochrCitation(), void OnStochrRefer(), void OnStochrRemark(), void OnStochrSpecies(), void OnStochrTemp();

View names: CStoremView

Description: This view class is designed for CStoremClass to display information contained in the CStoremClass instance in 7 edit boxes. There are eight push buttons, seven of them for transferring control to other view class objects and one of them for closing the current view.

Edit boxes: mStoremRemark1, mStoremRemark2, mStoremRemark3, mStoremRemark4, mStoremRemark5, mStoremRemark6, mStoremRemark7;

Push buttons: mStoremTemp, mStoremSpecies, mStoremRefer, mStoremPurity, mStoremCitation, mStoremCAS, mStoremClose, mStoremAquire;

Functions: void OnClickStoremSearch(), int mySearch(), virtual ~CStoremView(), virtual void DoDataExchange(CDataExchange* pDX), void OnClickedStoremAquire(), void OnStoremClose(),

void OnStoremTemp(), void OnStoremSpecies(),
void OnStoremRefer(), void OnStoremPurity(),
void OnStoremCitation(), void OnStoremCas();

View name: CStothaView

Description: This view class is designed for CStothaClass to display information contained in the CStothaClass instance in 15 edit boxes. There are eight push buttons, seven of them for transferring control to other view class objects and one of them for closing the current view.

Edit boxes: mStothaFrom1, mStothaFrom2, mStothaFrom3, mStothaFrom4, mStothaFrom5, mStothaTo1, mStothaTo2, mStothaTo3, mStothaTo4, mStothaTo5, mtothaValue1, mStothaValue2, mStothaValue3, mStothaValue4, mStothaValue5;

Push buttons: mStothaSpecies, mStothaRemark, mStothaRefer, mStothaPurity, mStothaCitation, mStothaCAS, mStothaClose, mStothaAquire;

Functions: void OnClickStothaSearch(), int mySearch(),
virtual ~CStothaView(),
virtual void DoDataExchange(CDataExchange* pDX),
void OnClickedStothaAquire(), void OnStothaClose(),
void OnStothaCas(), void OnStothaCitation(),
void OnStothaPurity(), void OnStothaRefer(),
void OnStothaRemark(), void OnStothaSpecies().

CHAPTER V

IMPLEMENTATION AND TESTING OF THE OOATDBMS

The Object-Oriented Aquatic Toxicity Database Management System (OOATDBMS) is implemented under Microsoft Visual C++ development environment version 2.0 running on Windows NT version 3.5. Windows NT is a new 32-bit operating system that has an advanced file system with security features, multithreading, true preemptive multitasking, enhanced network access, and portability to selected RISC computers. C++ is known as an object-oriented programming language. Microsoft Visual C++ is a complete windows application development system product [Kruglinski 1994, Pappas 1994].

The Basic MS Visual C++ Components

Application Framework

An application framework is a superset of a class library, that is, an integrate collection of object-oriented software components for a generic program. It defines the structure of the program.

App Studio

App Studio is a program used to design the application user interface and create the application's resources; i.e. dialog boxes, custom controls, accelerator keys, bitmaps, icons cursors, and strings.

AppWizard

AppWizard is a code generator that creates a working skeleton of a window application with features, class names, and source code filename.

ClassWizard

ClassWizard is a program that provides the programmer with the prototypes of a new class or function members and variables. It also works with App Studio Editor to create the user-interface objects and generate functions and message-maps for each user-interface objects.

Document

Document objects which are created by the document template objects manage the application's data. The basic class for all the application-specific documents is CDocument class. Each application can derive its own document class from CDocument class.

Microsoft Foundation Class Library (MFC)

Microsoft Foundation Class Library (MFC) is an application framework defined for MS Visual C++. The library's classes are presented as six categories, all the classes in MFC support application development for Microsoft Windows version 3.0 and later. These six categories classes are: Root Class, Application Architecture Classes, Visual Object Classes, General-Purpose Classes, Object Linking and Embedding (OLE) Classes, Macros and Globals. The application can derive its application-specific class from MFC Classes.

Multiple Document Interface (MDI) Application

A multiple document interface application is a kind of application that allows multiple document frame windows to be opened in the same instance of an application. It contains several MDI child windows, each child window contains a separate document.

Single Document Interface (SDI) Application

A single document interface application is a kind of application that allows only one document frame window to be opened in the same instance of an application.

Multiple Document Template

Multiple document template is a template for document in multiple document interface (MDI) application, which allows more than one documents opened at a time.

Single Document Template

Single document template is a template for document in single document interface (SDI) application, which allows only one document opened at a time.

View

View objects represent the client area of a frame window which is used to display, accept, edit or select the input for a document. The basic view class for all the application-specific is CView class. Each application can derive its own view class from four types of view classes, i.e., CView, CScrollView, CFormView, CEditView.

Visual Workbench

Visual workbench is a windows-hosted interactive development environment. It provides the windows application programmers with a lots of development tools, such as App Studio Resource Editor, C/C++ Compiler, Linker, Resource Compiler, and Debugger, etc.

Implementation of the OOATDBMS

A basic application project generated by AppWizard contains an application class, a frame class, a document class and a view class. Only one application class is allowed in each application project, however, there may be more than one frame class, document class and view class objects in the application project depending on the type of the project. For example, a multiple document interface application project may contain more than one frame classes, several types of document classes, each document class may associate with different types of view classes (also called multiple views). There are three models of multiple views on the same document, that is, (1) Several view objects of the same class, each view occupies in a separate MDI document frame window; (2) Several view objects of the same class occupy in the same document frame window; (3) Several objects of different classes occupy in a single frame window.

Based on the design, the implementation of the OOATDBMS is described as following skeleton.

The application is a MDI and multiple view project, which consists of two layers of class objects. The first layer contains four basic class objects for general MDI application, i.e., one application object, one main frame windows object, one main document class object and one main view class object. The second layer contains eight MDI document templates, eight MDI child windows objects, eight derived documents class objects and eight formview class objects. The main frame manages eight MDI child windows class objects, the main document class object manages eight document

class objects, the main view class object manages eight formview class objects, and the application object manages eight MDI documents templates.

Each document class object represents one database class object designed previously. Each formview class object represents one view class object respectively. Figure 3 shows the architecture of the class objects in OOATDBMS.

For this MDI application, eight different multiple document templates are defined. Each template defines the relationship of one frame windows object, one document object and one formview object. TABLE 7 lists the template names, frame objects, document objects, and formview objects.

TABLE 7

LIST OF TEMPLATES AND THE RELATIONSHIPS DEFINED

Template names	Frame objects	Document objects	Formview objects
mTemplate1	CMDIChildWnd1	CAqtoxClass	CAqtoxView
mTemplate2	CMDIChildWnd2	CRetoxClass	CRetoxView
mTemplate3	CMDIChildWnd3	CSpeciesClass	CSpeciesView
mTemplate4	CMDIChildWnd4	CStocasClass	CStocasView
mTemplate5	CMDIChildWnd5	CStoccbClass	CStoccbView
mTemplate6	CMDIChildWnd6	CStochrClass	CStochrView
mTemplate7	CMDIChildWnd7	CStoremClass	CStoremView
mTemplate8	CMDIChildWnd8	CStothaClass	CStothaView

