BUILDING FAST AND EFFICIENT DATABASE APPLICATIONS FOR THE WEB

By

XINXUE YUAN

Bachelor of Civil Engineering

Beijing Polytechnic University

Beijing, 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, 2000

BUILDING FAST AND EFFICIENT DATABASE

APPLICATIONS FOR THE WEB

Thesis Approved:

Jane Terry Nutter
Thesis Advisor

J Chandler

[illegible]

[illegible]
Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my thesis advisor, Dr. J. Terry Nutter, for her intelligent supervision, constructive guidance, constant inspiration, and valuable time she has given me throughout this study. My sincere gratitude extends to Dr. John P. Chandler and Dr. George E. Hedrick for all the help and support they have given to me; their guidance, encouragement, assistance, and friendship are invaluable.

I would like to give my respectful thanks to my parents, Mrs. Shuying Zhao and Mr. Shixian Yuan, for all the love and support they have given me throughout my life. I would like to sincerely thank my two older brothers for their love, encouragement, support and confidence they have contributed to me.

Finally, I would like to thank all the faculty of the Department of Computer Science for their support during my two and half year study here.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

Database Management Systems (DBMSs) and database models evolved out of an increasing need for manipulations of huge collections of information that could be not be managed by simple file-processing techniques [12]. In their infancy, access to databases through DBMSs was a simple matter. The DBMS lived on the same machine as the database; that one machine was also used for all access, achieved for the most part directly through the DBMS, with some residual reliance on custom software for specific kinds of reports, etc [4]. As computers evolved from stand-alone systems to cooperating groups of smaller machines, and as databases became large enough to spread over multiple machines, the one-machine-one-program (DBMS) model gave way to a client-server model, in which the data resided on one system that might or might not host the DBMS, but access came through other systems. The client-server model was based on the concept of dedicated Local Area Networks (LANs), consisting mostly of compatible hardware platforms running dedicated client software on the machines used for access, and server software on those used for DBMSs management [26]. To make it easier for people, especially those without extensive training in programming and computer science, to work efficiently with databases, much effort has been given to develop server-side applications to form user-friendly interfaces with those DBMSs [25]. But today, even that model is becoming obsolete. Hardware ages so quickly that such dedicated LANs with all their machines would have to be replaced every couple of years to keep up; and client and server software changes just about as quickly. More seriously, in terms of computational environment models, today, the people who need access to a single database don't necessarily work together in the kind of proximity assumed by LANs. The invention of Internet technology overcame these limitations and yields a much more powerful Internet-based client-server model [38]. The heart of Internet technology is the World Wide Web (Web), that allows users on one computer to access information or resources through the worldwide network [1]. Hence, on the client side, this model makes it possible for users to access DBMSs from anywhere in the world, using any kind of platform that can access the Internet. Moreover, with the use of Web browsers, there is no need for each user to install dedicated client-side software which, most of the time, is licensed. On the server side, with the use of Internet-oriented Java technology, this model makes server software platform-independent, and provides more flexibility, extensibility, and portability [20].

It is not an easy task to implement the client-server model based on the Internet. Web database applications should execute quickly, provide as much information as possible, and have a very user-friendly interface. On the other side, the providers and designers should utilize an approach that makes it easy to design and to develop such applications, provides an efficient way to extend and reuse the available code, and brings more advanced functionality [11][21]. Currently, there are many approaches to developing Internet-based web database applications. The three dominant solutions are the Common Gateway Interface (CGI) approach, the Open Database Connectivity (ODBC) approach, and the Java Database Connectivity (JDBC) approach [20][36]. The CGI approach is a legacy approach that uses CGI as the standard protocol to communicate between a CGI application and a web server, and uses the CGI application to access back-end databases [26]. This approach has been used in many earlier Web database applications, but performance suffers. Each time a browser sends a request, a new CGI process must be loaded, and must initiate a new database connection. If a large number of users hit the site, repeatedly loading CGI applications for each request may dramatically degrade performance [7]. Thus, recent applications have investigated new approaches. The ODBC approach uses ODBC as an Application Programming Interface (API) for database interacting, and uses another API to communicate with the Web server [21}. The API use overcomes the efficiency limitation of the CGI approach by generating new threads for each new request. In addition, by inserting the driver manager between the application program and the DBMS driver, ODBC API lets each application access different data sources. ODBC approach is well established and is currently a very widely used solution [30]. But it has one severe drawback for developing Internet-based cross-platform applications. ODBC API uses a C interface, so that reusing the available code becomes an issue, and applications that will run on multiple platforms must recompile or even regenerate the code [15].

The JDBC approach is a pure-Java solution, and solves the platform-dependency problem with the ODBC approach. It uses JDBC, a pure-Java API introduced by Sun Microsystems, in place of ODBC API. JDBC API remains the basic design features of ODBC API [15]. It is very efficient and also enables one application to access databases through DBMSs. Moreover, the internet-oriented nature of Java makes pure-Java applications platform-independent. Java is also very easy to use while providing many advanced and powerful features [34]. Therefore, JDBC approach is the best choice for publishing databases over the Web.

The objectives of this thesis are to design and develop a general strategy for implementing the JDBC approach. This strategy utilizes the three-tier client-server model over the Internet and uses Java Foundation Classes (JFC) API for Graphical User Interface (GUI) design; and to design and build two integrated applications to illustrate this strategy. This task started from architectures and techniques that were developed to facilitate flexible and powerful access to databases using dedicated client machines with dedicated software over dedicated LANs, and extends those architecture and techniques to perform the same functions for any potential client machine anywhere in the world, assuming nothing except a modem, internet access, and a Java-enabled Web browser.

The thesis is organized as follows,

- Chapter 1: Introduction
- Chapter 2: Literature Review
- Chapter 3: JDBC Architecture and JDBC Working Models
- Chapter 4: Application Design, Implementation and Results
- Chapter 5: Conclusion and Future Work

CHAPTER II

LITERATURE REVIEW

2.1 Client-Server Technology

Currently, the dominant architecture for distributed computing is the client-server model, made possible by a combination of powerful computer hardware and reliable, fast, relatively low-cost communications technology [44]. Under the client-server model, each application consists of two parts: the Client that initiates communication and requests information, and the Server that responds and services information requests [35].

The client-server model has become the central idea of network computing, based on Local Area Networks (LANs) or the Internet. A client-server system generally contains three layers: user interface, business logic, and data sources [44]. The business logic is implemented by business objects, the data access logic and other services specific to data are implemented in a separated layer called the data service layer, and the user interface is implemented to make it easier for users to access the system [35].

The client-server technology has progressed from the traditional two-tier model to the three-tier model. Most business applications being developed today use the three-tier model [35]. In a traditional two-tier model, business logic and user interface layers are implemented on the client side, and the clients talk directly with the servers, in which data sources are located [44]. In a three-tier model, a middle tier is added between the clients and the data servers, and the business logic layer is moved from the clients into the middle tier. In practice, clients send requests to the middle tier through the user interface, then the business logic on the middle tier handles these commands, communicates with the data sources, and finally sends results back to the clients [35][42].

The three-tier model has many advantages over the two-tier model. First, the three-tier model is very powerful and flexible. The developer can change the underlying data services without changing the client application. Moreover, the developer can change the business logic that is implemented at the middle tier without affecting the client or the server. Second, this model is more scaleable, as the developer can add more power at the business logic or database server level without touching the client application. Third, the application server acts as a sort of "funnel" by maintaining a few database connections while servicing a much larger number of clients, thus greatly improving system performance. Finally, managing access to critical back-end data resources from the middle tier increases security [38][44].

The disadvantage of three-tier applications is that they are more difficult to build than two-tier applications. The obstacle is that, unlike the visually-oriented development of two-tier applications, an integrated development environment is difficult to use for the three-tier model and much more hand-coding is required [44].

2.2 World Wide Web

The World Wide Web, or WWW, came into being in the early 1990s. It is currently the most powerful and flexible Internet-based computer network that allows users on one computer to access information or resources stored on another through the world-wide network. The Web operates on a client-server model. Users run Web client browser software to contact a Web server and request information or resources. The computer on which the information or resources reside uses Web server software to locate and send requested material back to the Web browser [1][21].

The three primitive sets of rules specified by WWW designers for creating, publishing, and finding documents are URL (Uniform Resource Locator), HTTP (Hypertext Transfer Protocol), and HTML (Hypertext Makeup Language) [9]. A URL is the address of the resource to be retrieved from a Web server. It tells the browser what document to fetch, exactly where to find it on a specific server somewhere on the Internet, and what access protocol to use. HTTP is the protocol that governs transfer of information between the Web server and the Web browser. Information or documents on the Web reside in different Web "pages." Pages may contain text, graphics, multimedia, and any other kinds of Internet resources, and are connected to each other using hypertext links that allow a user to move from any page to any other page. HTML is the mark-up language that is used to create Web pages. HTML defines the display format of a Web page and enables hypertext links to be embedded in the page [1] [9].

The World Wide Web evolves at a very rapid pace, and many new technologies have been applied to it. Now users can not only visit Web pages, but also run programs that reside on remote computers rather than on the user's local computer, and two computers can interact with each other in a variety of complex ways [1]. One very important and useful application on the Web is Web database publication, which offers providers a powerful way to organize and maintain databases, and offers large numbers of remote users an efficient way to access databases [26][38].

2.3 Database and DBMS

"A database is a collection of related data" [10]. A database management system (DBMS) is a set of programs to handle all access to the database. With large amounts of data stored and protected by it, the DBMS provides a single user or multiple concurrent users facilities for defining, constructing and manipulating databases [10][12].

There are many ways to classify DBMSs. The most commonly used criterion is the data model. The data model is a set of data abstraction concepts that are used to describe data, data relationships, and consistency constraints [10][12]. Relational DBMSs (RDBMSs), based on the relational data model, are the predominant systems among present-day DBMSs. A relational database consists of a collection of tables that contain columns and rows, a set of rules that define relationships among the tables, and Structured Query Language (SQL), which provides mechanisms for manipulating data in the database and specifying security constraints [10][12]. Almost all widely used commercial DBMSs, including Oracle, Microsoft SQL Server, and Microsoft Access, are RDBMSs.

Other criteria used to classify DBMSs include the number of users, the number of sites, the cost of the DBMS, etc. For example, Oracle is a multiple-user RDBMS, while Microsoft Access is a desktop system supporting only one user at a time [10][12].

2.4 SQL

SQL, which stands for Structured Query Language, is an industry standard language to access and manipulate relational databases. The first SQL standard, SQL-86, was accepted as a standard by the American National Standards Institute (ANSI) and the

International Standards Organization (ISO) in 1986. The extended standard SQL-89 was published in 1989, and the current version of the ANSI/ISO standard is SQL-92 [12].

The components of SQL, based on the 1989 and 1992 standards, are [10][12]:

- Data Definition Language (DDL): the SQL DDL includes commands to create, alter, and drop the tables that form the structure of a relational database, and to specify data constraints – rules that table rows and columns must follow.
- Data Manipulation Language (DML): the SQL DML includes commands to retrieve and change data in the database. The four most common commands are INSERT, SELECT, UPDATE and DELETE.
- Data Control Language (DCL): the SQL DCL includes commands to control security and specify rights to access, alter, and manage all or selected parts of a database.

Because the SQL-92 standard is much larger than its predecessor SQL-89, and has a very wide scope specifying a very large number of features, it is broken into three levels: SQL-92 Entry Level, SQL-92 Intermediate Level and Full SQL-92 [40]. Today, nearly all major DBMSs support SQL-92 Entry Level. JDBC specification requires that a vendor's JDBC implementation must support at least SQL-92 Entry Level [49].

2.5 Web Database Publishing Technologies and Development Strategies

Though SQL is well suited to manipulate databases, it is poorly suited to develop applications, and programmers use it primarily as a means of communicating directly with databases. As more companies and users connect to the Internet, and with the growing number of applications, especially in the business world, that are tightly linked to databases, Web database publishing technology has begun to dominate traditional SQL programming [21][30].

A typical Web database publishing system consists of four key components: the application, the Web browser, the Web server and the database driver [21]. The application sends out requests, the Web browser provides the user interface, the database driver works as an information store, and the Web server connects the Web browser and the database driver.

There are several advantages to providing database applications through the Web [21][26][30][36][41]:

- Allowing a huge number of Internet-connected users to access the database.
- Offering a standardized user interface to all potential users.
- Offering platform-independent applications.
- Providing high transaction volumes (permitted by HTTP).
- Providing the ability to display multimedia data.
- Offering simple, low-cost application development.
- Providing an easy and powerful way to organize and maintain information for database providers.
- Providing an efficient and low-cost way to access the database for database users.

Currently, many technologies support developing Web database applications [20][41]. The author briefly describes three leading solutions here.

2.5.1 The CGI Approach

CGI (Common Gateway Interface) is a standard protocol for interfacing external applications with information servers, such as HTTP or Web servers. It sets up a standardized means of communicating between a CGI application and a Web server. A CGI application, also called a script, lets Web users access databases, and lets applications get information from forms that people fill out online. CGI applications can be written in virtually any programming language; the most commonly used are Perl (Practical Extraction and Reporting Language), C, and C++ [7][21].

When a user clicks on a "submit" button from a Web browser to post a form or send a request, HTTP uses the specific URL to locate a Web server. The server uses CGI to pass the request or form and its parameters to the CGI application. Then the CGI application executes the corresponding code to communicate with the back-end DBMS, and returns the results in HTML format to the server using CGI; the server sends the results back to the user's browser [21]. Figure 2-1 shows database access through the CGI approach.

In the early stage, the CGI approach was widely used for developing interactive Web applications. It works on both two-tier and three-tier client-server models, but in most cases it is very slow and cumbersome [26][38]. Every time a Web browser sends a request, the Web server triggers the CGI application, which must be loaded to execute the request, then terminates. Repeatedly loading large CGI applications for every request consumes time and resources, and adds overhead to the Web server, especially when many users request data at about the same time. Thus, the CGI approach provides a poor solution for complex applications [7][[20].

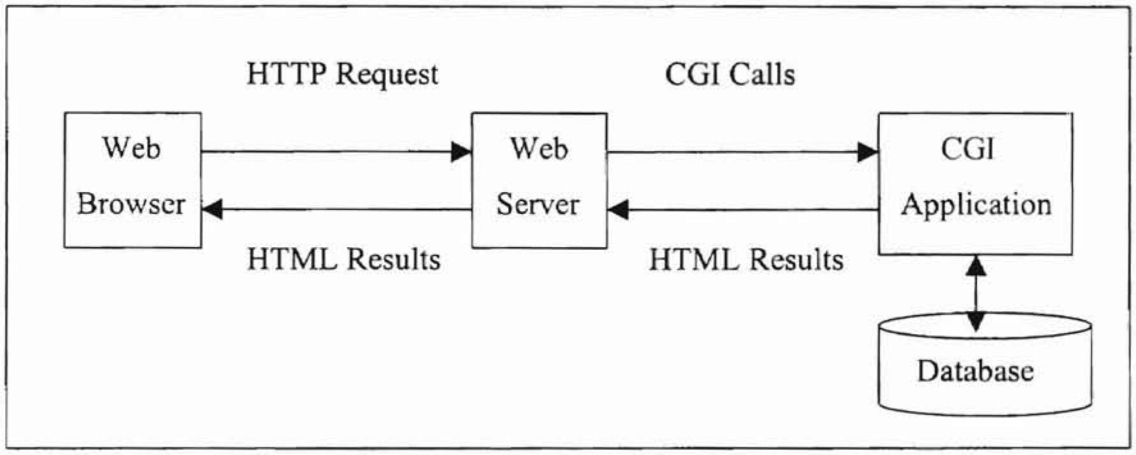


Figure 2-1 Database Access Through CGI Approach

2.5.2 The ODBC Approach

ODBC (Open Database Connectivity) was first created by Microsoft Corporation in 1992, as an API (Application Programming Interface) for accessing databases. The ODBC architecture has four major components or layers [30] [36]:

- Application: the user application calls ODBC functions to send SQL statements to the data source. It can be written in one of many programming languages, such as C, or C++.
- Driver Manager: this is a DLL (Dynamic Link Library) that provides access to

individual ODBC function calls. It may also load driver DLLs and check ODBC function calls.

