Database Technology and Database-Driven Web Applications

Fangxing Li
ABB Inc

ABSTRACT

Database technology involves the access and manipulation of information. It is critical to the development of highly efficient information systems. It also plays an important role in the development of web-based applications that require information processing and distribution between web browsers and web servers. This article provides a quick guide to database technology and illustrates common structures of database-driven web applications.

INTRODUCTION

Like many other information systems, utility information systems contain and process a large amount of data. Without the ability to manage that data efficiently, it is difficult for utilities to provide satisfactory services to customers. The development of information technology has answered this challenge. Databases and database management systems (DBMS) have been broadly deployed to manage bulk data in enterprise information systems.

Database

A database is a self-describing collection of data. The term "self-describing" implies that the database contains not only the actual data, but also the structure of the data (or the meta-data). [1-3]. A database may achieve high integrity because the meta-data typically describe the relationships among different tables. This feature of "self-describing" is

the main difference between a database and a flat file that was used in the early age of computing. A flat file does not contain any information about the structure and relationships among different pieces of data, and therefore is less integrated than a database.

DBMS

A DBMS is a software tool that helps users define, access, and maintain the underlying data contained within a database. As the definition shows, a database is the collection of structured data, and a DBMS is a software program that helps users manage the database efficiently. Figure 1 describes the logic interaction between a user, a DBMS, and a physical database.

There are many commercial DBMS products available from different vendors. Microsoft's Access is a popular example of a desktop DBMS. Microsoft's SQL Server is an example of an enterprise DBMS that works across a network for multiple users. Other popular DBMSs are IBM's DB2, Oracle's series of database management products, and Sybase's products.

DATABASE TABLES

Typically, a database is organized as a collection of multiple two-dimensional tables. Each entry in a table is single-valued. Each row in a table represents an instance of a real world object. Each column in a table represents a single piece of data of each row.

Figure 2 shows two tables in a database of a utility information system. Each row in the first table represents information about a different metering device. Each row in the second table represents the amount measured by a specific water meter at a given time.

Figure 1: Information flows among a user, a DBMS and a physical database

In the first table, there is a specific column ID, which uniquely identifies a metering device in the real world. This column is the primary key of this table. The primary key of a table may be a combination of several columns. For example, the primary key of the second table is the combination of ID, Measurement_Date, and Measurement_Time. That is, the remaining columns (in this case only one column – 'Gallons') are uniquely determined by the combination of the first three columns.

Metering_Equipment Table

ID	*Name*	*Utility*	*Facility*	*Manufacturer*
X1001	**GasA1**	**GAS**	**BLDG1**	**M101**
X1002	**ElecA1**	**ELECTRIC**	**BLDG1**	**M201**
X1003	**WaterA1**	**WATER**	**BLDG1**	**M301**
X1011	**GasB1**	**GAS**	**BLDG2**	**M101**
X1012	**ElecB1**	**ELECTRIC**	**BLDG2**	**M201**
X1013	**WaterB1**	**WATER**	**BLDG2**	**M301**

Water_Consumption Table

ID	*Measurement_Date*	*Measurement_Time*	*Gallons*
X1003	2/1/2003	12:00:00	341.23
X1003	2/1/2003	12:15:00	355.68
X1003	2/1/2003	12:30:00	362.42
X1003	2/1/2003	12:45:00	377.81

Figure 2: Two tables in a utility information system

RELATIONAL DATABASES

As the previous definition shows, a database contains structural information about the data as well as the actual data. How is the structure defined? Or, what is the structural model? The most popular model

over the past 20 years is the relational model. Other models include the hierarchy model and the network model, which existed for some time but did not gain considerable market share. There are also some emerging models like the object-oriented model and the object-relational model, both of which are gaining some market share. However, in general the relational model is still the dominant force. Hence, this article focuses on relational databases and related technologies.

What is a relational database? Theoretically, a relational database is a database comprised of a set of tables that follows the rules of *normalization*. The definition of normalization in database theory is complicated if expressed in a mathematical way. For simplicity, the primary principles of normalization can be roughly interpreted as the following guidelines:

1. When you design a table in a database, your table should avoid repeating groups.

2. The columns in a table depend on the primary key only.

3. There is no column depending on another column that is not part of the primary key.

The "repeating groups" problem is illustrated by the following example. A table is created to store all purchase orders, while each order may have different items. The first table in Figure 3 shows an un-normalized design, which can handle orders with no more than two items. If the maximum number of items is 30, then the columns must be expanded to contain 30 groups of item and quantity, i.e., from {Item1, Qty1} to {Item30, Qty30}. This could cause a serious waste of space if most of the other orders have less than 5 items.