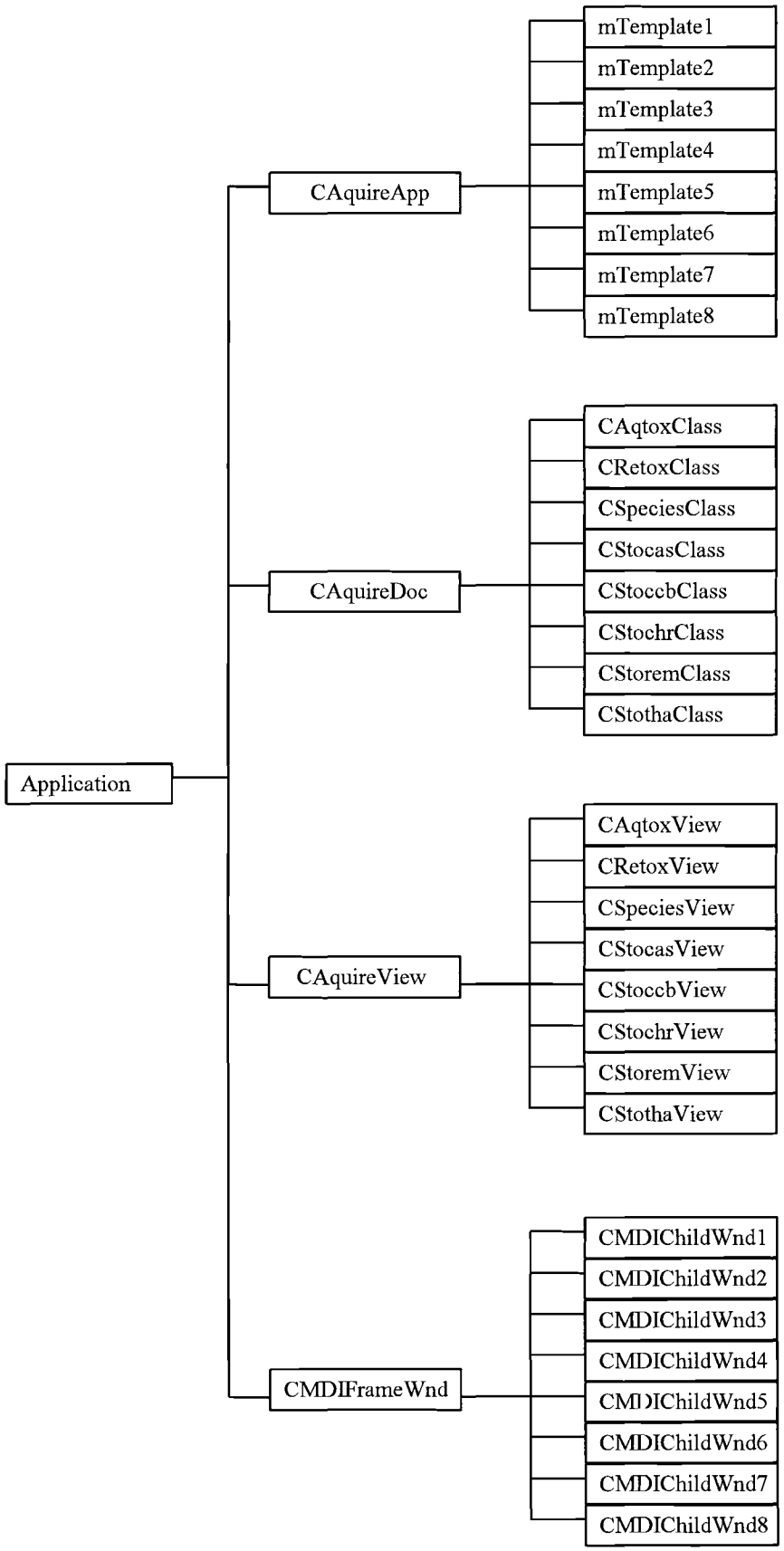


Figure 3 The architecture of class objects in the OOATDBMS

System Testing for OOATDBMS

The system testing is one of the five procedures defined in software engineering life cycle, which intends to check the verification and validation of the system. Verification focuses on checking whether the result agrees with the system specification, while validation focuses on checking the system behavior match the requirements [Jacobson 1992]. Generally, there are many types of tests, one should use a combination of them to perform the test for his/her system. Three types of the most important tests have been used to test the OOATDBMS. There are: (1) Unit testing; (2) Integration testing, and (3) User testing. Each type of test focuses on the different parts of the system functions.

Unit Testing

Unit testing is a most primitive testing performed after a program (or called a routine) is developed. This test focuses on the verification of the program. It tries to test and find out the problems such as: whether the program completes all the required functions, or if there is an exception handling function for data overflow, etc.

Integration Testing

Integration testing is a test performed after all the programs in the system are completed. This test is focuses on co-operation of different units that have been developed. It tries to test and find out the problems such as: whether program A outputs the correct data to program B, or if program A calls a sub-routine of program B using a correct argument, etc.

User Testing

User testing is a test performed after the system pasts the integration testing. The users, other than the programmer conduct the test. This test is focuses on the validation of the system. It tries to test and find out whether the system performs all the expected requirements, or if there is some modification the user expects to change, etc.

The OOATDBMS is developed and tested using PC 486/ DX 33 with 8 megabytes memory and 540 MB hard disk space. The system pasts the three types of the system test and the performance is satisfied by most professional or non-professional users.

CHAPTER VI

SUMMARY AND CONCLUSIONS

The concepts of object-oriented model and programming languages have become popular since last decade. The object-orientation concepts have evolved in three different disciplines: first in programming languages, then in artificial intelligence, and then in database. The basic concepts of object-oriented programming languages are objects, classes, abstract data type, data encapsulation, inheritance and polymorphism. Object-oriented analysis (OOA) is an analysis model developed to describe the functionality of the system. It integrates the data and the associative functions as objects rather than separates the data from the associative functions. Object-oriented design (OOD) is a design model used to design and implement the system analysis done by OOA or other analysis methods. Although no standard OOA and OOD methodologies is defined currently, the OOA/ Coad-Yourdon and OOD / Booch are one of the well-known OOA and OOD methodologies most in use.

One of the major applications of an object-oriented model is to combine the object-orientation concepts of programming and data model with database systems. This type of database system is called Object-Oriented Database Systems (OODBS). There are five major advantages of a OODBS: (1) data structures are flexible; (2) there are many facilities for describing data; (3) data values can be inherited; (4) complex models can be built; and (5) source codes are reusable.

The Object-Oriented Aquatic Toxicity Database Management System (OOATDBMS) is the object-oriented database management system designed for the original AQUIRE relation database. It is analyzed and designed by the object-oriented analysis / Coad-Yourdon and the object-oriented design / Booch methodologies. It consists of two sub-systems, one is a database and retrieval sub-system, another is a user interface sub-system. The database and retrieval sub-system retrieves the data from

the original AQUIRE database and maintains them as an object-oriented database. The user interface sub-system is a message-driven windows programs. It provides users with graphical user interface objects, such as windows, menus, and push buttons, etc. The user interface sub-system co-operates with database and retrieval sub-system to take users' input and display the information on the screen. The OOATDBMS is a complete and efficient database management system for the AQUIRE database because it can retrieve all the information existing in the database. Moreover, since the OOATDBMS is an object-oriented modular system with user-friendly interface implemented under Microsoft Visual C++ development environment, the system is very easy to use and maintain for future modification.

There are some restrictions apply to the OOATDBMS. The system retrieves the information by one search key at a time and the string should be entered in full if it is used as a search key. The suggestion for future system modifications is to implement multiple-key search, which will make the system more powerful and efficient.

BIBLIOGRAPHY

Alagic, Suad. (1989). Object-oriented Database Programming. Springer-Verlag.

Alvey, P.L., Preston, N.J. & Greaves, M.F. (1987). High Performance for Expert Systems: I Escaping from the Demonstrator Class. Medical Informatics, 12, 85-95.