- Driver: the driver processes ODBC function calls. It connects to a data source on command from the user application, sends SQL statements to the data source, and returns results back to the application. Drivers are often developed by DBMS vendors.
- Data Source: the data source consists of the DBMS and any data stores the user application defines.

The ODBC approach is well-established, and is currently a very widely used approach. Most applications using the ODBC approach today work on the three-tier model [20][30]. In a three-tier model over the Internet, the ODBC approach uses ODBC API for database interacting, and uses other APIs to communicate with the Web server [30]. The API use overcomes the efficiency limitation of CGI approach by generating new threads for each new request. In addition, by inserting the driver manager between the application program and the DBMS driver, ODBC API lets each application access different data sources. Figure 2-2 shows database access through the ODBC approach using an Internet-based three-tier client-server model.

Using ODBC approach raises serious concerns. ODBC API uses a C interface. This makes it very difficult to write a program that will run on multiple platforms, and thus makes it very difficult to reuse the available code in other applications running on different platforms without modification [20][30].

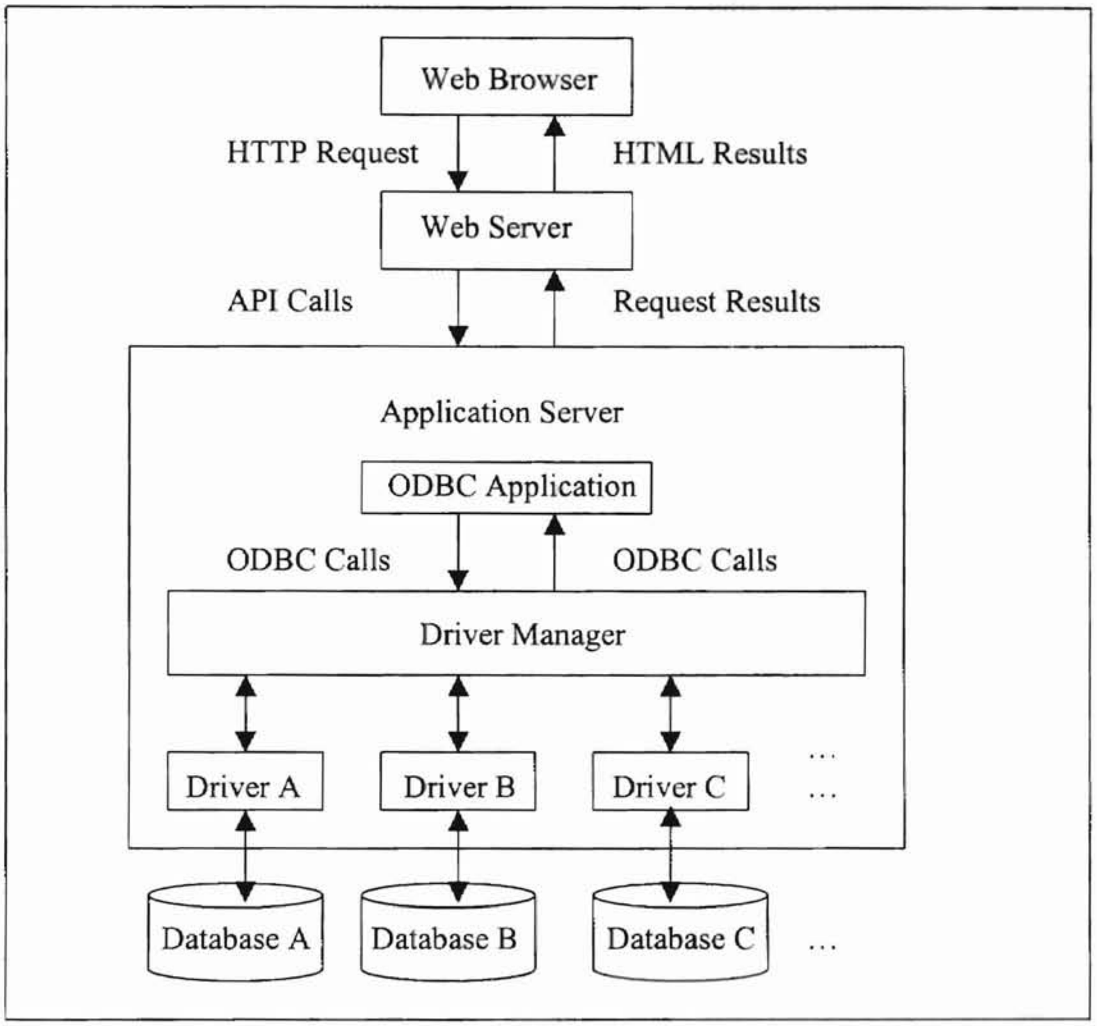


Figure 2-2 Database Access Through ODBC Approach

2.5.3 The JDBC Approach

JDBC (Java Database Connectivity), first developed by Sun Microsystems Inc. and its partner companies in March 1996, is a standard Java™ API for connecting to relational databases and executing SQL statements [30]. Since JDBC builds on ODBC, it retains the basic design features of ODBC. Like ODBC, JDBC architecture consists of four layers: JDBC application, JDBC driver manager, JDBC driver, and data source [39].

Today, most complex Web database publishing systems using the JDBC approach adopt the three-tier client-server models, that gives them more scalability, flexibility, and security [44]. Figure 2-3 shows database access through the JDBC approach in an Internet-based three-tier model, which is similar to the ODBC approach [20].

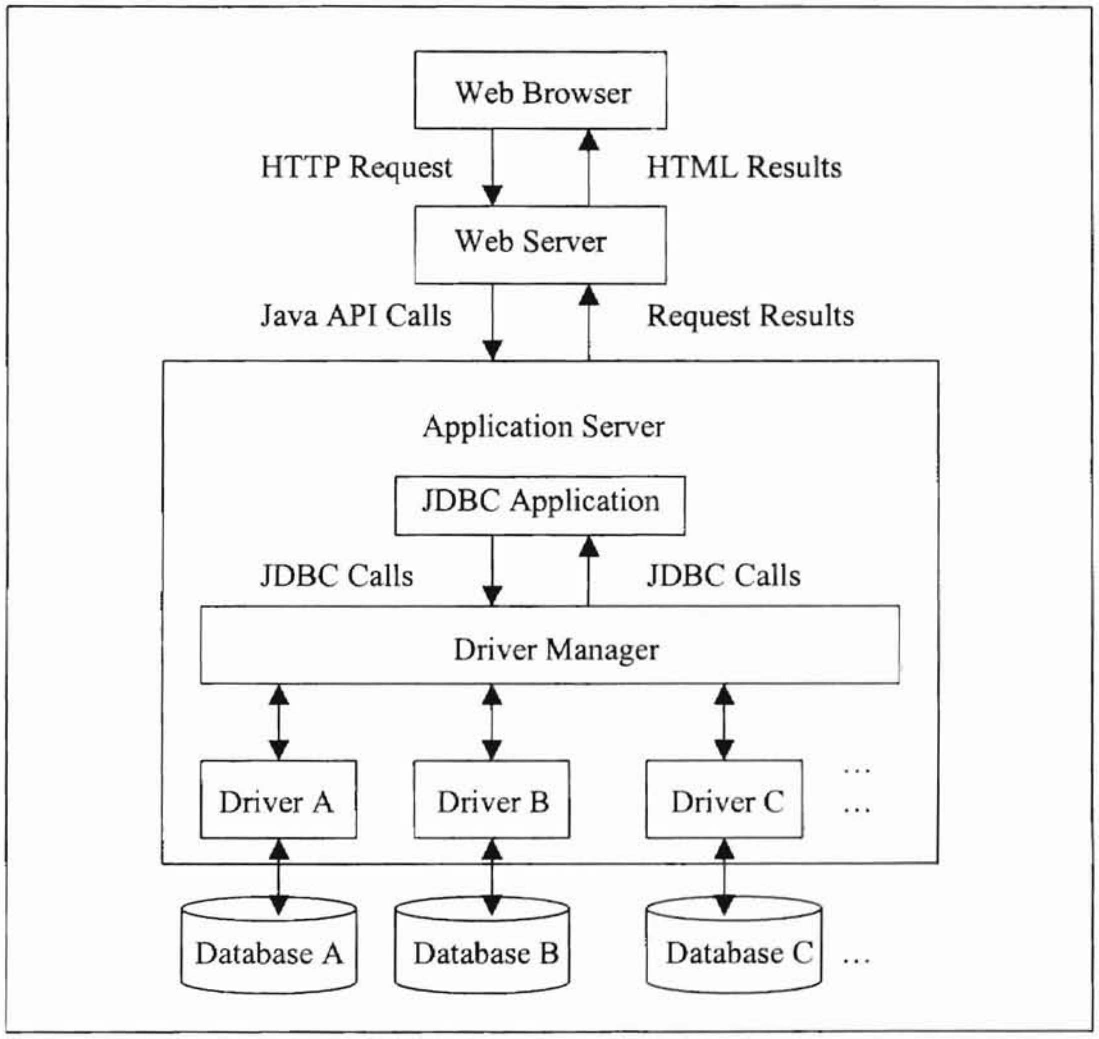


Figure 2-3 Database Access Through JDBC Approach

Unlike the ODBC approach, the JDBC approach uses pure-Java APIs. Based on a three-tier model over the Internet, it uses JDBC API to communicate with the back-end

DBMSs, and uses other Java APIs to communicate with the Web server [39]. Java is a powerful but simple object-oriented programming language. Java source code is compiled into an architecture-neutral byte-code, and at run-time, this byte-code can be interpreted by the Java Virtual Machine (JVM) on any Java-enabled operating system. Since virtually all modern web browsers are Java-enabled, Java is platform-independent [34]. This means that the same JDBC-enabled Java application can run on different platforms, and access all of the major RDBMSs, without even being recompiled [39].

Another advantage of the JDBC approach is that JDBC was designed to be very compact and simple. Hence the JDBC mechanisms are easy to understand and use, while allowing advanced features and capacities where required [39].

CHAPTER III

JDBC ARCHITECTURE AND WORKING MODELS

This chapter consists of two sections. Section one introduces the JDBC architecture, including JDBC components, JDBC driver types, SQL conformance, the JDBC API, scenarios of use, and JDBC security considerations. Section two describes two-tier and three-tier JDBC working models, and presents the dedicated strategy for implementing the JDBC approach.

3.1 JDBC Architecture

3.1.1 JDBC Components

Like ODBC, JDBC architecture consists of four components: the JDBC application, JDBC driver manager, JDBC driver and data source [39].

- JDBC Application: this is the user application which calls JDBC API to interact with the data source. It is coded in the Java language.
- Java Driver Manager: this is the backbone of the JDBC architecture. It keeps track of available drivers and handles establishing a connection between a data source and the appropriate JDBC driver. This management layer of JDBC lets an application communicate easily and efficiently with different data sources through different types of JDBC drivers.
- JDBC Driver: the JDBC driver processes JDBC function calls to a data source on command from the user application. It sends SQL statements to the data

source, and returns results to the application. Drivers are often developed by the DBMS vendors. Currently, there are four types of JDBC drivers.

- Data Source: the data source consists of any DBMS and any kind of data stores the user application defines.

3.1.2 JDBC Driver Types

JDBC drivers fit into one of four categories [30]:

- JDBC-ODBC Bridge Driver: this driver maps JDBC function calls to ODBC calls, and thus provides JDBC access via most ODBC drivers. This bridge is provided by Javasoft, and gives JDBC the capability to access almost all DBMSs, as ODBC drivers are widely available. This driver requires loading some binary code on each client machine.
- Native-API Partly-Java Driver: this kind of driver converts JDBC calls into calls on the client API for Oracle, DB2 or other major DBMSs. Like the previous bridge driver, this style of driver also requires loading binary code on each client machine.
- Net-Protocol All-Java Driver: this kind of driver translates JDBC calls into a DBMS-independent net protocol. This protocol is then translated to a DBMS protocol by a dedicated server that works as a middleware to connect its all-Java client to many different databases. This server and the specific protocol are developed by JDBC driver vendors.
- Native-Protocol All-Java Driver: this driver translates JDBC calls directly into the network protocol used by DBMSs. Since many of these protocols are

proprietary, this kind of driver is primarily developed by database vendors.

Because drivers in categories 3 and 4 are implemented in pure-Java, they can offer all the advantages of Java. Hence it appears likely that eventually they will be the preferred ways to access databases from JDBC [2][30].

3.1.3 SQL Conformance

Structured Query Language (SQL) is the industrial standard language for accessing DBMSs. However, there are two major areas of difficulty when developing JDBC applications. First, different DBMSs support significantly different versions of SQL. Second, although most DBMSs support some subset of one of the SQL standards, they may not conform to the most advanced SQL syntax or semantics as defined by the latest standard [4]. For example, Oracle supports stored procedures, but Microsoft Access does not.

To deal with the first problem, JDBC defines a set of generic SQL type identifiers to represent the most commonly used SQL data types [15][30]. By mapping JDBC types to database-specific SQL types, and vice versa, programmers can develop applications for transferring data between the application using Java types and a database using specific SQL types, without having to be concerned about the exact SQL type name used by the target database. Table 3.1 shows the mapping between some JDBC types and database-specific SQL types used in four major DBMSs [15].

JDBC Type	Oracle	IBM DB2	MS Access
Bit	N/A	N/A	N/A
Integer	Integer	Integer	Integer
Real	Real	N/A	Real
Float	Float	Float	Float
Double	Double Precision	Double Precision	Double
Numeric	Numeric	Numeric	N/A
Decimal	Decimal	Decimal	N/A
Char	Char	Char	Char
VARCHAR(n)	VARCHAR(n) $n <= 2000$	VARCHAR(n) $n <= 255$	VARCHAR(n) $n <= 255$
Binary	N/A	Char for Bit Data	Binary
Date	date	Date	Date
Time	Date	Time	Time

Table 3.1 Some JDBC Types Mapped to Database-specific SQL Types

JDBC provides three mechanisms to deal with the second problem [15][30]. First, JDBC allows any SQL query string to be passed to the DBMS driver. If the SQL functionality is available in that DBMS, correct results will be returned; otherwise, an error message will be sent back. Second, JDBC provides escape clauses. These clauses signal the driver that the code within them should be handled differently for different DBMSs. For complicated applications, JDBC also provides descriptive information about DBMSs, so that applications can be developed to meet the requirements and capabilities of different DBMSs, without risking receiving errors. In the JDBC specification, it is required that, for SQL conformance, a vendor's JDBC driver must support at least ANSI SQL-2 Entry Level, so that users can rely on a standard level of JDBC functionality.

3.1.4 JDBC API

The JDBC API is a standard SQL database access interface. It provides Java applications with a uniform interface to most databases, and provides an industrial standard base on which high level tools and interfaces can be built. JDBC API is now part of the Java Enterprise APIs and is included in the Java Development Kit (JDK). The current version of JDBC API is the JDBC 2.0 API, included in the JDK 1.2 release [49].

The JDBC 2.0 API consists of two components. The first component, termed the JDBC 2.0 Core API, is implemented in the java.sql package, and contains all the basic interfaces. The second component, termed the JDBC 2.0 Standard Extension, is implemented in the javax.sql package, and extends the functionality of the JDBC API from a client-side API to a server-side API [49]. This thesis restricts its attention to the JDBC 2.0 Core API.

The java.sql package contains all the JDBC interfaces and classes that are needed to implement the JDBC functionality for accessing databases (see Table 3.2) [49].