The second table in Figure 3 shows a normalized design, which involves only four columns. The column Sub_Order_ID is used together with Order_ID to avoid the repeated grouping problem in the first table. In other words, if two rows have the same Order_ID, then the items in these two rows are associated with the same order. The column Sub_Order_ID can be used to identify different items within the order. With this design, there is no limit on the number of items within one order. Furthermore, there is no redundant information stored and efficiency is achieved.

Purchase_Order Table: Un-normalized Design

Order_ID	*Item1*	*Qty1*	*Item2*	*Qty2*
1	Circuit Breaker	4	Transformer	1
2	Sectionalizer	2	Distributed Generator	1

Purchase_Order Table: Normalized Design

Order_ID	*Sub_Order_ID*	*Item*	*Qty*
1	1	Circuit Breaker	4
1	2	Transformer	1
2	1	Sectionalizer	2
2	2	Distributed Generator	1

Figure 3: Two designs for a purchase order table

The term "depending on," mentioned in the above guidelines, can be interpreted as "being uniquely determined by." That is, if column A depends on column B, then the value of A is uniquely determined by the value of B, but not vice versa. For example, if we know the social security number (SSN) of a person, then we know his or her name, title, address, etc. Thus, the column of name, title, or address depends on the column SSN. However, if we know the name of a person, it is possible that we cannot identify his or her SSN, since people may have the same names. In the second table in Figure 3, the columns Item and Qty each depend on the combination of the columns Order_ID and Sub_Order_ID.

To achieve efficient database systems, the normalization rules or the above rough guidelines should be followed. Practical tests show that an un-normalized database may result in much poorer performance (5+ times slower) and need much more programming involvement. Although database designers may consider normalization by intuition without knowing it, they should be required to explicitly fol-

low the rules to ensure high performance and efficiency. The performance and efficiency issue is particularly important for web-based database-driven applications, since users of web-based applications may experience longer delays than users of stand-alone applications. The longer delays may be attributed to the following features of web applications:

- There may be many clients (users) concurrently accessing the server.
- The users may be geographically distributed across the country.

SQL

SQL is a standard to create, retrieve, and manipulate a relational database. Originally, SQL stood for the acronym of "structured query language," but now it has become generally accepted as a non-acronym standard to access the internal data of a database, usually a table-based relational database.

Unlike full-featured programming languages such as C/C++, VB, and Java, SQL is not a full-fledged programming language. It may be considered as a sub-language to create, retrieve, and manipulate information contained in databases. It can be dynamically coded into high-level languages like C/C++, Java, or VB to facilitate the control of the underlying databases.

SQL consists of a set of text-based commands or queries to control data. The SQL commands can be classified into two major categories: data definition language (DDL) and data manipulation language (DML). DDL is used to create tables or change table structures, while DML is used to insert, delete, or modify rows of tables. The DDL commands include the statements CREATE, ADD, DROP, ALTER, and others. The DML commands include statements of SELECT, INSERT, UPDATE, DELETE, JOIN, UNION, and others.

The following brief examples are given as a quick guide to explain how the SQL commands work. The examples are based on the needs of a utility information system administrator who wants to create a database table to host information, populate the table, manipulate the table, etc.

1. Create a table

The following command creates a blank table with the similar schema as the second table in Figure 2.

```
CREATE TABLE Water_Consumption(ID TEXT(20), Measurement_Date DATE, Measurement_Time TIME, Gallons DOUBLE)
```

The above CREATE command creates a table named Water_Consumption. The table has four columns called ID, Measurement_Date, Measurement_Time, and Gallons. The data types of these four columns are a text string of 20 characters, Date, Time and Double, respectively. The Date data type is usually input in the format of "MM/DD/YEAR" or "YEAR/MM/DD." The Time data type is usually input in the format of HOUR:MINUTE:SECOND with the HOUR filed using a 24-hour clock. For example, "18:45:00" is input for 6pm, 45 minutes and 0 seconds.