Alvey, P.L., Preston, N.J. & Greaves, M.F. (1987). High Performance for Expert Systems: II Escaping from the Demonstrator Class. Medical Informatics, 12, pp. 97-114.

Andrews, T. C. (1987). Combining Language and Database Advances in an Object-Oriented Development Environment. in Proc. of 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications, Oct. 430-440.

Berman, Jay I., Moore, Harry H. 4th. & Wright, Jon R.(1994). Classic and PROSE Stories: Enabling Technologies for Knowledge-based Systems. AT & T Technical Journal, 73(1), 69-80.

Bertino, Elisa, & Martino, Lorenzo. (1991). Object-Oriented Database Management System:Concepts and Issues. Computer, 24(4), 33 - 47.

Bertino, E., & Kim, W. (1989). Indexubg Techniques for Queries in Nested Objects. IEEE Trans. in Knowledge and Data Engineering. Oct.

Bjoninerstedt, A., & Beitts, S. (1988). AVANCE: an Object Management System, in Proc. 3rd Intl. Conf. on Object-Oriented Programming Systems, Language, and Applications. San Diego, Calif., Sept.

de-Brito, J., Branco, F.A. & Ibanez, M. (1994). A Knowledge-based Concrete Bridge Inspection System. Concete International, 16(2), 59-63.

Carey, M., DeWitt, D, & Vanfenberg, S. (1988). A Data Model and Query Language for EXODUS. in Proc. of ACM SIGMOD Intl. Conf. on Management of Data. Chicago, IL, June, 413-423.

Christerson, M., Jonsson, P., & Ovvergaard, G. (1992). Object-Oriented Software Engineering. New York: Addison-Wesley Publishing Company.

Collins, Dave (David Hunter) (1995). Designing Object-Oriented User Interfaces. Redwood City, CA : Benjamin Cumming.

Date, C.J. (1990). An Introduction to Database Systems (Fifth Edition). Addison-Wesley Publishing Company.

Doherty, N. F., Kochhar, A. K., & Main, R. (1994). Knowledge-based Approaches to Fault Diagnosis: A Practical Evaluation of the Relative Merits of Deep and Shallow Knowledge. Proceedings of the Institution of Mechanical Engineers, Part B, Journal of Engineering Manufature, 208(B1), 39-45.

Durham, T. (1989). Having the Last Word on Information Retrieval. Computing, May.

Fishman, D., et al. (1987). IRIS: An Object-Oriented Database Management System. ACM Trans. on Office Information Systems, 5(1), 48-69.

Fichman, Robert G., & Kemerer, Chris F. (1992). Object-Oriented and Conventional Analysis and Design Methodologies. Computer 25(10), 22 - 40.

Gamm, W., & Herrmann, F. (1988). OCEX-Ein Expertensystem zur Uberpreufung von Kundenauftagen und zur Konfigurierung von Produktions-Seuerungsanforderungen. In K.-P. fahnrich(ed.): Expertersysteme in Panung und Produktion, Kongreband, KOMMTECH.

Gurewish, Nathan. (1994). Master Visual C++ 1.5. Indianapolis, IND. : SAMS.

Hornick, M., & S. Zdonik. (1987). A Shared, Segmented Memory System for an Object-Oriented Database. ACM Trans. on Office Information Systems, 5(1), 70-95.

Hurson, A. R., Pakzad, S. H., & Cheng, Jia-bing. (1993). Object-oriented Database Mangement Systems: Evolution and Performance Issues. Computer, 26(2), 48-58.

Johnsonvaugh, Richard. (1995). Object-Oriented Programming in C++. Englewood Cliffs, N. J. : Prentice Hall.

Keen, M. McBride (1986). Expert Systems in Clarifying Employment Law. Proceedings of KBS '86. Online Publication.

Kiwi, the team (1988). contact Mecchia, A. A System for Managing Data and Knowledge Bases. Proceedings of an ESPRIT technical meeting held in Brussels, Nov.

Kruglinski, David J. (1994). Inside Visual C++, 2nd ED. Redmond, WA: Microsoft Press.

Kulpaiboon, Kumpera. (1993). An Object-Oriented Database Retrieval System for Aquatic Toxicity Data Files. M.S. Thesis. Computer Science Department, Oklahoma State University.

Maier, D., et al. (1986). Development of an Object-Oriented DBMS. in Proc. 1st. Intl. Conf. on Object-Oriented Programming Systems, Language, and Applications. Porland, Oregon,Oct., 472-486.

Makinouchi, A., & H. Ishikawa (1988). The Model and Achitecture of the Object-Oriented Database System JASMIN. working paper, Fujitsu, Ltd. Kawasaki, Japan.

Maraschini, F. (1988/89). Easyfind - Methods and Tools for Intelligent Database Access. Inisys Italia report.

Mariani, J. A., (1993). Realizing Relational Style Operators and Views in the Oggetto Object-oriented Database System. Information and Software Technology, 35(4), 207-216.

Neilsen, Kjell. (1995). Software Development with C++ : Maximinzing Reuse with Object Technology. Boston: AP Professional.

Pappas, Chris H., & Murray, William H. III.(1994). The Visual C++ Handbook. Berkeley, CA: Osborne McGraw-Hill.

Rine, David C. (1992). Object-Oriented Computing. Computer, 25(10), 13-12.

Rowe, l., & Stonebraker, M. (1987). The POSTGRES Data Model. in Proc. Intl. Conf. on Very Large Database. Brighton, Englandm, Sept., 83-95.

de-SAM Lazaro, A., Zhang, Jie, & Kendallm, L. A. (1994). Knowledge-based Approach for Improvement of CNC Part Programs. Journal of Manufaturing Systems, 13(1), 20-30.

Schildt, Herbert. (1990). C, The Complete Reference. Berkeley, CA: Osborne McGraw-Hill.

Sun, W., Meng, W., & Yu, C. (1992). Query Optimization in Distributed Object-Oriented Database Systems. The Computer Journal, 35(4), 98 - 107.

Velez, F.G. & Bernard, V. Darnis. (1989). The O2 Object Manager: an Overview. in Proc. 15th Intl. Conf. on Very Large Database, Amsterdam. the Netherlands, Aug.

Venkatasubramanian, Venkat, & Vaidhyanathan, Ramesh. (1994). A Knowledge-based Framework for Automating HAZOP Analysis. AIChE Journal, 40(4), 496-505.

Wagner, Peter. (1992). Dimensions of Object-Oriented Modeling. Computer, 25(10), 12-21.

Weinreb, D. et al. (1988). An Object-Oriented Database System to Support an Integrated Programming Environment. IEEE Database Engineering Bulletin, 11(2), 33-43.

Weiser, S., & Lochovsky, F. (1989). OZ+: an Object-Oriented Database System. Object-Oriented Concepts, Applications, and Databases, (ed. W. Kim, and F. Lochovsky), Addision-Weslsy.

Wells, David L. (1992). Architecture of an Open Object-Oriented Database Management System. Computer, 25(10), 74 -82.

APPENDIXES

APPENDIX A

DATA REPRESENTATION

VAX FORTRAN Data Representation

The AQUIRE database consists of nine data files. Eight of them are stored in VAX FORTRAN binary format and one is stored in ASCII format. All of the fields in the data files are represented by one of the following data types: CHARACTER, binary INTEGER * 2, binary INTEGER * 4 and REAL * 4. VAX CHARACTER data are stored in ASCII format. VAX binary INTEGER numbers are stored in two's complement representation with the bytes stored in increasing order of significance, that is, the least significant byte is first. The sign bit of the INTEGER number occupies the most significant bit with zero for positive numbers and one for negative numbers.