Functionality Type	Class
Driver	java.sql.Driver java.sql.DriverManager java.sql.PropertyInfo
Connection	java.sql.Connection
Statements	java.sql.Statement java.sql.PreparedStatement java.sql.CallableStatement
ResultSet	java.sql.ResultSet
Error/Warning	java.sql.SQLException java.sql.BatchUpdateException java.sql.Warning
New Built-in SQL Types	java.sql.Array java.sql.Blob java.sql.Clob java.sql.Struct
SQL Reference	java.sql.Ref
SQL-Java Type Mapping	java.sql.SQLData java.sql.SQLInput java.sql.SQLOutput
MetaData	java.sql.DatabaseMetaData java.sql.ResultSetMetaData
Date/Time	java.sql.Date java.sql.Time java.sql.TimeStamp
Miscellaneous	java.sql.Types java.sql.DataTruncation

Table 3.2 Classes in java.sql Package

3.1.5 Scenarios of Use

In the standard JDBC specification, JDBC designers present three common scenarios for JDBC use to develop database applications: Java applets, Java standalone applications and three-tier access [26][49].

A Java applet is a Java program that is included in web documents such as HTML pages. When the user uses a Java-enabled browser to view a document that contains an applet, the applet's code is downloaded over the net and then executed by the Java Virtual Machine (JVM) on the local system. By their nature, applets seem to provide a good solution for developing Internet/Intranet database applications. However, using applets raises several concerns. One is security. By default, applets are severely constrained in the operations they are allowed to perform. In particular, downloaded applets may not access local files or make network connections to other hosts. Another concern is performance. To be executed, applets must be downloaded to the client's local system. For low-bandwidth Internet connections, download time may severely degrade application performance. Hence, applets are well suited for presenting friendly user interfaces, but are poorly suited to host business logic, such as JDBC functionality.

JDBC API can also be used in Java standalone applications installed and executed on client machines. In this case, the applications are trusted, and allowed to access local files, open network connections, etc., just like normal Java applications. The most common use of Java applications is within Intranets, but they can also access databases through the Internet.

One big limitation with the two scenarios above is that they are based on the classic two-tier client-server model, in which the clients communicate directly with the back-end DBMSs, and thus applications are platform-dependent and vendor-specific, and are difficult to deploy, maintain, and upgrade. Hence, JDBC designers recommend the third scenario: three-tier access, that overcomes this limitation [49]. In this case, only user interfaces are presented on the client side, business logic and functionality of the applications is located and operated on the server-side. Three-tier access is based on the three-tier client-server model, and is best suited for JDBC-enabled Web database applications. The two-tier and three-tier client-server models are discussed in section 3.2.

3.1.6 JDBC Security Considerations

JDBC-enabled Web database applications are pure-Java applications. They should follow the standard Java security model that has been evolving since the inception of Java technology. The original model provided by JDK 1.0 is the Sandbox model [16]. According to this model, local Java applications have full access to vital system resources, while downloaded Java applets can only access limited resources. A Security Manager class determines which resource access is allowed.

JDK 1.1 introduced the concept of signed applets [16]. Like local applications, correctly digitally-signed applets are trusted by end systems, and have full access to local system resources.

In the Java 2 platform, the security model changed dramatically. By introducing the Access Control List (ACL) mechanism, the new model eliminated the built-in concept that all local applications, servlets, and signed applets are trusted. Instead, all Java applications are subject to the same security control [16][31]. When code is loaded, it is assigned permissions based on the external configurable security policy file. Each permission grants a specific level of access to a particular system resource; code can only access the resources that are granted to it. By using permissions and an access control policy file, this ACL architecture allows fine-grained, highly configurable and extensible access control. In addition, the ACL model is fully backward-compatible, so that the Security Manager mechanism is still supported.

3.2 JDBC Working Model

JDBC can be used in both two-tier and three-tier models. In the traditional two-tier model, a front-end client runs the Java applet or Java application, which employs JDBC API and talks directly to the back-end database server. The client's commands or requests are delivered to the server, and the results or requested information are sent back to the client. The data source may be located on another machine to which the client is connected via a network. Figure 3.1 illustrates the two-tier architecture for database access using JDBC.

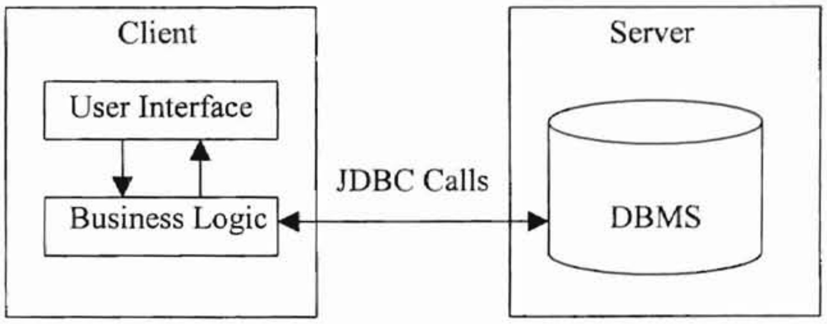


Figure 3.1 Two-tier JDBC Model

The two-tier application is easy to build, but has a number of limitations. First, the client must be configured to include a driver for each type of database being accessed.

This restricts flexibility and choice of DBMS for application, and can be costly, since such drivers usually carry a license fee. Second, the number of active clients is limited. This limitation results from the server maintaining a connection via "keep-alive" messages with each client, even when no work is being done. The number of open connections to the database server affects system performance. Finally, the two-tier architecture provides limited flexibility in moving program functionality from one server to another without manually regenerating procedural code on each client. The two-tier application is platform- and vendor-specific, and is difficult to deploy, maintain and upgrade. Hence, the JDBC approach based on the two-tier model is a poor solution.

The three-tier model overcomes the limitations with the two-tier model. In this model, the client no longer communicates directly with the DBMS. Instead, a middle tier is added between the client and the data server. The middle tier functions as an application server, moving the business logic from the client into the middle tier. In practice, the client sends requests to the application server through the user interface; the business logic handles these commands and then sends them as JDBC calls to the data source. The data source processes the commands and sends the results back to the application server, which then sends them to the client (See Figure 3.2) [16].

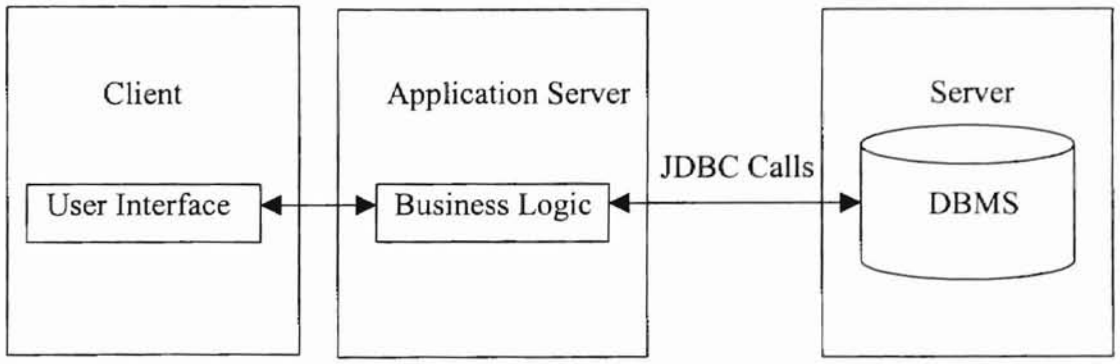


Figure 3.2 Three-tier JDBC Model

The three-tier model has many advantages over the two-tier model. First, the three-tier model is very powerful and flexible. The developer can change the underlying database services without changing the client application. Moreover, the developer can change the business logic that is implemented at the application server without affecting the client or database server. Second, this model is more scaleable, as the developer can add more power at the business logic or database server level without touching the client application. Third, the application server acts as a sort of "funnel" by maintaining a few database connections while servicing a much larger number of clients, thus greatly improving system performance. Finally, managing access to critical back-end data resources from the middle tier increases security. For instance, clients outside of a firewall can access an application server without compromising the security of the data sources the application uses.

The disadvantage of three-tier applications is that they are more difficult to build than two-tier applications. The obstacle is that, unlike the visually-oriented development of two-tier applications, an integrated development environment is difficult to use for the three-tier model and much more hand-coding is required.

Today, many enterprise database applications have used the Internet-based three-tier model, allowing users to access DBMSs from literally anywhere in the world. In this case, Graphical User Interface (GUI) design has become increasing central. Web database applications should provide very user-friendly interfaces to make it easier for people, especially those without extensive training in programming and computer science, to work effectively with the back-end DBMSs. In this thesis, the author designs and develops a general strategy for implementing the JDBC approach, as shown in Figure

3.3. This strategy utilizes the three-tier client-server model and uses Java Foundation Classes (JFC) API for Graphical User Interface (GUI) design. The author designs and builds two applications to illustrate this strategy, as discussed in next chapter.

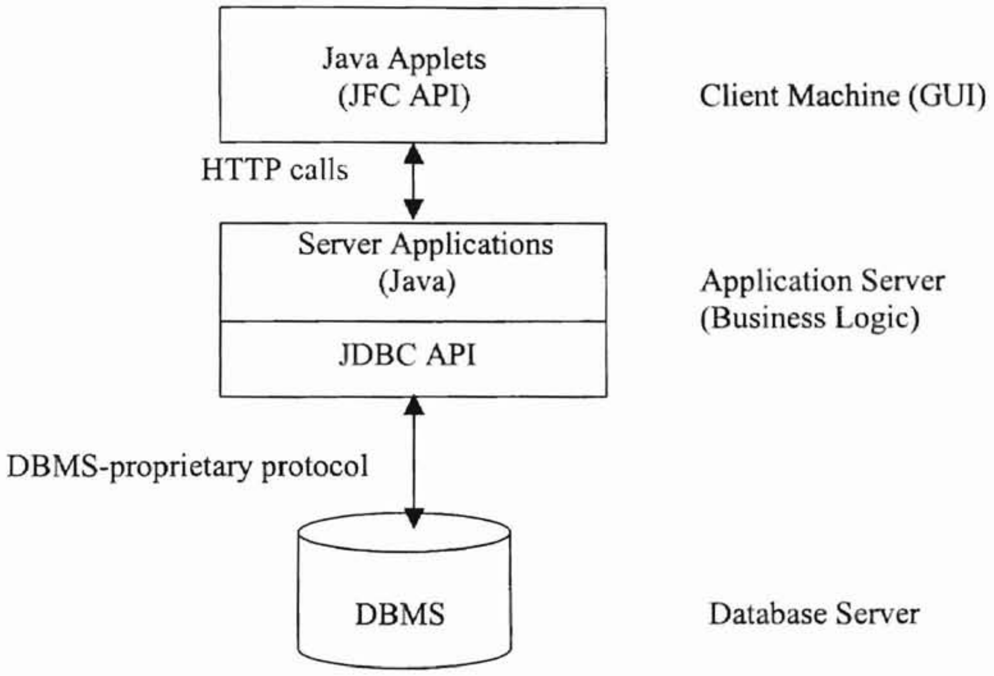


Figure 3.3 JFC Use in Internet-based Three-tier JDBC Approach

CHAPTER IV

APPLICATIONS DESIGN, IMPLEMENTATION AND RESULTS

To illustrate the strategy that the author presented in Chapter III, the author designed and developed two applications. This chapter introduces in detail the applications' computing architecture design together with JFC features, JDBC component design, implementation details, and results.

4.1 Computing Architecture Design

In this thesis, the author designed and developed two applications. Application I is a JDBC-enabled database application running on the Windows operating system; Application II is an JDBC-enabled database application launched by a Web browser.

Both applications adopt the three-tier client-server model. As described in Chapter III, the client contains only the user interface. In application I, the client can be any MS-DOS system; in Application II, the client is a web browser. For both applications, the server houses the data source, and the middle tier contains the business logic. The middle tier resides on IDS Servers; the data sources are implemented on Microsoft Access 97.

In the three-tier model, the client sends commands to a middle tier of services, which passes processed requests to the data source server. The data source processes the commands and sends the results back to the middle tier, which then returns them to the client. For simplicity, the author adopted a personal computer configuration in which all three tiers run on the same machine. Figure 4.1 depicts these two models.

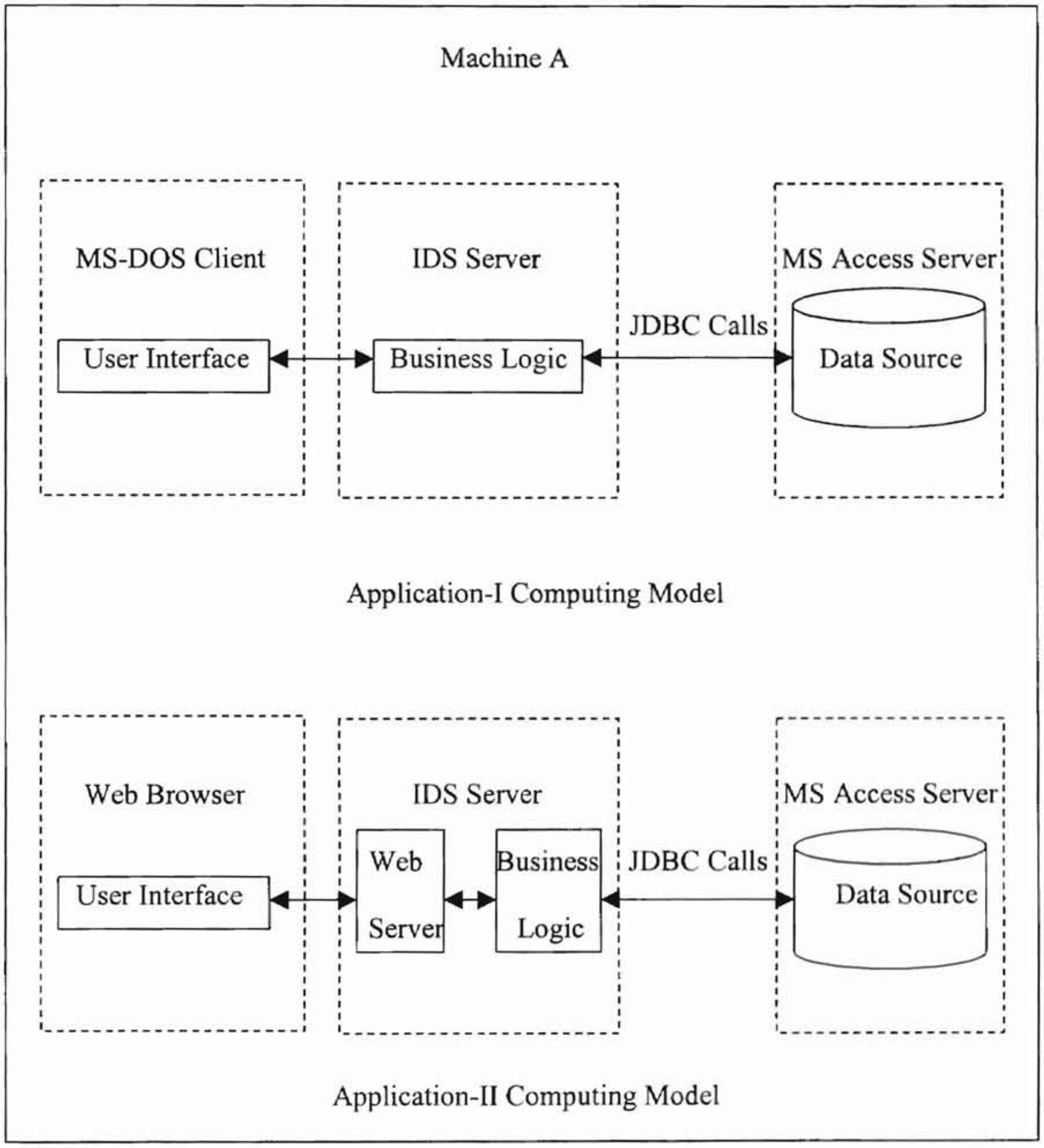


Figure 4.1 Computing Models for Application-I and Application-II

The IDS Server is an Internet database access server that lets developers create web database applications that communicate with back-end DBMSs. It can also serve as a database access server in a distributed application on a network. The IDS Server supports access to all ODBC-compliant databases, as well as native Oracle, Sybase and several other major databases, through their respective client Application Programming Interfaces (APIs). The IDS Server has been implemented on Windows 95/98, Windows NT, Solaris, and Linux [18].