INTEGER * 2 is stored in two contiguous bytes starting on an arbitrary boundary and represents the integer value from -32,768 to 32,767. INTEGER * 4 is stored in four contiguous bytes starting on an arbitrary boundary and represents the integer value from -2,147,483,648 to 2,147,483,647.

REAL * 4, also called floating-point data type, is stored in four contiguous bytes starting on an arbitrary boundary. Bits are labeled from the right to left, 0 through 31. The data are sign-magnitude with the sign bit occupying the 15th bit. Bits 7 through 14 are an excess 128 binary exponent for the floating-point data which represents the exponent value from -127 to 127 by the corresponding binary number minus 128. The normalized 24-bit faction of the floating-point data is stored in the bits 0 through 6 and bits 16 through 31 without the representation of the most significant fraction [Kulpaiboon 1993].

Figure 4 shows the VAX FORTRAN data representation.

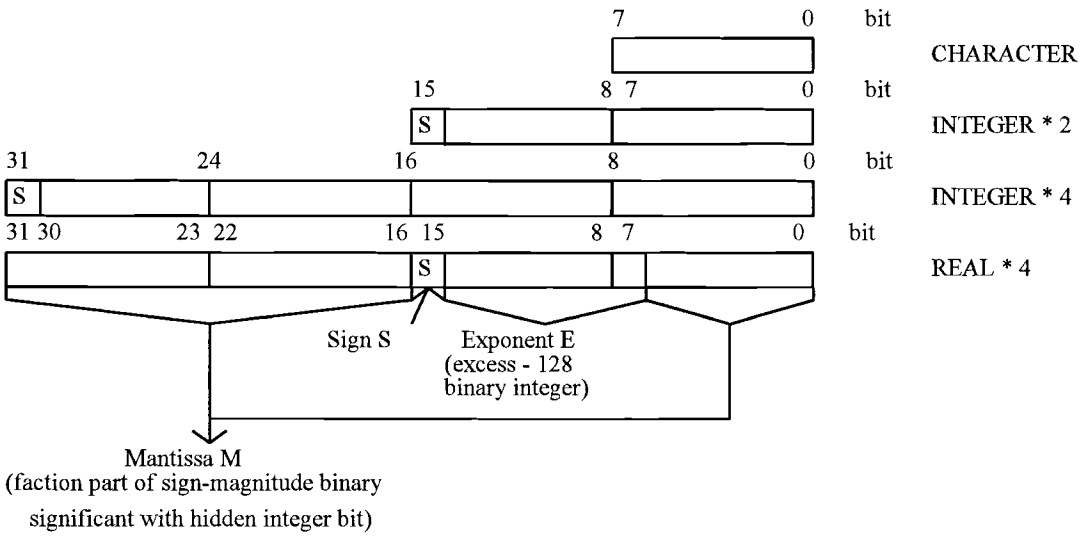


Figure 4 The VAX FORTRAN Data Representation

INTEL 80486 Data Representation

The major data types implemented by the INTEL 80486 processor are char, short integer, long integer, and float. Char is stored in ASCII format. Short and long integers are stored in contiguous two or four bytes in two's complement representation following little-ending addresses model. The 32-bit floating-point number of INTEL 80486 adopted the IEEE 754 standard. It comprises a 23-bit mantissa field M, an 8-bit exponent field E and a sign bit S. The base B is 2 and the sign bit occupies the left-most bit position. Mantissa is a faction which forms a sign-magnitude binary number with sign-bit. Since the exponent representation is the 8-bit excess-127, the actual exponent value is calculated as E - 127. The formula used to calculate the number is

$$N = (-1)^{S} 2^{E-127} (1.M). \qquad (0 < E < 255)$$

Figure 5 shows the INTEL 80486 data representation.

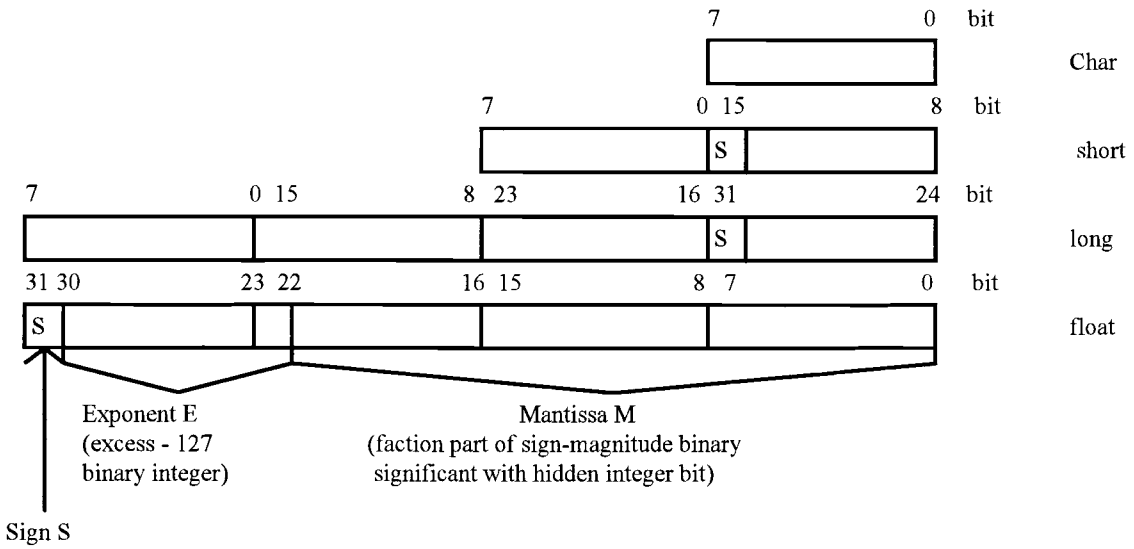


Figure 5 The INTEL 80486 Data Representation

DATA CONVERSION

Since the retrieval system is designed and implemented on the INTEL 80486 computer, the original data need to be converted from the VAX FORTRAN binary format to the INTEL 80486 binary format before they can be calculated and displayed. There are three types of conversion: (1) convert INTEGER * 2 to short integer; (2) convert INTEGER * 4 to long integer; and (3) convert REAL * 4 to float. No conversion is needed for the character data since they are stored in ASCII format on both machines.

The first two conversions can be processed by defining a struct data type in C language, the INTEL 80486 processor will return the correct short and long integer data values. However, the third type of conversion needs to be processed manually by the following steps:

Assumed VAX.value and INTEL.value represent the REAL * 4 and float data values on two machines,

Let

INTEL.sign = VAX.value > > 15;

INTEL.faction = (VAX.value & 0x7f) / 128 + 1;

INTEL.mantissa = ((VAX.value & 0xff80) > > 7) - 127;

for (int k = 0; k < INTEL.mantissa; k++) INTEL.exponent < = 1;

then,

INTEL.value = ((-1) ** INTEL.sign) * INTEL.faction * INTEL.exponent.

Note: 0x999 represents hexadecimal data.

APPENDIX B

OOATDBMS INSTALLATION GUIDES

The Object-Oriented Aquatic Toxicity Database Management System (OOATDBMS) requires the following minimum configurations:

- A PC with an 80386 or higher processor (80486 or higher recommended), running Microsoft Windows version 3.1 or Microsoft Windows NT version 3.5.
- A VGA monitor (SVGA monitor recommended).
- 8 megabytes of available memory.
- A hard disk with 85 MB free disk space in the working directory.

Installation Procedures

The source program and data files with their indexes of OOATDBMS are stored in 11 3.5" floppy HD diskettes. The data files are compressed using PKZIP compression utility. The installation procedures include five steps:

1. Create \OOATDBMS directory under hard drive [X] root directory;
2. Insert the program disk in the 3.5" floppy disk drive, such as a: drive, copy SETUP1.exe and PKUNZIP.EXE to [X]:\OOATDBMS;
3. Use the File Manger to run the SETUP1.EXE from the hard disk drive, such as [X]:\OOATDBMS\SETUP1.EXE

   or

   From the File menu in Program Manager, choose Run and type [X]:\OOATDBMS\SETUP1.EXE to install the execution program;
4. Insert the data disk #10 in the floppy disk drive;
5. Use the File Manger to run PKUNZIP.EXE to uncompress the AQUIRE.ZIP from the floppy disk drive, such as [X]:\OOATDBMS\PKUNZIP a:\AQUIRE.ZIP

or

From the File menu in Program Manager, choose Run and type [X]:\OOATDBMS\PKUNZIP a:\AQUIRE.ZIP, then follow the prompt to install the data files.

The installation programs SETUP1.EXE and AQUIRE.ZIP install one source program, eight indexes, and nine data files and nine description files into the directory [X]:\OOATDBMS. After the installation procedures is completed, there are totally twenty-seven files under the directory [X]:\OOATDBMS. Table 8 shows the names, sizes and their descriptions of the files that have been installed.

TABLE 8

LIST OF THE FILES AFTER INSTALLATION

File Names	File Sizes (Bytes)	File Descriptions
AQUIRE.EXE	1,761,072	System source program
AQZIREF.IDX	632,358	Index for AQTOX file
AQZISPEC.IDX	632,358	Index for AQTOX file
RETOXREF.IDX	59,706	Index for Citation file
LATIN.IDX	95,676	Index for species file
COMMON.IDX	95,744	Index for species file
MAJOR.IDX	16,896	Index for species file
MINOR.IDX	22,528	Index for species file
SPREF.IDX	59,706	Index for species file
1.TXT	4,640	Description file for 10.DAT
2.TXT	5,280	Description file for 11.DAT
3.TXT	2,880	Description file for 12.DAT
4.TXT	1,680	Description file for 13.DAT
5.TXT	2,560	Description file for 14.DAT
6.TXT	2,080	Description file for 15.DAT
7.TXT	1,760	Description file for 16.DAT
8.TXT	1,680	Description file for 17.DAT
9.TXT	1,920	Description file for 18.DAT
10.DAT	733,312	CAS number and Chemical name
11.DAT	16,863,040	AQTOX main information
12.DAT	5,413,888	Citation Information
13.DAT	191,420	Species Information
14.DAT	22,810,472	CAS number in each test
15.DAT	7,380,060	Concentration-Conf. int-BCF
16.DAT	5,269,500	Purity / Chemical characteristics
17.DAT	18,778,752	Remark Information
18.DAT	12,445,200	Temp-Hardness-Alk-D.O.-pH

APPENDIX C

User's Manual

The OOATDBMS is a MDI windows application, providing users with many common graphical user interface items, such as menus, edit boxes, push buttons, etc. Users can run the system as any other windows application by running the AQUIRE.EXE file either under File menu of Program Manager , or under File menu of File Manager.

The system consists of one menu screen and eight different view windows. The menu screen contains five menu items, i.e. File, Retrieve, View, Window, and Help. Each menu item contains several sub-menu items and the functions of these sub-menu items are as the same as any other standard window menu items except the sub-menu items under Retrieve menu. The detail information for those different sub-menu items under Retrieve menu is described as follows.

Item Name: AQUIRE

Function: Create and display the AQUIRE main information screen if it does not exist, otherwise, activate the screen and display it on the front window.

Item Name: Citation

Function: Create and display the Citation information screen if it does not exist, otherwise, activate the screen and display it on the front window.

Item Name: Species

Function: Create and display the Species information screen if it does not exist, otherwise, activate the screen and display it on the front window.

Item Name: CAS

Function: Create and display the Chemical Abstract Service information screen if it does not exist, otherwise, activate the screen and display it on the front window.

Item Name: Concentration-Conf.-BCF

Function: Create and display the Concentration-conf-BCF information screen if it does not exist, otherwise, activate the screen and display it on the front window.

Item Name: Purity/Chem Characteristic

Function: Create and display the Purity / Chemical Characteristic information screen if it does not exist, otherwise, activate the screen and display it on the front window.

Item Name: Temp-Hardness-Alk.-D.O.-pH

Function: Create and display the Temp-Hardness-Alk.D.O.-pH information screen if it does not exist, otherwise, activate the screen and display it on the front window.

Item Name: Remark

Function: Create and display the Remark information screen if it does not exist, otherwise, activate the screen and display it on the front window.

The system first displays the AQUIRE main information window with default information. Users can enter a search key in one of eight edit boxes which can accept users' input, then click the Search button, or press ENTER. If users click other push buttons, such as the Citation button, the related citation information window will be activated instead of current window. The information will be displayed if the record is found, otherwise, a message will be displayed.

There are two methods that users can use to retrieve other information. One is selecting the respective sub-menu item from the Retrieve menu under the main

windows; another method is clicking the respective push buttons on the current view window.

When one view window is activated, users can enter search key to search the information and retrieve other information by clicking one of the corresponding push buttons to activate other view window. Finally, each view window can be closed by two methods, i.e. either to click the Close button on the current window or to click the windows close button.

In sum, the OOATDBMS is a very user-friendly and efficient database management system for AQUIRE database, no matter for professional or non-professional users.

APPENDIX D

SAMPLE OUTPUT

The Object-Oriented Aquatic Toxicity Database Management System (OOATDBMS) is a windows application system. It can take the user's input, retrieve the information and display it on the different screen windows. The following screen windows are the sample output screens displaying the information for the record # 4 in the data file 11.dat. Figure 6 shows the AQUIRE main information. Figure 7 to Figure 13 show the associative information respectively.