Aside from providing high performance database access, an IDS Server can also serve as a web server. Hence, a single IDS Server can host and deploy an entire database-enabled application system, as it does in application II [18].

Both applications use the IDS Server 3.1.1 Evaluation Version, which is available for free public download, and which supports JDBC 2.0 API [18].

4.2 JDBC Component Design

4.2.1 Graphical User Interface

Graphical User Interface (GUI) design has become increasingly central to Web database applications. To make it easier for people, especially those without extensive training in programming and computer science, to work effectively with the back-end DBMSs, Web database applications should provide very user-friendly interfaces. Today, a number of software products and tools on the market help developers implement user-friendly interfaces. In application I, the author employed a new yet very robust Java technology, Java Foundation Classes (JFC), to design and implement the GUI.

Early in Java's history, the Abstract Window Toolkit (AWT) fulfilled a key role in

GUI design for Java applications. It provides developers with a rudimentary library for building graphical components and services, and offers two important features for all Java applets and applications [6][23]:

- 100% portability for any Java technology-enabled platform from a single set of source code;
- native look and feel on different deployment platforms by using each Java component class to "wrap" the native implementation or peer (JDK refers to this as the "Peer Model").

However, AWT has several weaknesses. It provides only very basic components, functionality and services. This makes it very difficult to develop modern large-scale and commercial-grade applications using AWT. In addition, AWT's peer model is too restrictive in rendering and event handling. As a consequence, it is difficult to extend or override different aspects of the component [5][23].

In Feb. 1998, Sun Microsystems, Inc. released the first shipping version of the Java Foundation Classes (JFC) software, which is available for free download [7]. JFC consists of a comprehensive set of pre-built GUI components and foundation services for 100% pure Java development. It is a superset of AWT, and is fully AWT-compatible. JFC extends the AWT component set by adding a comprehensive set of lightweight and peerless components. These 100% pure Java components provide developers with enormous flexibility and the ability to customize the look and feel of JFC-based applications. In addition, JFC delivers sophisticated foundation services such as Java 2D^{TM} API, Pluggable Look and Feel, accessibility features for people with disabilities, and many other services. JFC's rich toolkit of components and services offers developers great functionality, portability and extensibility [17].

The latest version of JFC, JFC 1.1, contains the following features [5]:

- Swing Components
- Pluggable Look and Feel Support
- Delegation Event Model
- Accessibility API
- Java 2D^{TM} API (Java 2 platform only)
- Drag and Drop Support (Java 2 platform only)

Application I's GUI uses the first three JFC features. At the heart of JFC is a broad set of basic high-level GUI components (code-named "Swing"), including buttons, menus, tool bars, trees and tables. Unlike AWT components, Swing components are implemented without native code. These lightweight components have no platform restrictions, giving them more functionality than AWT components. For instance, Swing buttons can display images and don't have to be rectangular, while AWT buttons can only display text and must be rectangular. In addition, all the Swing components are JavaBeans components. JavaBeans is the Java component architecture standard that allows developers to create components and expose their capabilities in a consistent, standardized manner. Hence, it is very easy for developers to bring in other JavaBeans GUI components to enhance their JFC applications [5][22].

Another key feature used in application I is the JFC Pluggable Look and Feel option. Because of limitations of the peer model, AWT components necessarily share the look and feel of the native platform. Using the lightweight, peerless JFC components and the pluggable look and feel feature, developers can create applications that have the look and feel of a user's native desktop for Windows, Solaris or Macintosh machines. In addition, they can create their own Java-based look and feel as a cross-platform solution [5][22]. In application I, the author implemented three look and feel patterns to give application users the ability to switch the application's GUI from one look and feel to another at runtime without restart.

AWT uses a very simple event model implemented through the Component class. Any class that handles events must trace its ancestry back to the Component class [23]. This requires complex if-then-else conditional logic at the top level to determine which object triggers an event. This technique is not scalable, and is ill-suited to high-performance distributed applications. In application I, the author used the JFC Delegation Event Model, that overcomes this limitation [5]. With this model, events are identified by their class, and are propagated or delegated from an event Source to an event Listener. Any object interested in a particular event can become a listener and deal with the event without passing it to the super event handler. Hence, this model provides an elegant yet powerful way to develop large scale applications by providing a clean separation between the GUI and the business logic.

Although JFC has many advanced and modern features for GUI design, no effort has been given to employ JFC features in JDCB-enabled Web database applications. Thus, through application I, the author illustrates this improvement.

In application II, a simple web page using Hyper-Text Markup Language (HTML) technology provides the GUI. HTML is widely used in web applications GUI design and is described in Chapter II.

4.2.2 JDBC Driver

Both applications use the IDS JDBC Driver from IDS Server 3.1.1 Evaluation Version to communicate with the data source. The IDS JDBC Driver is a compact, efficient, high-performance type-3 JDBC driver [18].

As illustrated in Chapter III, there are four types of JDBC Driver. Type 1 and type 2 drivers rely partially or completely on native binary modules. As a result, drivers of these types are poor choices for web database applications over the Internet. Type 3 and type 4 drivers are written in pure Java code and are truly platform-independent. Both can be distributed via web servers or installed as local Java packages in any client computers.

The IDS JDBC driver consists of several Java packages [18].

- ids.sql package: contains all of the IDS software Java classes that implement the JDBC API. This package is suitable for JDK 1.1, JDK 1.2, and browsers with compatible Java VM.
- j102.sql package: suitable for web browsers with JDK 1.0.2.
- ids.ssl and ids.security packages: the secure socket layer and cryptography packages for Secure JDBC, an important feature provided by the IDS Server that uses state-of-the-art cryptography technology to protect the data exchanged between the IDS JDBC driver and the IDS server against malicious attacks.
- j102.ssl and j102.security packages: the counterparts of ids.ssl and ids.security packages for JDK 1.0.2 browsers.

4.2.3 Data Source

Both application use the Northwind database as the data source. The Northwind database comes with Microsoft Access 97 as a sample database. It has eight relations or tables as shown in Table 4.1.

Relation	Attributes
Categories	CategoriesID, CategoryName, Description, Picture
Customer	CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax
Employees	EmployeeID, LastName, FirstName, Title, TitleOfCourtesy, BirthDate, HireDate, Address, City, Region, PostalCode, Country, HomePhone, Extension, Photo, Notes, ReportsTo
Order Details	OrderID, ProductID, UnitPrice, Quantity, Discount
Order	OrderID, CustomerID, EmployeeID, OrderDate, RequiredDate, ShippedDate, ShipVia, Freight, ShipName, ShipAddress, ShipCity, ShipRegion, ShipPostalCode, ShipCountry
Products	ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued
Shippers	ShipperID, CompanyName, Phone
Suppliers	SupplierID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax, HomePage

Table 4.1 Relations and Their Attributes in Northwind Database

Figure 4.2 Shows an Entity-Relationship (E-R) diagram for the Northwind database. The database system may reject any operation that violates the referential

integrity constraints, greatly reducing the likelihood of database corruption.

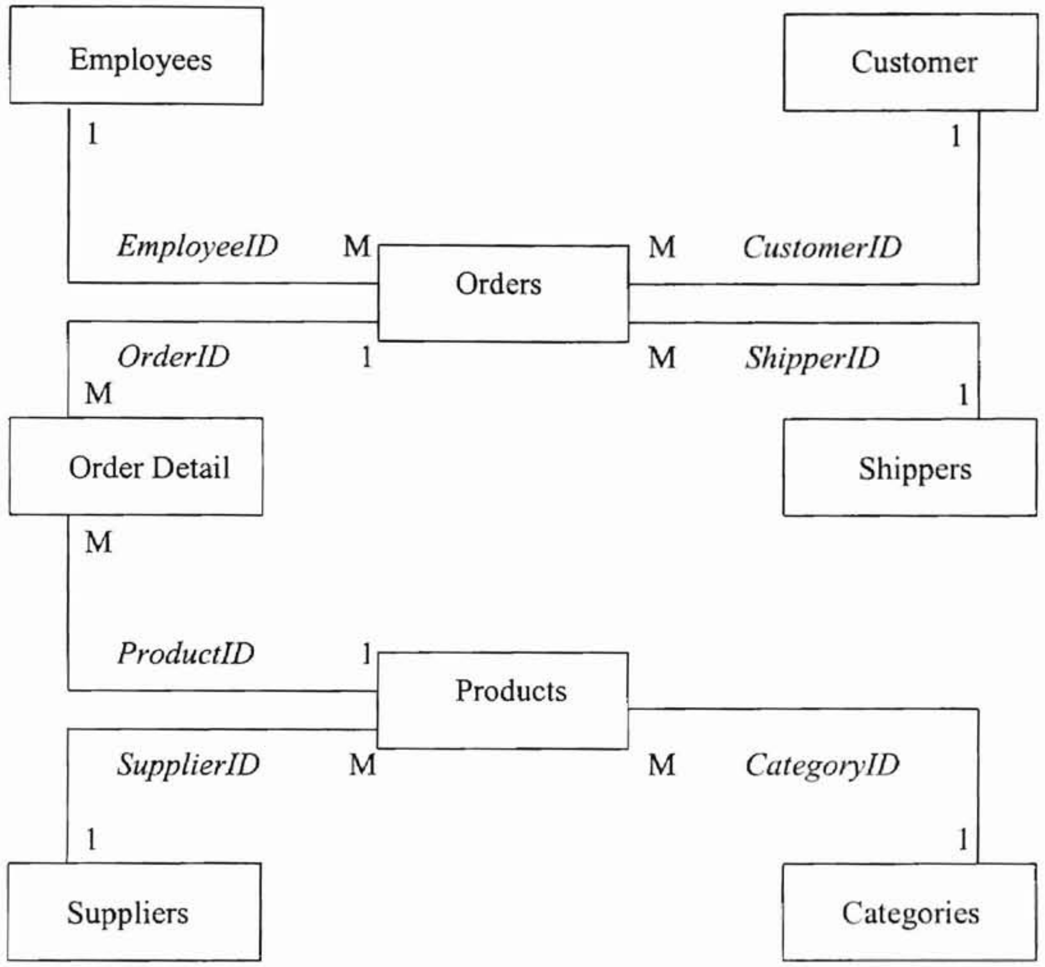


Figure 4.2 E-R Diagram for Northwind Database

4.3 Implementation Details

4.3.1 System Environment

Table 4.2 gives system requirements for the applications. Among the required software, Java 2 platform, Standard Edition, v 1.2.2 and the IDS Server 3.1.1 Evaluation Version can be downloaded free from Sun Microsystems and IDS Software, respectively.

Hardware	• OS/Platform - Win32/Intel (Windows 95/98/NT)
	• RAM - 16 MB minimum, 64 MB memory recommended
Software	• Java^{TM} 2 platform, Standard Edition, v1.2.2
	• IDS Server Evaluation Version, v3.1.1
	• Microsoft Access 97 System
	• JDK 1.0.2-Compatible Web Browser

Table 4.2 System Environments

4.3.2 IDS Server and IDS JDBC Driver Set Up

On Windows, the IDS Server is distributed in a single self-extracting file, that creates the directory hierarchy in Figure 4.3. The directory "IDSServer\classes\ids" contains the IDS JDBC driver for JDK 1.1 and JDK 1.2; "IDSServer\classes\j102" contains the driver for JDK 1.0 [18].

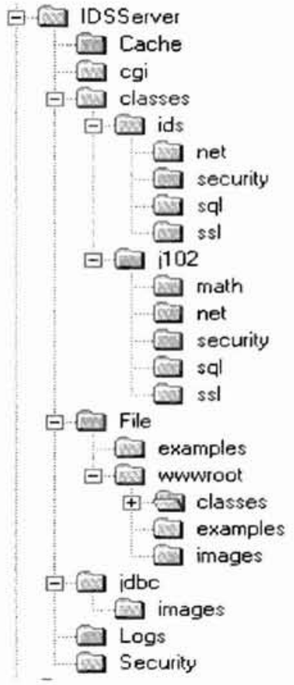


Figure 4.3 IDS Server Directory Hierarchy

4.3.3 JDBC API Implementations

1. Importing the JDBC Driver Classes for Use

For any database application to use the IDS JDBC driver, the pure-Java IDS JDBC driver classes should be imported into the beginning of the program. Figure 4.4 and Figure 4.5 show the line of code in the beginning of both application I and application II.

```
import ids.sql.*;
```

Figure 4.4 Importing JDBC Driver Classes in Application-I

```
import j102.sql.*;
```

Figure 4.5 Importing JDBC Driver Classes in Application-II

2. Establishing a Connection

All JDBC operations begin with the creation of a connection with the DBMS the application needs to access. There are two steps to do this: to set up a Connection URL and to make a connection by the JDBC driver using this URL.

The Connection URL is used by JDBC driver to connect to a remote or local DBMS. Its basic syntax for IDS JDBC Driver is as follows. The syntax complies with the naming convention recommended by the JDBC API and resembles a regular URL.

```
Jdbc:ids://host_domain:port_number/conn?dbms=odbc&dsn='data source'\
    &uid='user id'&pwd='password'&rcs=0&mts=0&ssl=0&share=0
```

Figure 4.6 Syntax of Connection URL for IDS JDBC Driver

In figure 4.6, the leading "jdbc:ids://" signifies an IDS JDBC Connection URL and is mandatory. host_domain and port_number are the IP address and port number of the installed IDS Server. In the two applications, the IDS Server runs in a stand-alone PC, the server's IP address is "localhost" or "127.0.0.1", and the port number is 12 by default. After the port number, the keyword "conn" is the JDBC required identifier for the IDS Server and is mandatory. The substring that follows the question mark '?' in the syntax is a parameter list. All parameters are in the "name=value" format separated by an ampersand '&'. Among the parameters, "dbms" specifies the type of database interface to be used; the default value is "odbc". The "dsn" parameter specifies the ODBC data source of the JDBC connection and is mandatory. All other parameters are not mandatory and are not used in the two applications.

There are two ways to make a connection with the IDS JDBC Driver. First is through the JDBC DriverManager class by calling the static method getConnection( ); application I uses this method. The other way is to create a connection by first instantiating the Driver class directly, then calling the connect( ) method of the Driver instance. Application II uses this way. Figure 4.7 and 4.8 show the code fragment used in two applications.

```
Connection theConnection;
String databaseURL;

DatabaseURL = "jdbc:ids://localhost:12/conn?dsn='NorthWind'";
Class.forName("ids.sql.IDSDriver").newInstance( );
TheConnection = DriverManager.getConnection(databaseURL);
```

Figure 4.7 Establishing a Connection in Application-I

```
Connection theConnection;
IDSDriver theDriver;
Stirng databaseURL;

DatabaseURL = "jdbc:ids://localhost:12/conn?dsn='NorthWind'";
TheDriver = new j102.sql.IDSDriver( );
TheConnection = theDriver.connect(databaseURL, null);
```

Figure 4.8 Establishing a Connection in Application-II

3. Creating and Executing JDBC Statements

A Statement object sends SQL statements to a database. The IDS JDBC Driver is implemented in full compliance with the JDBC API Specification, and has additional features. In the two applications, the author only used general JDBC features, including the Statement object.

Three are three kinds of Statement objects: Statement, PreparedStatement and

CallableStatement. The Statement described in this section is used to execute a simple SQL statement with no parameter. The PreparedStatement object is suitable for executing a precompiled SQL statement with or without parameters, and will be described in a later section. The CallableStatement is used to execute a call to a database stored procedure. Because Microsoft Access does not support the stored procedure feature, CallableStatement object is not implemented in the applications.

Once a connection to a particular database is established, that connection can be used to send SQL statements A Statement object is created with the createStatement method in Connection class, as seen in Figure 4.10.