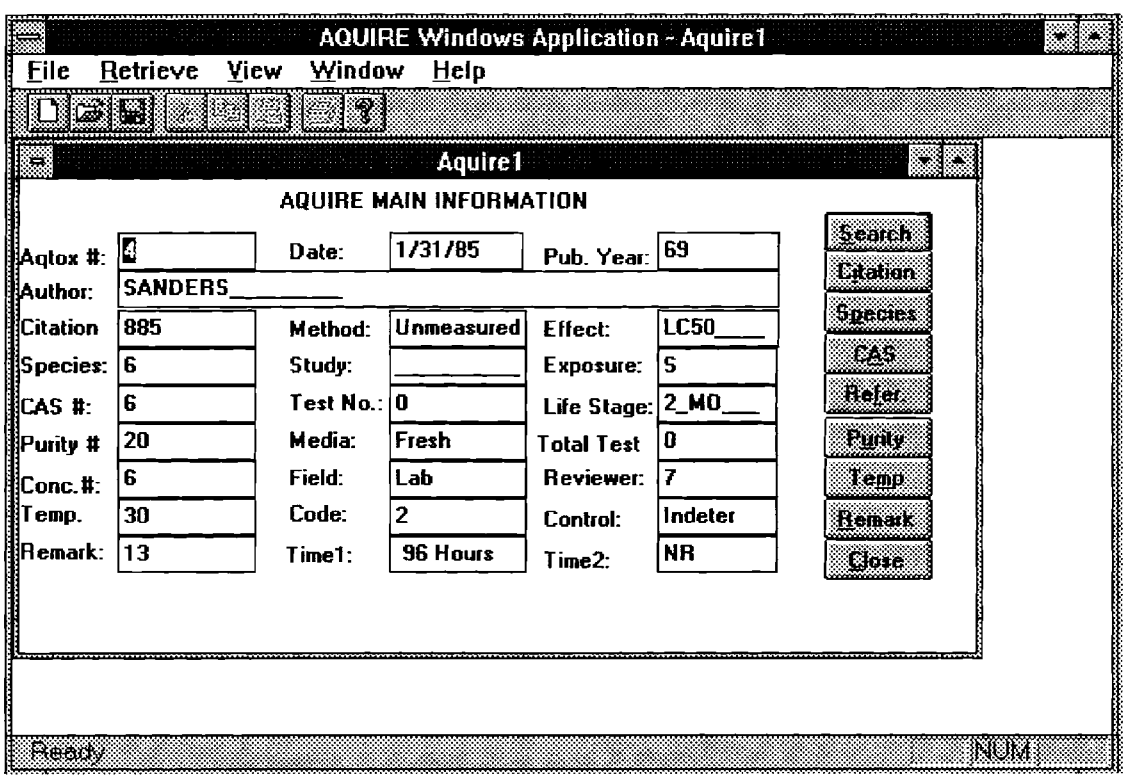


Figure 6 AQUIRE Main Information

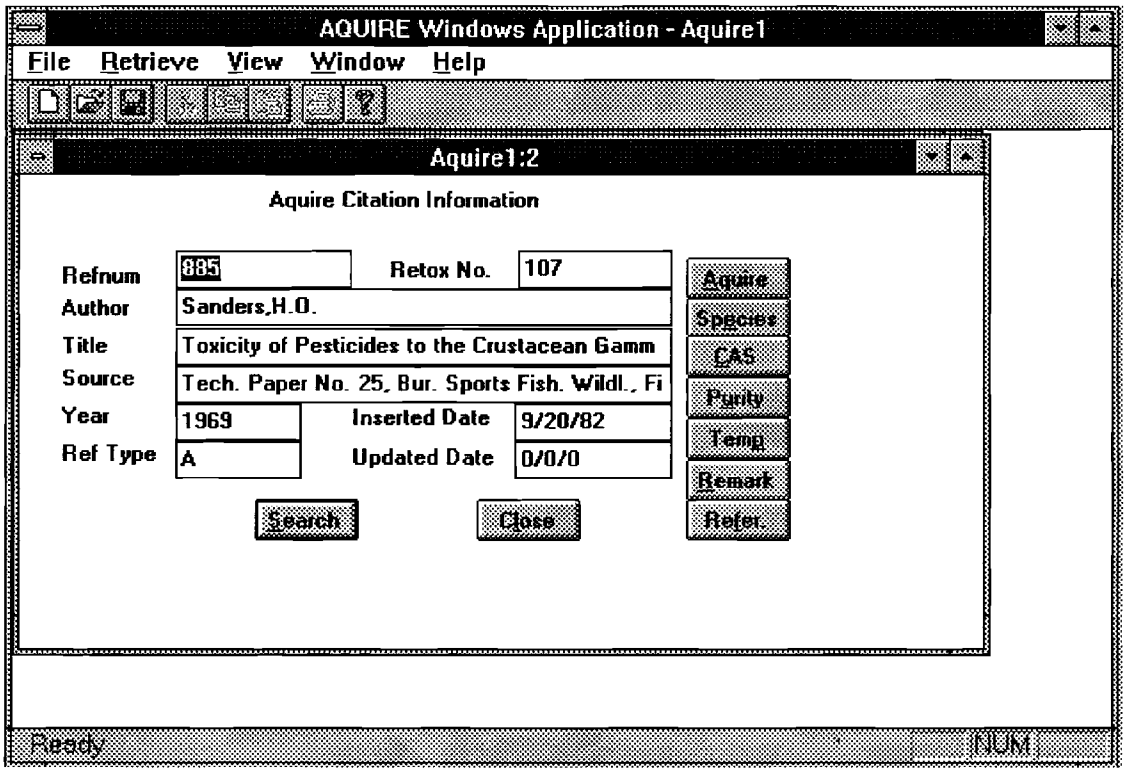


Figure 7 AQUIRE Citation Information

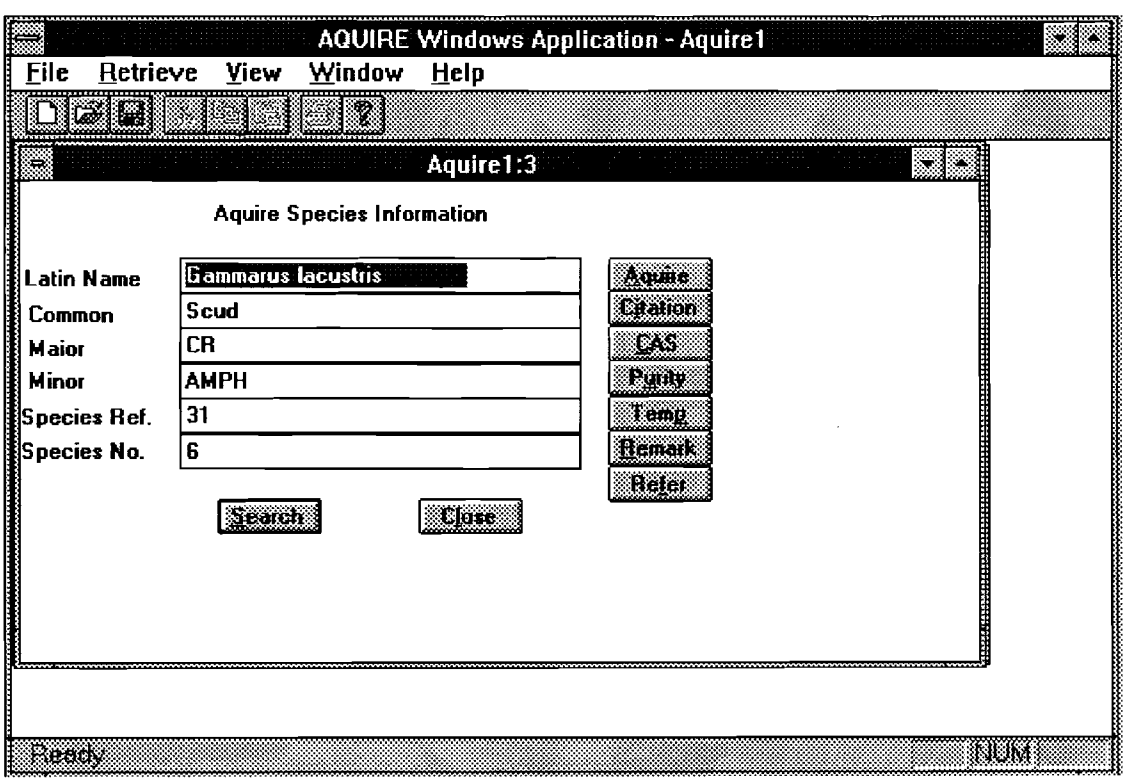


Figure 8 AQUIRE Species Information

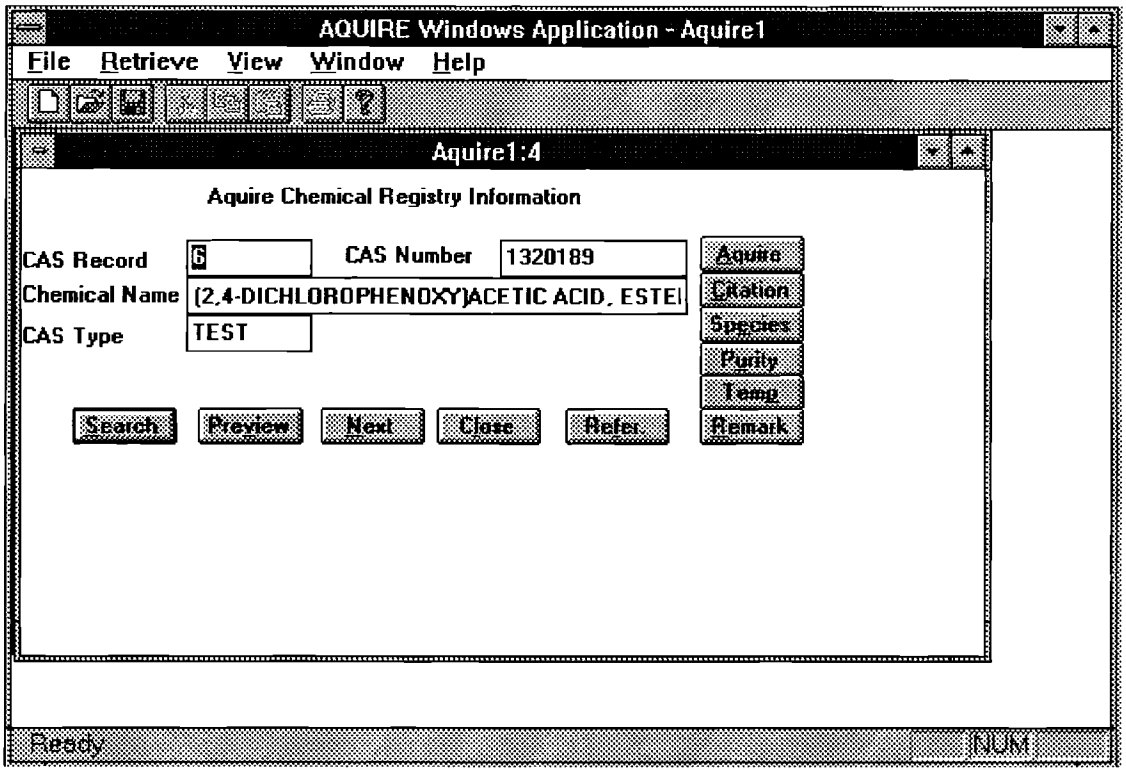


Figure 9 AQUIRE Chemical Registry Information

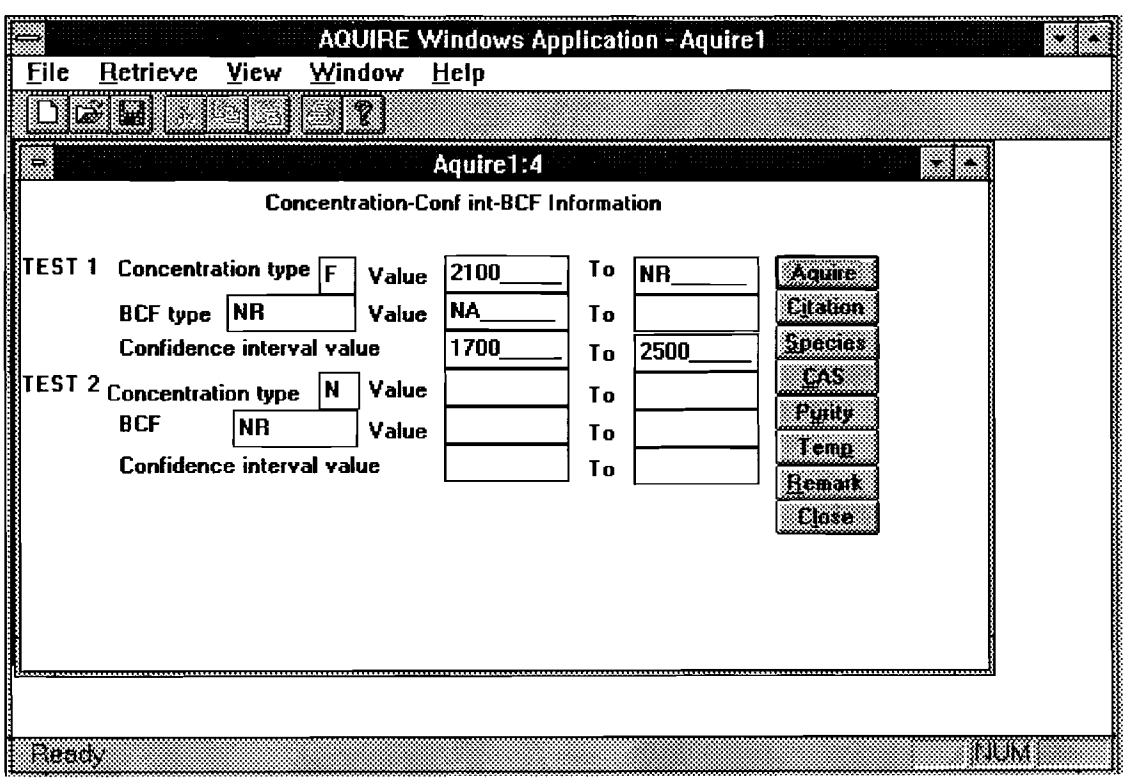


Figure 10 Concentration-Conf int-BCF Information

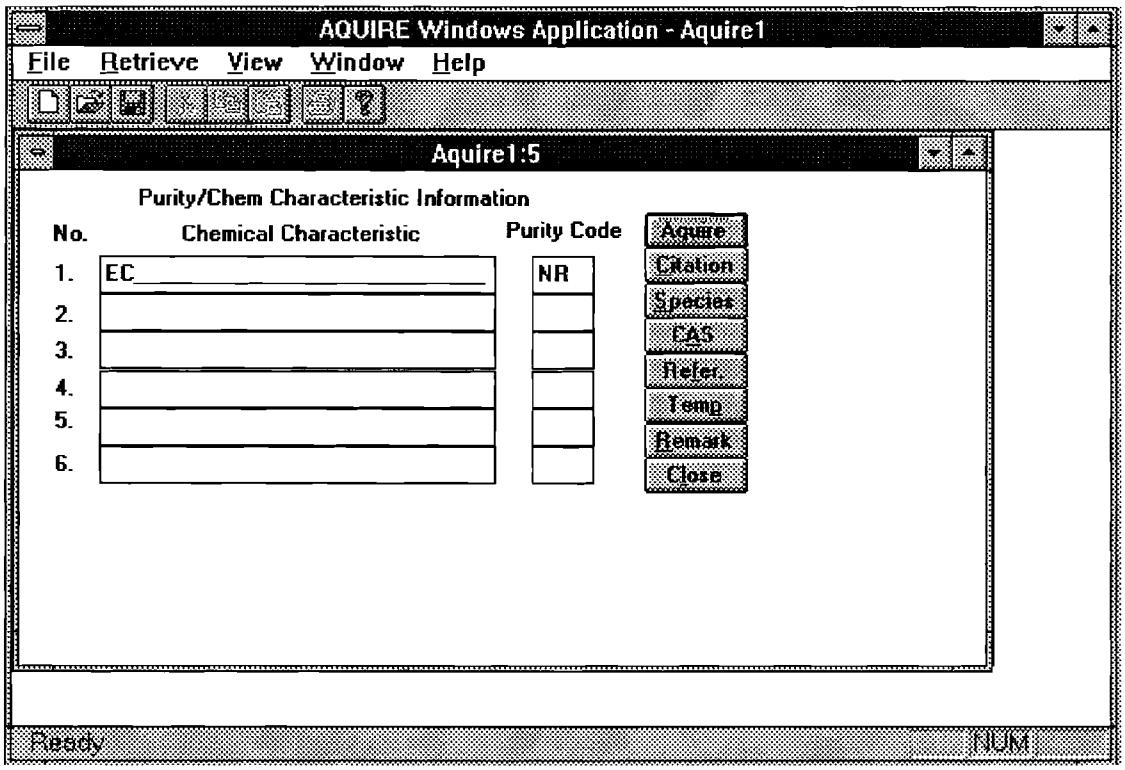


Figure 11 Purity / Chemical Characteristic Information

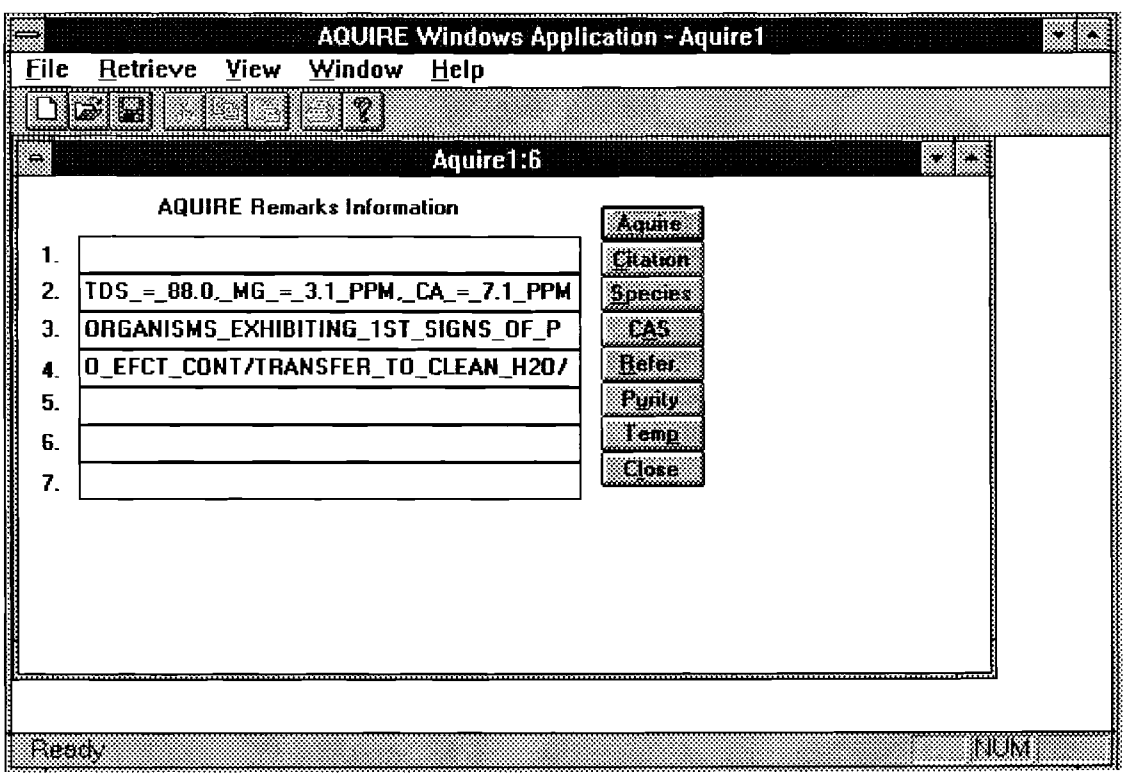


Figure 12 AQUIRE Remarks Information

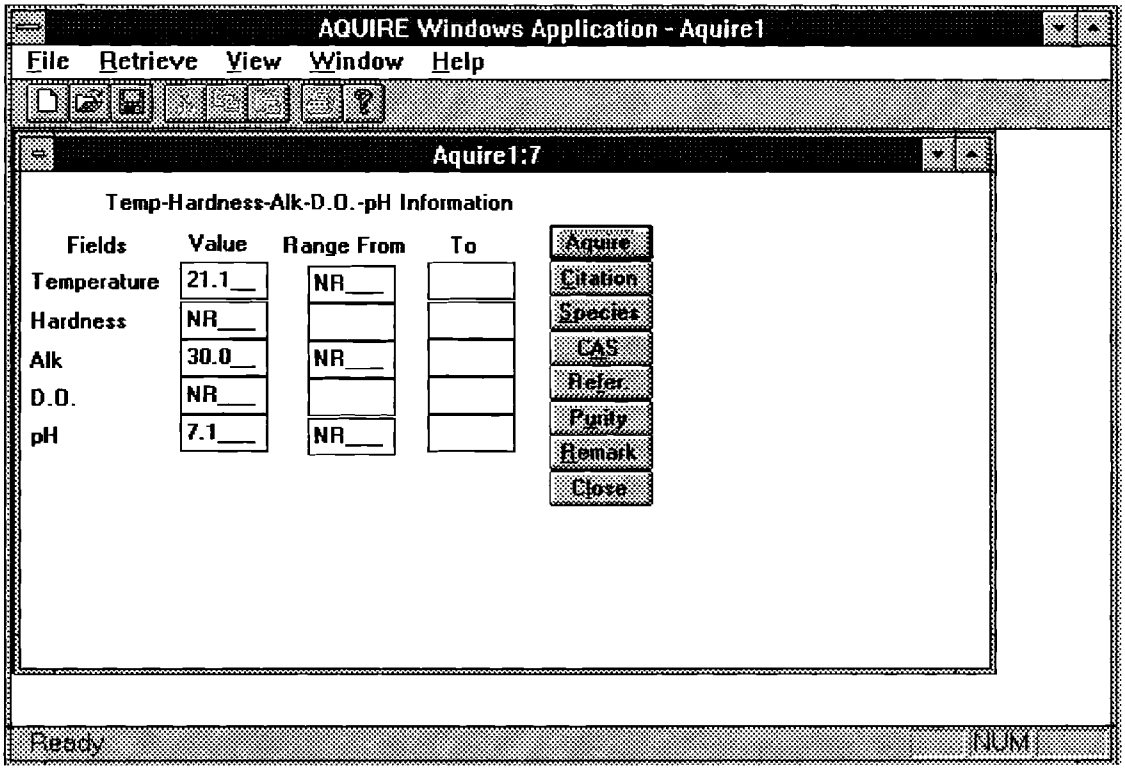


Figure 13 Temp-Hardness-Alk-D.O.-pH Information

APPENDIX E

A PROCEDURE FOR PRINTING RESULTS

In OOATDBMS, users can print the results displayed on the screen by following the procedure described below:

1. Retrieve information and display the information on the screen;
2. Adjust the size of the windows to make all the variables visible;
3. Press Print Screen key from the keyboard;
4. Open Paintbrush application from the Windows Assessories group;
5. Maximum the size of Paintbrush application window;
6. Select Paste from the Edit menu on Paintbrush application window;
7. Select Print from File menu on Paintbrush application window to print the window.

VITA

Hua Liou

Candidate for the Degree of

Master of Science

Thesis: DESIGN AND IMPLEMENTATION OF AN OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEM FOR AQUIRE DATABASE

Major Field: Computer Science

Biographical:

Personal Data: Born in Guilin, P.R.China, on September 18, 1968, the daughter of Jinkun Liu and Meiji Fan.

Education: Graduated from Zhongshan University, Guangzhou, P.R. China in July, 1990; received Bachelor of Science degree in Computer Science. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December 1995.

Professional Experience: Programmer/Analyst, American CCD-Online System (China), Inc., 1990 to 1992; System Administrator, Oklahoma State University Wellness Center, 1993 - 1994; Graduate Research Assistant, Oklahoma State University, Department of Agriculture Economics, 1994 - present.