After creating the Statement object, the SQL statement that will be sent to the database is supplied as the argument to one of the "execute" methods on this Statement object. Two most often used methods are executeQuery and the executeUpdate. The executeQuery is designed for SQL statements that produce a single result set, such as SELECT statements. The executeUpdate is used to execute INSERT, UPDATE or DELETE statements, and SQL Data Definition Language (DDL) statements like CREATE TABLE or DROP TABLE. Figure 4.9 also shows a sample of executing SQL statement.

```
Connection theConnection;
Statement theStatement;
            .
            .  // Established a connection
            .
theStatement = theConnection.createStatement( );

String theQuery;
String theUpdate;

theQuery = "SELECT * FROM Employees";
theUpdate = "UPDATE Employees SET LastName = 'Johnston'
            WHERE EmployeeID = 10";
theStatement.executeQuery(theQuery);
theStatement.executeUpdate(theUpdate);
```

Figure 4.9 Creating and Executing JDBC Statements

4. Retrieving Values from an SQL Query

JDBC returns the results of executing an SQL query in a ResultSet object. A ResultSet object is a table that contains rows that satisfy the conditions of the query.

The data stored in a ResultSet object is retrieved through a set of "get" methods. Each get method retrieves the data from the various columns of the current row and converts it to a particular Java data type. The next( ) method in ResultSet is used to move to the next row of the ResultSet table, making it the current row. Figure 4.10 gives a example of how to retrieve results from a query.

```
Connection theConnection;
Statement theStatement;
            .
            .  // Established a connection
            .
theStatement = theConnection.createStatement( );

ResultSet theResultSet;
int theEmployeeID;
String theLastName;

String theQuery = "SELECT EmployeeID, LastName FROM Employees";
TheResultSet = theStatement.executeQuery(theQuery);

While (theResultSet.next( )) {
    TheEmployeeID = theResultSet.getInt("EmployeeID");
    TheLastName = theResultSet.getString("LastName");
    System.out.println(theEmployeeID + " : " + theLastName);
}
```

Figure 4.10 Retrieving Results from the Query

5. Mapping SQL and Java Data Types

For the set of get methods, the JDBC driver attempts to convert the underlying data to the specific Java Type and then returns a suitable Java Type. Table 4.3 shows a list of the Microsoft Access 97 data types, their corresponding Java types, and the corresponding JDBC get methods.

Access Type	Java Type	get Method
Text	String	getString()
Memo	String	getASCIIStream()
Number	Java.sql.Numeric	getNumeric()
Yes/No	Boolean	getBoolean()
Byte	Byte	getByte()
Integer	Short	getShort()
Long	Int	getInt()
Long	Long	getLong()
Single	Float	getFloat()
Double	Double	getDouble()
OLE object	byte[]	getBytes()
Date/Time	Java.sql.Date	getDate()
Date/Time	Java.sql.Time	getTime()
Date/Time	Java.sql.Timestamp	getTimeStamp()

Table 4.3 Mapping of Access and Java Data Types, and the get Methods

6. Creating, Setting and Executing PreparedStatements

The PreparedStatement object is primarily used for the execution of dynamic SQL statements. It allows the application to use the same statement and supply it with different values each time the statement is executed. A PreparedStatement object is created with the Connection class prepareStatement( ) method, as shown in Figure 4.11.

```
PreparedStatement thePreparedStatement;
                    .
                    .  // Established a connection
                    .
thePreparedStatement = theConnection.prepareStatement
      ("UPDATE Employees SET LastName = ? WHERE EmployeeID = ?")
```

Figure 4.11 Creating a PreparedStatement

In Figure 4.12, each question mark (?) is a placeholder for a value that will be supplied when the statement is executed by the PreparedStatement set methods. After being supplied the values, most PreparedStatements can be executed by using the PreparedStatement class executeQuery and executeUpdate methods, these methods are similar in use to the two execute methods in Statement class. As an example, the code fragment in Figure 4.12 sets the two values in Figure 4.11 and executes the above prepared statement.

```
                .
                . // Continue from last example
                .
    thePreparedStatement.setString(1, "johnston");
    thePreparedStatement.setInt(2, 100);
    thePreparedStatement.executeUpdate( );
```

Figure 4.12 Setting and Executing the PreparedStatement

One big advantage of using PreparedStatement object is that the prepared statements are pre-compiled by the database. When they are executed, the DBMS can just run the SQL statements without having to compile them first. This results in quicker response time and lower work loads for the DBMS, especially when very large statements or statements that are repeated many times are executed.

7. Closing the Connection

When the application is done accessing the data source, it must close the database connection in order to free any resources associated with the connection. Any open ResuleSet, Statement, PreparedStatement or other objects being created are closed automatically. See Figure 4.13.

```
Connection theConnection;
          .
          .   // Established a connection
          .
// After database accessing is done
theConnection.close( );
```

Figure 4.13 Closing the Connection

4.4 Results

4.4.1 Execution of Application I

- Figure 4.14 shows the welcome window of the application I GUI. There are five menus on this window: Connect, Query, Utilities, Options and Help. If there is no connection being established between the application and a data source, the Query and Utilities menu are unselectable.

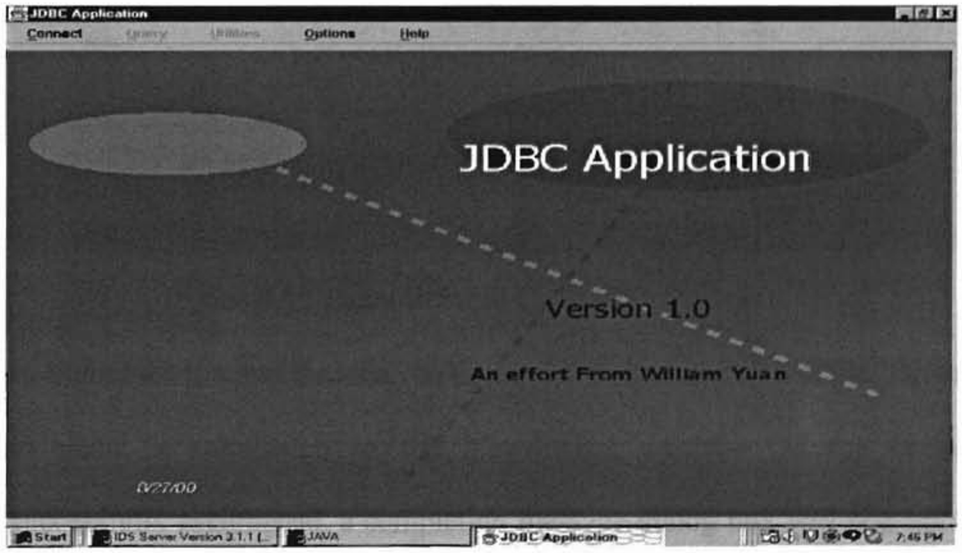


Figure 4.14 Welcome Window of the GUI

- The Connect menu contains three menu items: Open Connection, Close Connection and Exit, as shown in Figure 4.15. To access a data source, the user must make a connection to it first. When the user clicks the Open Connection menu item, a dialog box will show up and enable the user to select a data source from a drop-down list of available sources. This application only takes use of one source, the Northwind database, and no user id or password is required. Figure 4.16 shows this dialog box.

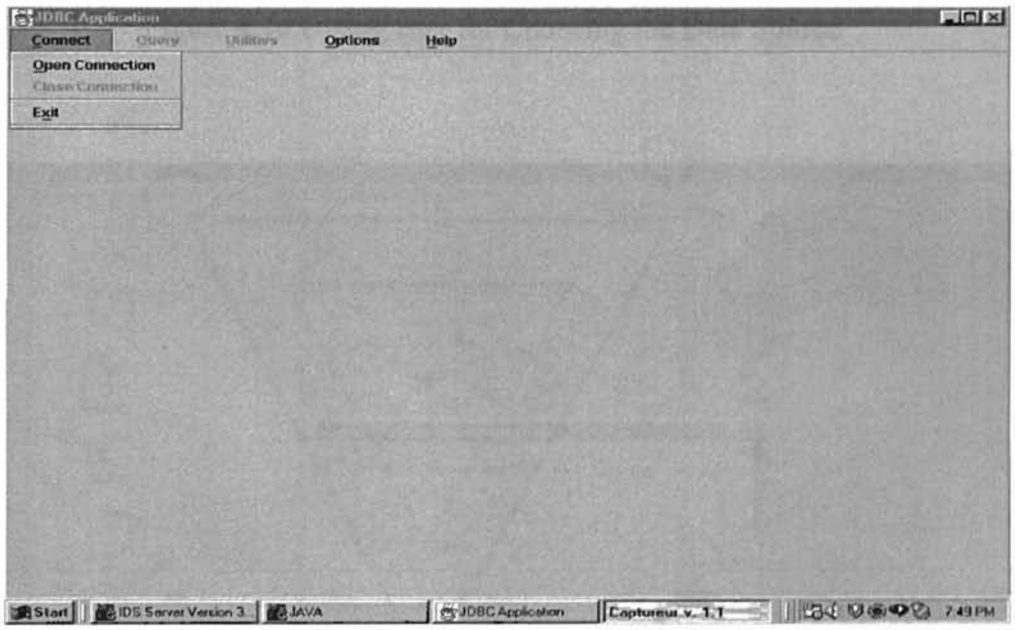


Figure 4.15 Menu Items in Connection Menu

In Figure 4.16, when the user clicks the OK button, the IDS JDBC Driver will try to make a connection to the Nowthwind database. If the connection is successfully established, a completion message dialog box will pop-up, as shown in Figure 4.17; otherwise, an error message box appears.

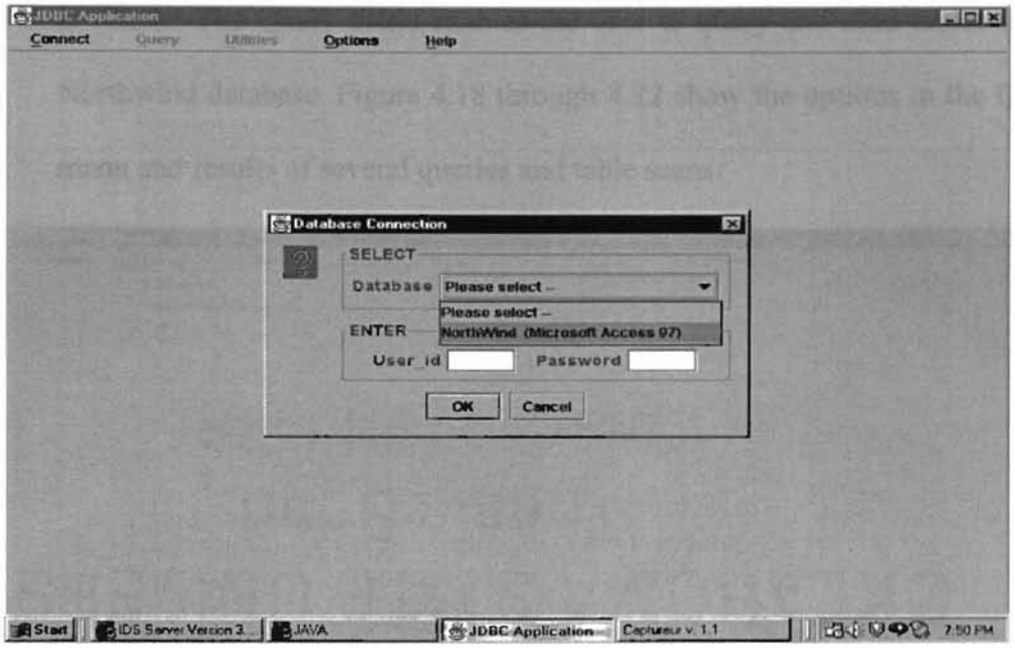
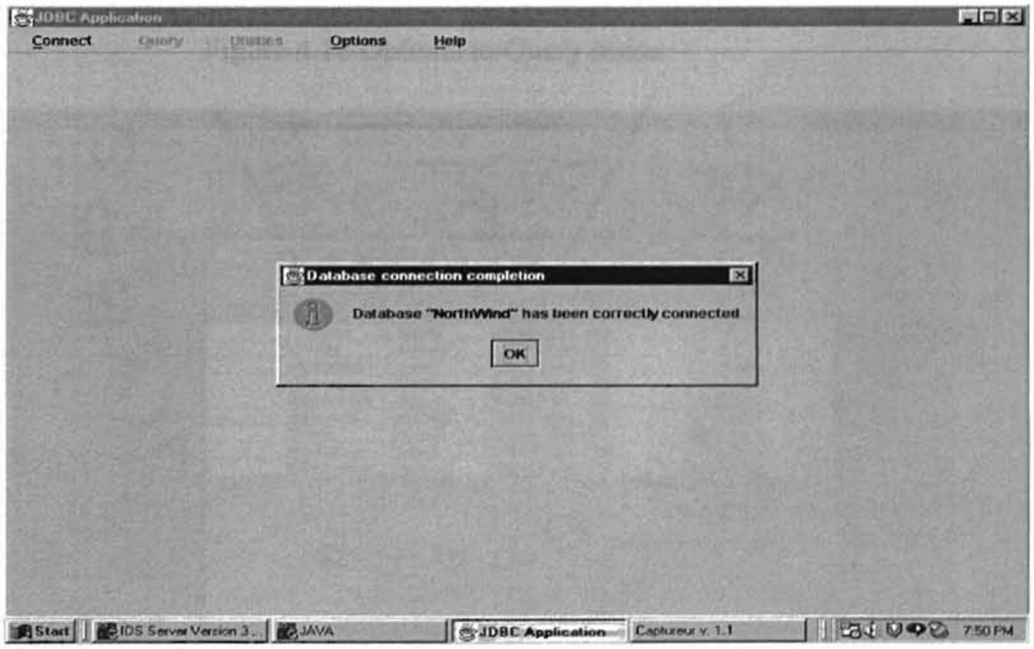


Figure 4.16 Dialog Box for Choosing the Data Source



Database "NorthWind" has been correctly connected

Figure 4.17 Connection Completion Dialog Box

- After setting up the connection, the Query and Utilities menus become available. The Query menu enables the user to query and scan tables in the Northwind database. Figure 4.18 through 4.22 show the options in the Query menu and results of several queries and table scans.

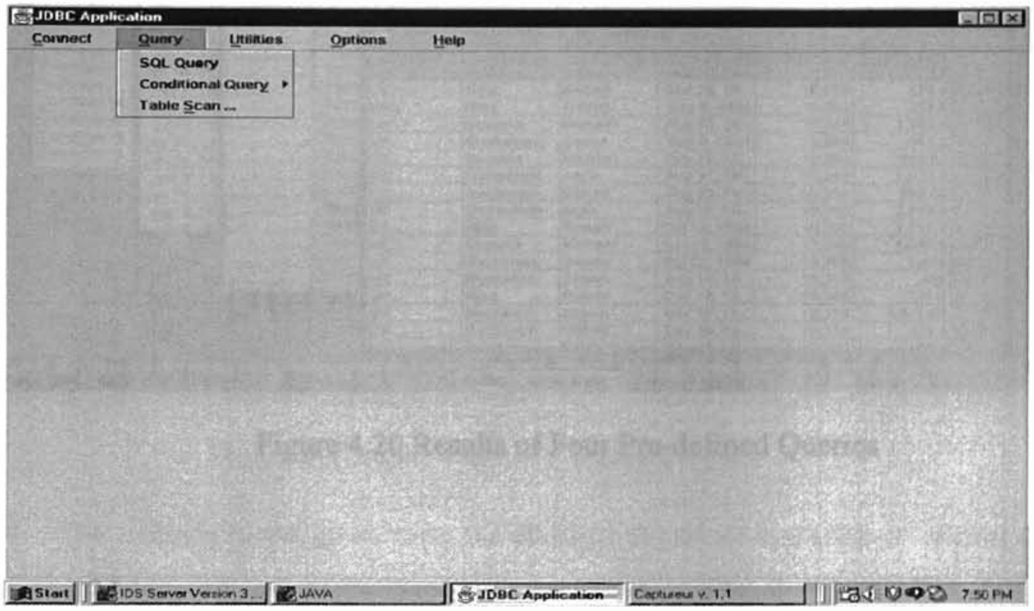
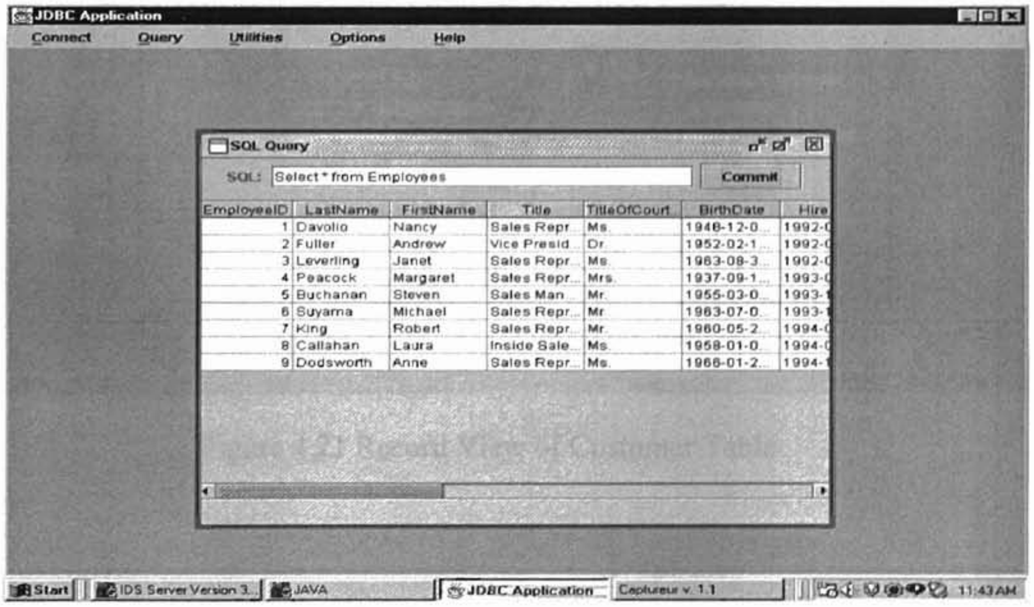


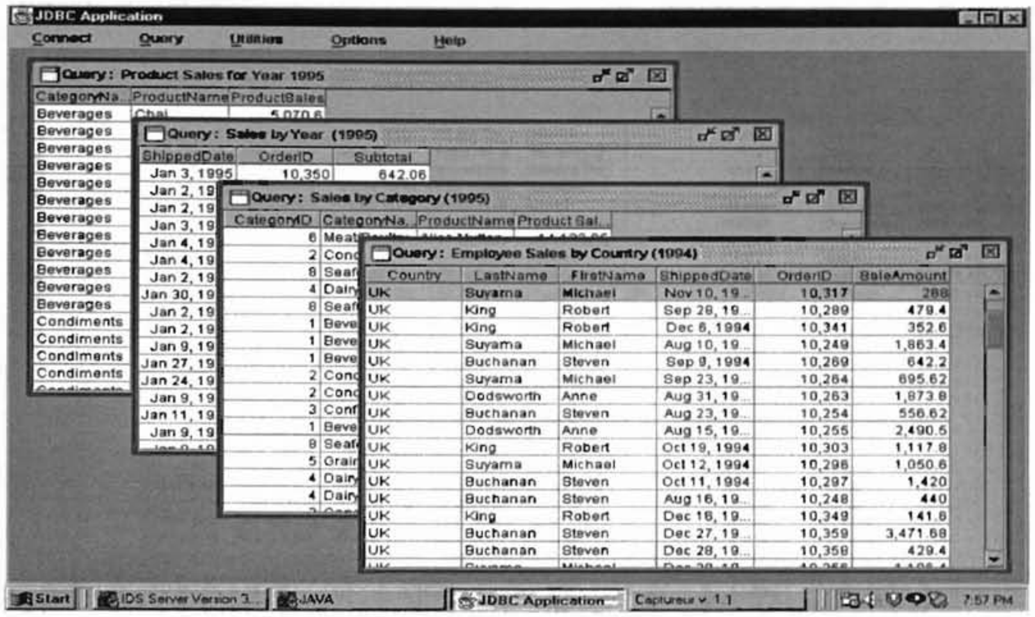
Figure 4.18 Options in Query Menu



SQL: Select * from Employees

EmployeeID	LastName	FirstName	Title	TitleOfCourt	BirthDate	Hire
1	Davolio	Nancy	Sales Repr...	Ms.	1948-12-0...	1992-(
2	Fuller	Andrew	Vice Presid...	Dr.	1952-02-1...	1992-(
3	Leverling	Janet	Sales Repr...	Ms.	1963-08-3...	1992-(
4	Peacock	Margaret	Sales Repr...	Mrs.	1937-09-1...	1993-(
5	Buchanan	Steven	Sales Man...	Mr.	1955-03-0...	1993-1
6	Suyama	Michael	Sales Repr...	Mr.	1963-07-0...	1993-1
7	King	Robert	Sales Repr...	Mr.	1960-05-2...	1994-(
8	Callahan	Laura	Inside Sale...	Ms.	1958-01-0...	1994-(
9	Dodsworth	Anne	Sales Repr...	Ms.	1966-01-2...	1994-1

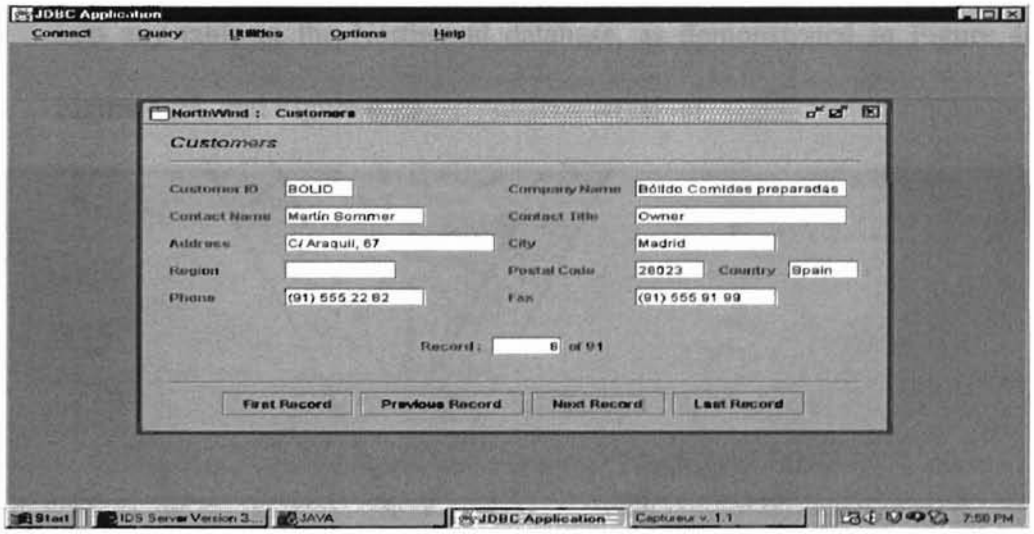
Figure 4.19 Result of a Run-time Query



Query : Employee Sales by Country (1994)

Country	LastName	FirstName	ShippedDate	OrderID	SaleAmount
UK	Suyama	Michael	Nov 10, 19...	10,317	288
UK	King	Robert	Sep 28, 19...	10,289	479.4
UK	King	Robert	Dec 6, 1994	10,341	352.6
UK	Suyama	Michael	Aug 10, 19...	10,249	1,863.4
UK	Buchanan	Steven	Sep 9, 1994	10,269	642.2
UK	Suyama	Michael	Sep 23, 19...	10,264	695.62
UK	Dodsworth	Anne	Aug 31, 19...	10,263	1,873.8
UK	Buchanan	Steven	Aug 23, 19...	10,254	556.62
UK	Dodsworth	Anne	Aug 15, 19...	10,255	2,490.5
UK	King	Robert	Oct 19, 1994	10,303	1,117.8
UK	Suyama	Michael	Oct 12, 1994	10,296	1,050.6
UK	Buchanan	Steven	Oct 11, 1994	10,297	1,420
UK	Buchanan	Steven	Aug 16, 19...	10,248	440
UK	King	Robert	Dec 16, 19...	10,349	141.6
UK	Buchanan	Steven	Dec 27, 19...	10,359	3,471.68
UK	Buchanan	Steven	Dec 28, 19...	10,358	429.4

Figure 4.20 Results of Four Pre-defined Queries

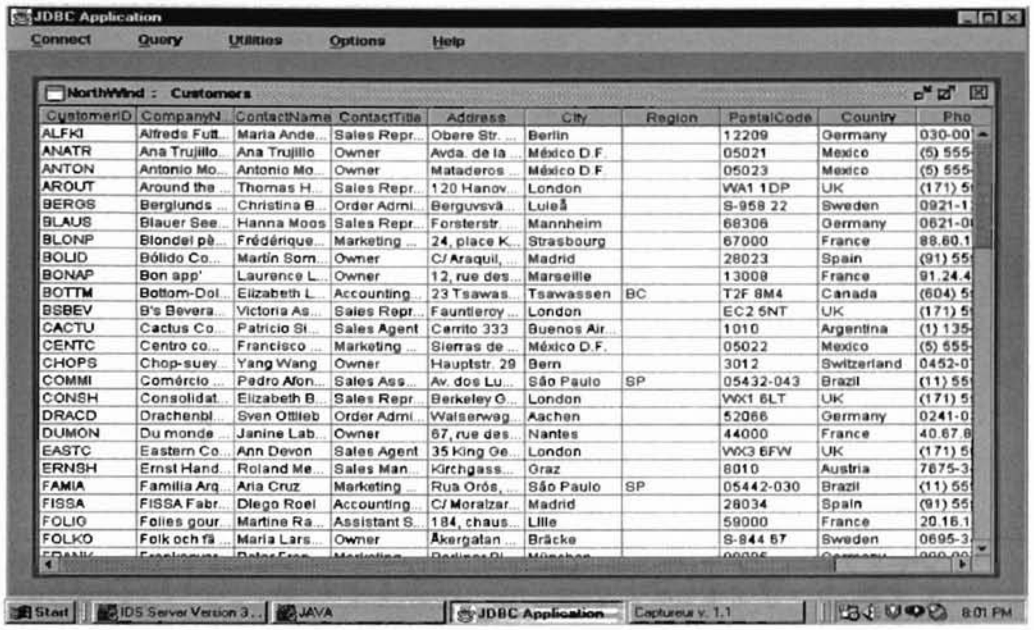


Customers

Customer ID	BOLID	Company Name	Bólido Comidas preparadas
Contact Name	Martín Sommer	Contact Title	Owner
Address	C/ Araquil, 67	City	Madrid
Region		Postal Code	28023 Country Spain
Phone	(91) 555 22 82	Fax	(91) 555 91 99

Record : 6 of 91

Figure 4.21 Record View of Customer Table



CustomerID	CompanyN	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Pho
ALFKI	Alfreds Futt...	Maria Ande...	Sales Repr...	Obere Str. ...	Berlin		12209	Germany	030-00
ANATR	Ana Trujillo...	Ana Trujillo	Owner	Avda. de la ...	México D.F.		05021	Mexico	(5) 555-
ANTON	Antonio Mo...	Antonio Mo...	Owner	Mataderos ...	México D.F.		05023	Mexico	(5) 555-
AROUT	Around the ...	Thomas H...	Sales Repr...	120 Hanov...	London		WA1 1DP	UK	(171) 5
BERGS	Berglunds ...	Christina B...	Order Admi...	Berguvsvä...	Luleå		S-958 22	Sweden	0921-1
BLAUS	Blauer See...	Hanna Moos	Sales Repr...	Forsterstr. ...	Mannheim		68306	Germany	0621-0
BLONP	Blondel pè...	Frédérique...	Marketing ...	24, place K...	Strasbourg		67000	France	88.60.1
BOLID	Bólido Co...	Martín Som...	Owner	C/ Araquil, ...	Madrid		28023	Spain	(91) 55
BONAP	Bon app'	Laurence L...	Owner	12, rue des...	Marseille		13008	France	91.24.4
BOTTM	Bottom-Dol...	Elizabeth L...	Accounting...	23 Tsawas...	Tsawassen	BC	T2F 8M4	Canada	(604) 5
BSBEV	B's Bevera...	Victoria As...	Sales Repr...	Fauntleroy ...	London		EC2 5NT	UK	(171) 5
CACTU	Cactus Co...	Patricio Si...	Sales Agent	Cerrito 333	Buenos Air...		1010	Argentina	(1) 135-
CENTC	Centro co...	Francisco ...	Marketing ...	Sierras de ...	México D.F.		05022	Mexico	(5) 555-
CHOPS	Chop-suey...	Yang Wang	Owner	Hauptstr. 29	Bern		3012	Switzerland	0452-0
COMMI	Comércio ...	Pedro Afon...	Sales Ass...	Av. dos Lu...	São Paulo	SP	05432-043	Brazil	(11) 55
CONSH	Consolidat...	Elizabeth B...	Sales Repr...	Berkeley G...	London		WX1 6LT	UK	(171) 5
DRACD	Drachenbl...	Sven Ottlieb	Order Admi...	Walserweg...	Aachen		52066	Germany	0241-0
DUMON	Du monde ...	Janine Lab...	Owner	67, rue des...	Nantes		44000	France	40.67.8
EASTC	Eastern Co...	Ann Devon	Sales Agent	35 King Ge...	London		WX3 6FW	UK	(171) 5
ERNSH	Ernst Hand...	Roland Me...	Sales Man...	Kirchgass...	Graz		8010	Austria	7675-3
FAMIA	Familia Arq...	Aria Cruz	Marketing ...	Rua Orós, ...	São Paulo	SP	05442-030	Brazil	(11) 55
FISSA	FISSA Fabr...	Diego Roel	Accounting...	C/ Moralzar...	Madrid		28034	Spain	(91) 55
FOLIG	Folies gour...	Martine Ra...	Assistant S...	184, chaus...	Lille		59000	France	20.16.1
FOLKO	Folk och fä ...	Maria Lars...	Owner	Åkergatan ...	Bräcke		S-844 67	Sweden	0695-3

Figure 4.22 Scan View of Customer Table

- The Utilities menu gives users the abilities to update any table or import data into any table in the Northwind database, as demonstrated in Figure 4.23, Figure 4.24, and Figure 4.25.

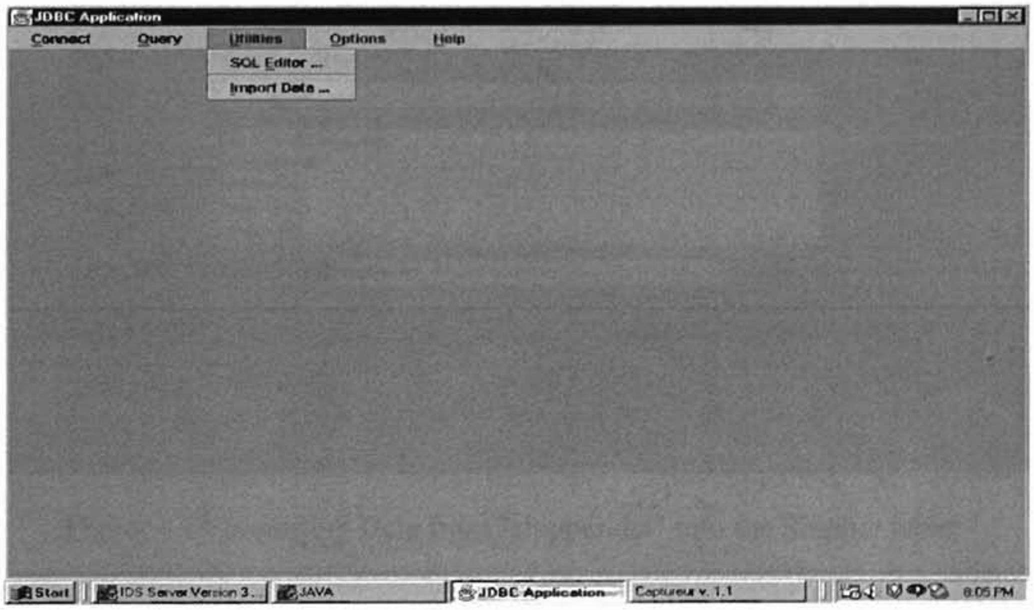
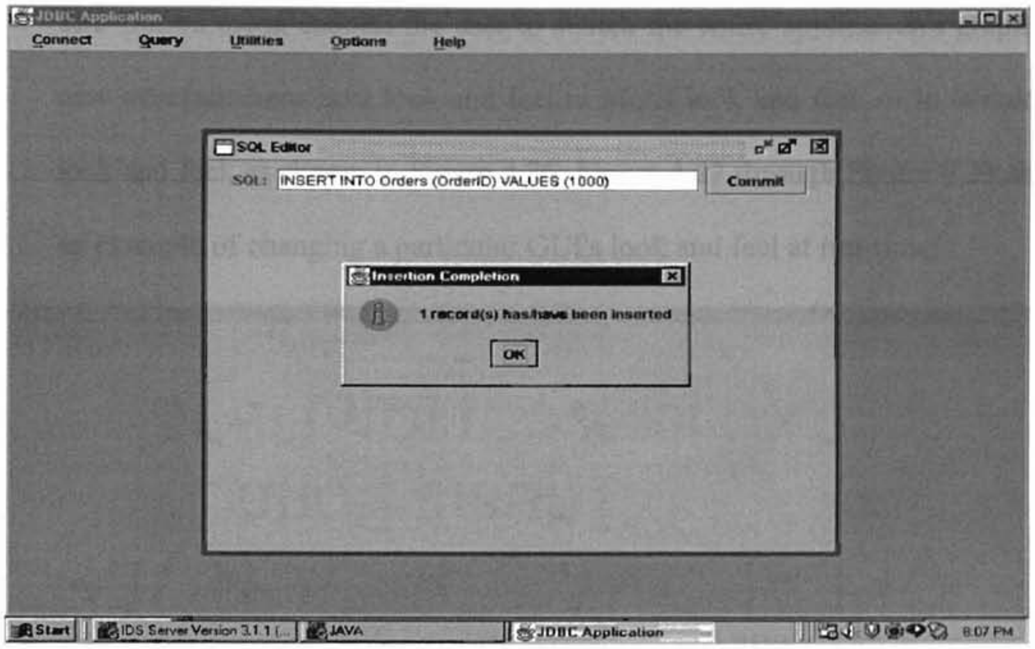


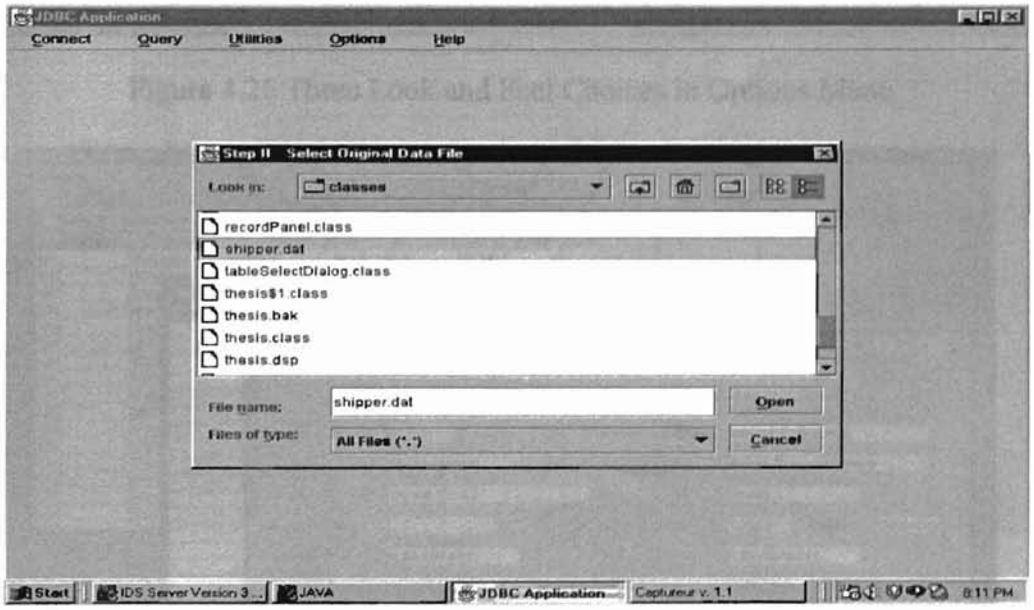
Figure 4.23 The Items in Utilities Menu



SQL: INSERT INTO Orders (OrderID) VALUES (1000)

1 record(s) has/have been inserted

Figure 4.24 Inserting Records into the Order table



Look in: classes

recordPanel.class
shipper.dat
tableSelectDialog.class
thesis$1.class
thesis.bak
thesis.class
thesis.dsp

File name: shipper.dat

Files of type: All Files (*.*)

Figure 4.25 Importing Data from "shipper.dat" into the Shipper table

- The Option menu enables the user to switch the entire application's graphical user interface from Java look and feel to Motif look and feel, or to Windows look and feel, as shown in Figure 4.26. Figure 4.27 through Figure 4.29 show an example of changing a particular GUI's look and feel at run-time.

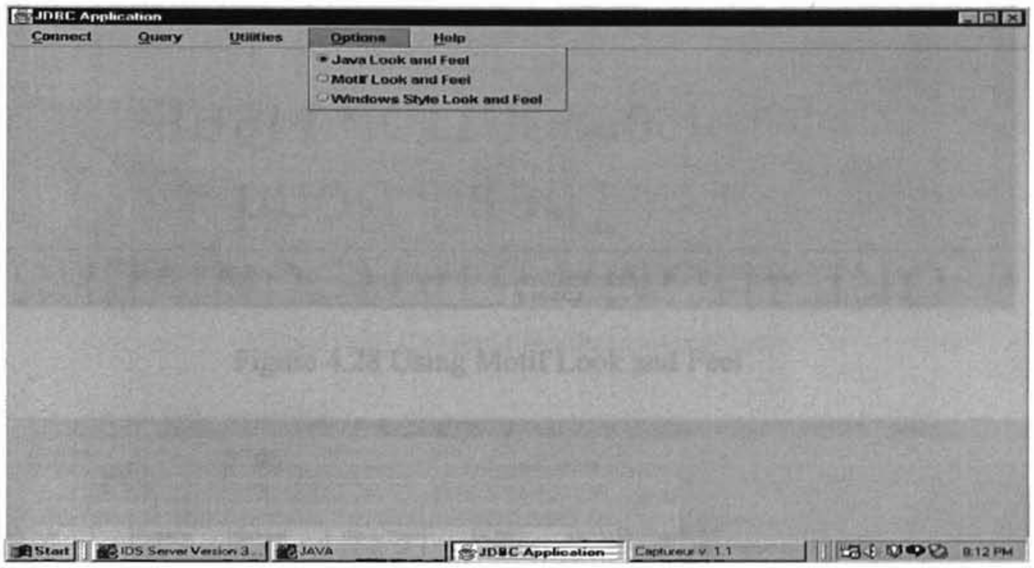
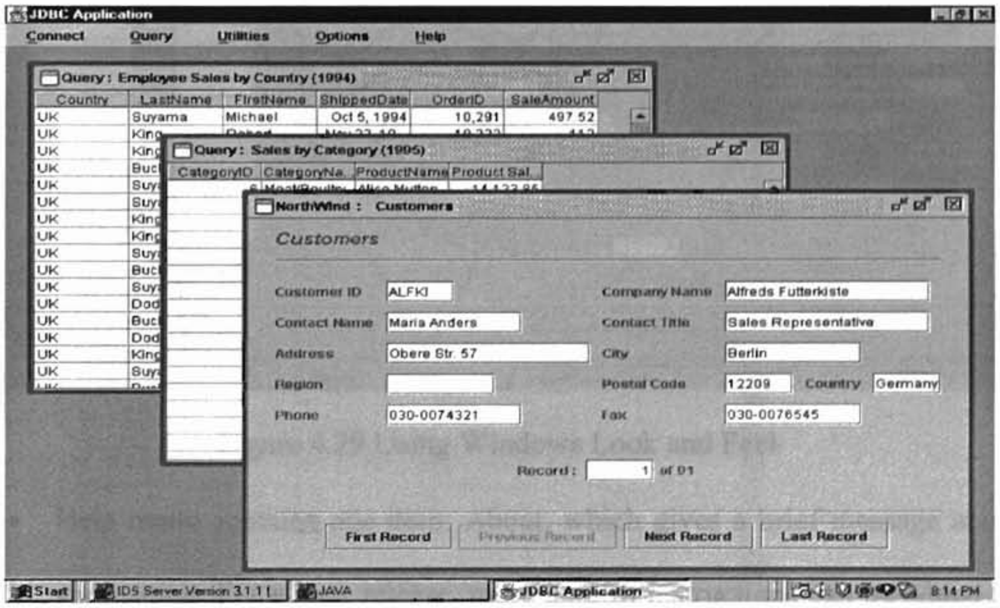


Figure 4.26 Three Look and Feel Choices in Options Menu



Query : Employee Sales by Country (1994)

Country	LastName	FirstName	ShippedDate	OrderID	SaleAmount
UK	Suyama	Michael	Oct 5, 1994	10,291	497.52

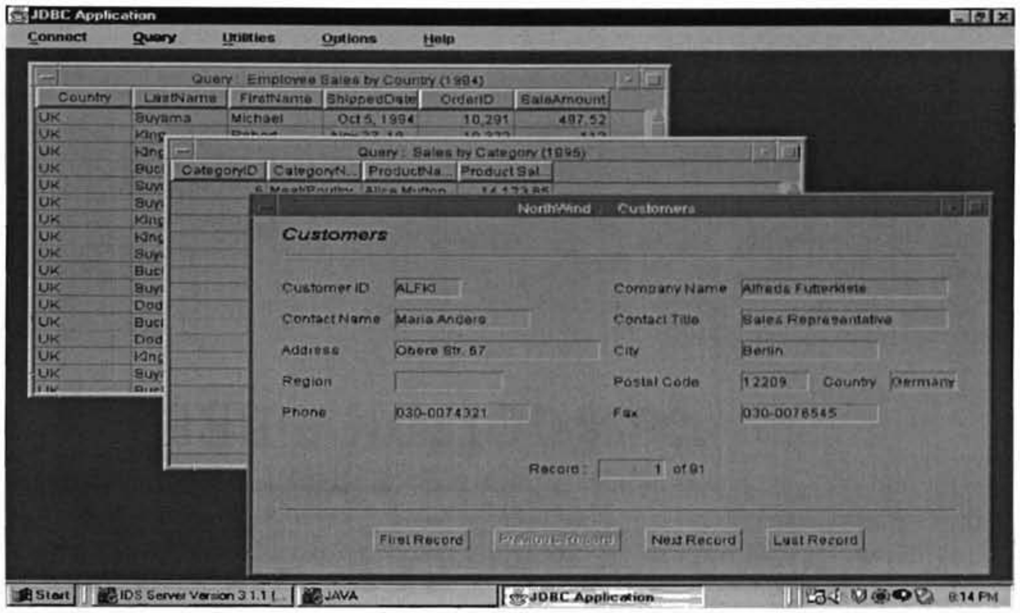
NorthWind : Customers

Customers

Customer ID: ALFKI
Company Name: Alfreds Futterkiste
Contact Name: Maria Anders
Contact Title: Sales Representative
Address: Obere Str. 57
City: Berlin
Region:
Postal Code: 12209
Country: Germany
Phone: 030-0074321
Fax: 030-0076545

Record: 1 of 91

Figure 4.27 Using Java Look and Feel


JDBC Application
Connect Query Utilities Options Help
Query : Employee Sales by Country (1994)
Query : Sales by Category (1995)
NorthWind : Customers
First Record
Previous Record
Next Record
Last Record

Country	LastName	FirstName	ShippedDate	OrderID	SaleAmount
UK	Suyama	Michael	Oct 5, 1994	10,291	497.52

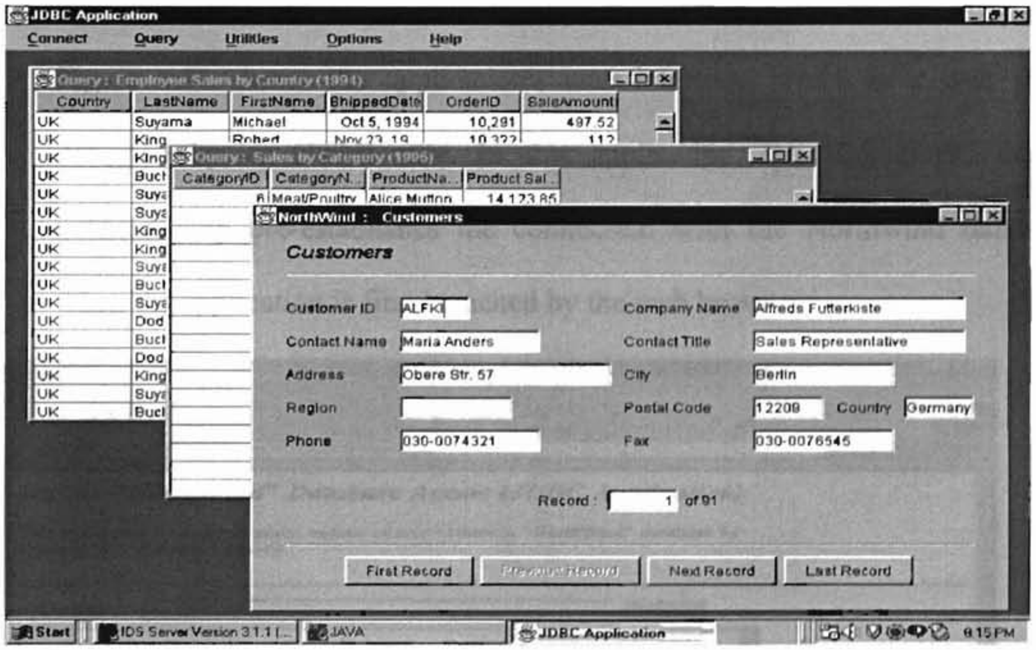
CategoryID	CategoryN...	ProductNa...	Product Sal...
6	Meat/Poultry	Alice Mutton	14,123.85

Customers

Customer ID	ALFKI	Company Name	Alfreds Futterkiste
Contact Name	Maria Anders	Contact Title	Sales Representative
Address	Obere Str. 57	City	Berlin
Region		Postal Code	12209 Country Germany
Phone	030-0074321	Fax	030-0076545

Record : 1 of 91

Figure 4.28 Using Motif Look and Feel


JDBC Application
Connect Query Utilities Options Help
Query : Employee Sales by Country (1994)
Query : Sales by Category (1995)
NorthWind : Customers
First Record
Previous Record
Next Record
Last Record

Country	LastName	FirstName	ShippedDate	OrderID	SaleAmount
UK	Suyama	Michael	Oct 5, 1994	10,291	497.52

CategoryID	CategoryN...	ProductNa...	Product Sal...
6	Meat/Poultry	Alice Mutton	14,123.85

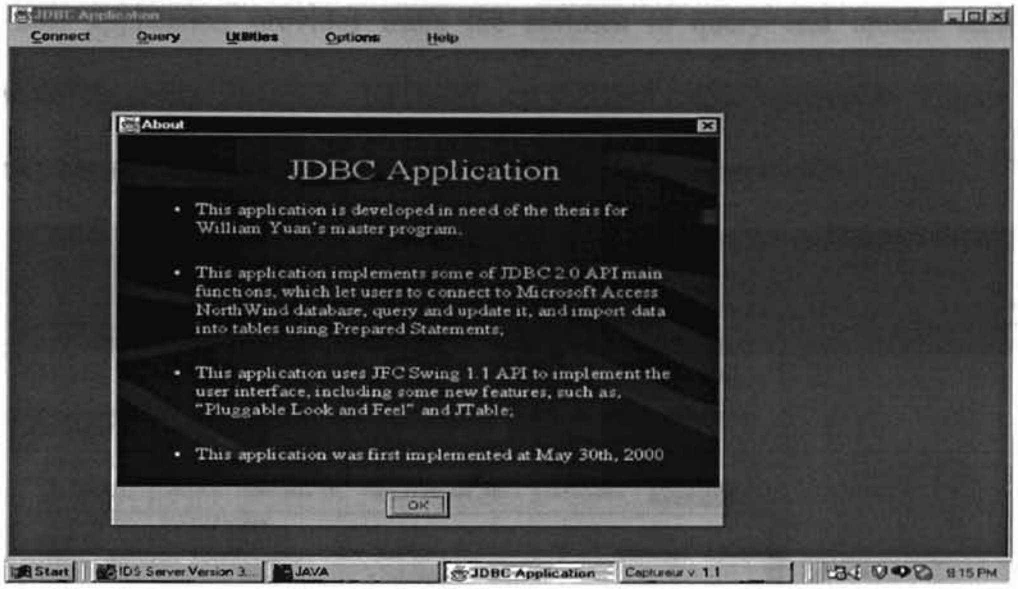
Customers

Customer ID	ALFKI	Company Name	Alfreds Futterkiste
Contact Name	Maria Anders	Contact Title	Sales Representative
Address	Obere Str. 57	City	Berlin
Region		Postal Code	12209 Country Germany
Phone	030-0074321	Fax	030-0076545

Record : 1 of 91

Figure 4.29 Using Windows Look and Feel

- Help menu contains one item: About, which gives a brief message about the application. For a later release, more helpful instructions can be added to this menu. Figure 4.30 the contents in the About item.



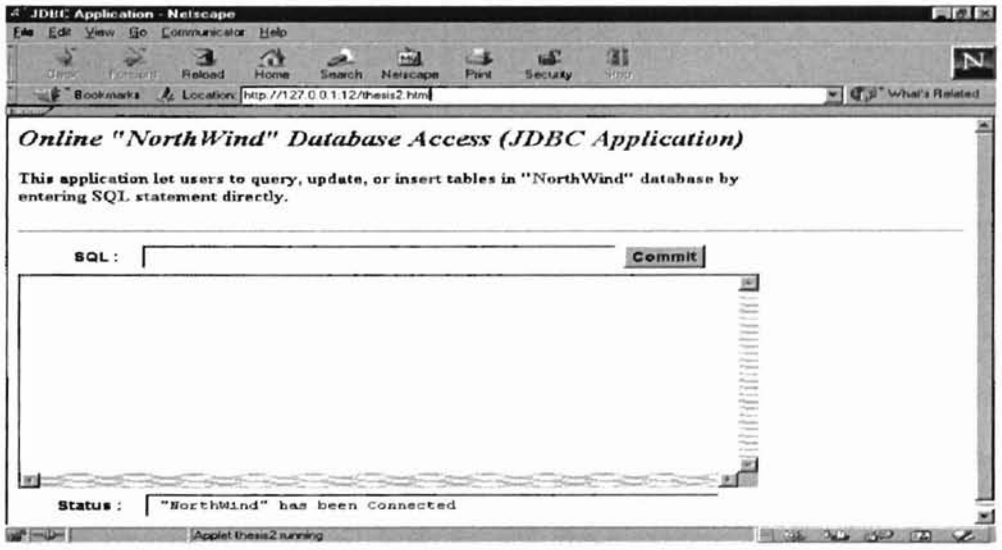
JDBC Application

- This application is developed in need of the thesis for William Yuan's master program.
- This application implements some of JDBC 2.0 API main functions, which let users to connect to Microsoft Access NorthWind database, query and update it, and import data into tables using Prepared Statements;
- This application uses JFC Swing 1.1 API to implement the user interface, including some new features, such as, "Pluggable Look and Feel" and JTable;
- This application was first implemented at May 30th, 2000

Figure 4.30 Contents in About Menu Item

4.4.2 Execution of Application II

- Figure 4.31 shows the application's user interface, which is a web page designed by an HTML file. In this application, the IDS JDBC driver automatically pre-establishes the connection with the Northwind database when the application is first launched by the web browser.



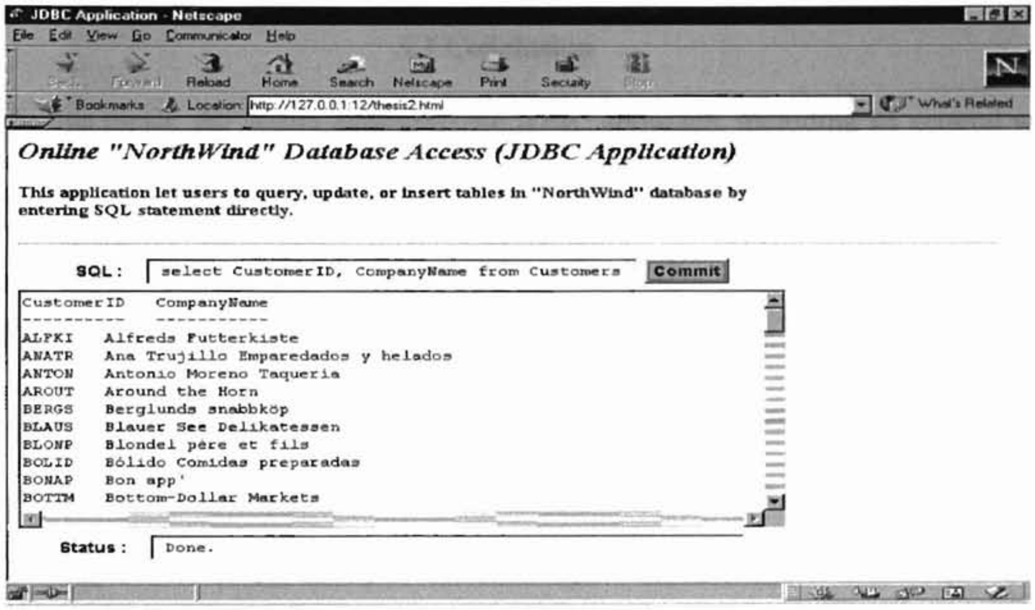
Online "NorthWind" Database Access (JDBC Application)

This application let users to query, update, or insert tables in "NorthWind" database by entering SQL statement directly.

```
"NorthWind" has been Connected
```

Figure 4.31 User Interface of the Application

This application provides users the abilities to query and update tables in Northwind by using SELECT, UPDATE or INSERT SQL statements. Figure 4.32 demonstrates an example of conditionally querying the Customer table.


JDBC Application - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://127.0.0.1:12/thesis2.html
What's Related

Online "NorthWind" Database Access (JDBC Application)

This application let users to query, update, or insert tables in "NorthWind" database by entering SQL statement directly.

SQL: select CustomerID, CompanyName from Customers   Commit

```
CustomerID   CompanyName
----------   -----------
ALFKI   Alfreds Futterkiste
ANATR   Ana Trujillo Emparedados y helados
ANTON   Antonio Moreno Taqueria
AROUT   Around the Horn
BERGS   Berglunds snabbköp
BLAUS   Blauer See Delikatessen
BLONP   Blondel père et fils
BOLID   Bólido Comidas preparadas
BONAP   Bon app'
BOTTM   Bottom-Dollar Markets
```

Status: Done.

Figure 4.32 a Run-time Query and the Results

CHAPTER V

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis, the author examined the JDBC approach, including its general work models, JDBC API features, and JFC API features. On this basis, the author designed and developed a general strategy for implementing the JDBC approach. This utilizes the three-tier client-server model over the Internet and uses JFC API for GUI design.

The work reported here indicates that, most of the advantages of JDBC come from its pure-Java API design and its ODBC-like features. The strategy presented for implementing the JDBC approach provides developers a very powerful and efficient way to design and develop fast and efficient enterprise database applications, and makes it easier and efficient for users to access databases through DBMSs from anywhere in the world. Advanced features of JFC API, such as lightweight Swing components, Pluggable Look and Feel, and Delegation Event Models features, make JFC API well suited to design and implement GUIs for JDBC-enabled Web database applications, providing more functionality, portability, and extensibility.

5.2 Future Work

To explore more JDBC features, the two applications can be extended along the following directions:

1. More advanced JDBC features and functionality can be added to the two

applications, such as implementing stored procedures and transaction management to strengthen them.

2. These two applications do not give any consideration for granting users role/privilege or for maintaining the concurrency and consistency of the database. The applications succeed with single-user desktop DBMSs like Microsoft Access. However, these advanced features are necessary to access full-scale, multi-user DBMSs such as Oracle.
3. Application I's GUI can be improved by using more Swing components.

REFERENCE

1 Ben-Natan, Ron. Objects on the Web, McGraw-Hill Companies, 1997.

2 Callahan, Tim. "So you want a Stand-alone Database." Java Developer's Journal, Vol. 3 (12), December, 1998.

3 Callaway, R. Dustin. Inside Servlets, Addison-Wesley Longman, Inc., 1999.

4 Carroll, Erin and Wilson, Andrew. "Database Programming with JDBC." Web Techniques, October, 1996.

5 Czernik, Thomas and Kamp, Rolf. "Self-contained Client Applets Using Swing." Java Developer's Journal, Vol. 5 (6), June, 2000.

6 Darby, Chod. "Applet and Servlet Communication." Java Developer's Journal, Vol. 3 (9), September, 1998.

7 Darby, Chod. "Migrating CGI Scripts to Java Servlets." Java Developer's Journal, Vol. 3 (1), January, 1998.

8 Darby, Chod. "Developing 3-tier Database Applications with Java Servlets." Java Developer's Journal, Vol. 3 (2), February, 1998.

9 Daniel, Minoli. Internet & Intranet Engineering, McGraw-Hill, Inc., 1997.

10 Date, C. J. An Introduction to Database System (Volume I), Addison-Wesley Publishing Company, Inc., 1990.

11 Davis, Judy. "Extended Relational DBMSs : The Technology, Part I." DBMS, June, 1996.

12 Elmasri, Ramez & Navathe, B. Shamkant. Fundamentals of Database Systems, the Benjamin/Cummings Publishing Company, Inc., 1989.

13 Flanagan, David, Fraley, Jim, Crawford, William and Magnusson, Kris. Java Enterprise In a Nutshell, O'Reilly & Associates, Inc., 1999.

14 Haecke, Van, Bernard. JDBC^{TM}: Java^{TM} database Connectivity, IDG Books Worldwide, Inc., 1997.

15 Hamilton, Graham, Cattell, Rick and Fisher, Maydene. JDBC^{TM} Database Access with Java, Addison-Wesley, 1997.

16 Heiser, Jay. "Java Security Mechanisms." Java Developer's Journal, Vol. 2 (3), March, 1997.

17 Horstmann, S. Cay and Cornell, Gary. Core Java 2 (Volume II - Advanced Features), Sun Microsystems, Inc., 2000.

18 IDS Server User's Guide, IDS Server Inc., February, 2000.

19 Karimi, Jahangir. "A software Design Technique for Client-server Applications." Concurrency Practice and Experience, Vol. 11 (1), January, 1999.

20 Lai, Siet-Leng and Lim, Joo Hwee. "Web Database Publishing." Java Developer's Journal, Vol. 2 (7), July, 1997.

21 Lang, Curt and Chow, Jeff. Database Publishing on the Web & Intranets, the Coriolis Group, Inc., 1996.

22 Ledru, Pascal. "Designing a Web Browser." Java Developer's Journal, Vol. 4 (4), April, 1999.

23 Levy, Elie. "Extending the AWT." Java Developer's Journal, Vol. 2 (7), July, 1997.

24 Levy, R. Michael. "Web Programming in Guide." Concurrency Practice and Experience, Vol. 28 (15), December, 1998.

25 Lewis, G. Ted. "Where is Client/Server Software Headed?" IEEE Computer, Vol. 28 (4), April, 1995.

26 Martinez-Campos, Fernando. "Bigger Than a Database." Database Programming and Design, April, 1997.

27 Matthew, D. Siple. The Complete Guide to Java Database Programming, McGraw-Hill, Inc., 1998.

28 McCarty, Bill. SQL Database Programming with Java, the Coriolis Group, Inc., 1998.

29 McClintock, Colleen. "Implementing Business Rules." Java Developer's Journal, Vol. 5 (7), July, 2000.

30 Nance, Barry. "Data Access Via ODBC and JDBC." Network Computing, January, 1997.

31 Neville, Patrick Sean. "Mastering Java Security Policies and Permissions." Java Developer's Journal, Vol. 5 (1), January, 2000.

32 Orfali, Robert and Harkey, Dan. Client/Server Programming with Java and CORBA, John Wiley & Sons, Inc., 1997.

33 Orfali, Robert, Harkey, Dan and Edwards, Jeri. The Essential Client/Server Survival Guide, John Wiley & Sons, Inc., 1996.

34 Piemont, Claudia. "Business Use Cases for Java." Java Developer's Journal, Vol. 2 (1), January, 1997.

35 Robertson, Bruce. "Driving Applications on the Network." Network Computing, January, 1996.

36 Rodley, John. Developing Database for the Web & Intranets, the Coriolis Group, Inc., 1997.

37 Roman, Ed. Mastering Enterprise JavaBeans™ and the Java™ 2 Platform, Enterprise Edition, John Wiley & Sons, Inc., 1999.

38 Sagar, Ajit. "Splitting Tiers." Java Developer's Journal, Vol. 4 (12), December, 1999.

39 Shah, Rawn. "Integrating Databases with Java via JDBC." JavaWorld, May, 1995.

40 Silberschatz, Abraham, Korth, F. Henry and Sudarshan, S. Database System Concepts, McGraw-Hill, Inc., 1997.

41 Spitzer, Tom. "Web Database Innovations." DBMS, August, 1997.

42 Tait, Barry. "Separating Presentation from Business Logic." Java Developer's Journal, Vol. 5 (7), July, 2000.

43 Thimbleby, Harold. "A Critique of Java." Software Practice and Experience, Vol. 29 (5), april, 1999.

44 Thompson, Charles. "A Scout's Guide to Three-tier Architecture." Database Programming and Design, August, 1997.

45 Venners, Bill. "Designing with Interfaces." JavaWorld, December, 1998.

46 Venugopal, Sesh. "Cross-database Portability with JDBC." Java Developer's Journal, Vol. 5 (1), January, 2000.

47 Viescas, L. John, Running Microsoft Access 97, Microsoft Press, 1997.

48 Whiting, Bill, Morgan, Bryan and Perkins, Jeff. Teach Yourself ODBC in 21 Days, SAMS Publishing, 1996.

49 White, Seth and Hapner, Mark. JDBC™ 2.0 API Specification, Sun Microsystems Inc., May, 1998.

50 Zielinski, Krzyszto. "Improving Scalability of Event-driven Distributed Objects Architectures." Software Practice and Experience, Vol. 30 (13), June, 2000.

2

VITA

Xinxue Yuan

Candidate for the Degree of

Master of Science

Thesis: BUILDING FAST AND EFFICIENT DATABASE APPLICATIONS FOR THE WEB

Major Field: Computer Science

Biographical:

Personal Data: Born in Beijing, China on July 29, 1967, the third son of Shixian Yuan and Shuying Zhao

Education: Graduated from the Civil Engineering Department of Beijing Polytechnic University (China) in June, 1990, and received the Bachelor's degree of Civil Engineering. Completed the requirements of the Master of Science at Oklahoma State University in December, 2000.

Professional Experience: Employed by Highway Planning and Design Institute, Department of Transportation, Beijing, China, as a Civil Engineer, 1990 to 1997; employed by United Airlines, IL, USA, as a Programmer Analyst, November 1999 to present