2. Populate a table

The following command adds a row into the table Water_Consumption. It should be noted that a text column is enclosed by opening and closing quotes (single or double). Columns in Date or Time data type should be enclosed in quotes as well.

```
INSERT INTO Water_Consumption VALUES ('X1003', '1/1/2003', '18:45:00', 165.82)
```

To add many rows into the table, users may use the command in the format like "INSERT INTO target SELECT … FROM source," in which the "SELECT…FROM" statement will be mentioned next.

3. Select data

The most popular command used in SQL is probably the SELECT statement. The following command selects all rows and all columns from a table.

```
SELECT * FROM Water_Consumption
```

The * represents all columns in the selected table. Users may select

partial columns by specifying the actual column names. For example, the following SQL command selects all rows but only 'ID' and 'Gallons' columns.

```
SELECT ID, Gallons FROM Water_Consumption
```

There are also various clauses that can be appended after the above SELECT statements to filter some rows. For example, the following command selects the information only related to Meter X1003 using WHERE clause.

```
SELECT ID, Gallons FROM Water_Consumption WHERE ID='X1003'
```

4. Delete data

The following command deletes all rows from the table Water_Consumption.

```
DELETE * FROM Water_Consumption
```

The WHERE clause can be used as a filter for DELETE statement. The following command deletes the rows from meter X1003 and with a date no later than 12/31/2001.

```
DELETE * FROM Water_Consumption WHERE ID='X1003' AND Measurement_Date<'01/01/2002'
```

Since this article is not a detailed SQL guide, the above examples do not cover all aspects of SQL commands. For details about SQL, users may check references 1 and 2. Despite its simplicity, this section is expected to serve as a quick start for further SQL studies.

SQL CODED IN OTHER PROGRAMMING LANGUAGES

Although SQL is a powerful tool specifically designed for database access, it is not a full-featured programming language. Hence, to maximize the benefit of SQL in applications, embedded SQL is used. That is, SQL is coded into a programming language like C/C++, VB, or Java in

a database-driven application. The programming language is employed to perform the common "programming" tasks while the embedded SQL queries are utilized to access the database. This interactive process can be described as follows:

1. The application creates a connection to a database so that the host application can "talk" with the database and its tables.
2. The application generates an SQL query to obtain a table in the database.
3. The content in the table is retrieved and then mapped to the internal data structure of the application.
4. Operations on the data are carried out. This could be very simple or complicated, depending on the application's requirement.
5. The updated data may be saved back into the database and output may be generated.

The VB code shown in Figure 4 illustrates a sample process of the five steps above, including how to set up a database connection, retrieve data from a database, identify rows with gallon amount over a predefined threshold, and generate a warning report. The report file contains all metering records with a gallon amount over the threshold.

In short, SQL queries could not do much but access the database. Programming languages are more flexible and powerful in many other functions, but not in direct database access. Hence, the combination of these two is a good choice to create fast, efficient, and easy-to-program applications involving underlying databases.

DATA ACCESS INTERFACES

After reading the previous example, readers may have this question: "How does the database receive the SQL query, interpret it, and then send the response back to the VB application?" To answer this question, the mechanism of database access interfaces is explained.

Database access interfaces are software modules that provide connections between an application and a specific database. They play a key role in implementing database-driven applications. Different vendors may have different database drivers. At times, this could cause portability and extensibility problems, since users may have to deal with differ-

```
Dim dbs As Database, rst As Recordset
Dim OutputFileName As String
Dim k As Long
Dim TheID As Long, TheDate As String, TheTime As String, TheGallons As Double

'Connect to a database
Set dbs = OpenDatabase("MyTestEIS.mdb")
'Create a SQL query to obtain       the database table Water_Consumption
Set rst = dbs.OpenRecordset("SELECT * FROM Water_Consumption")
If rst.RecordCount = 0 Then
       MsgBox "No record in the table."
       Exit Sub
End If

OutputFileName = "OutputTest.csv" 'Output to a CSV spreadsheet file
Write #1,   "ID", "DATE", "TIME", "GALLONS" 'Output a header

Open OutputFileName For Output As #1 'Open an output report file
For k = 1 To rst.RecordCount
       'Retrieve the field and store it in VB's internal data structures
       TheID = rst("ID")
       TheDate = CStr(rst("Measurem      ent_Date")) 'Get date in string format
       TheTime = CStr(rst("Measurement_Time")) 'Get time in string format
       TheGallons = rst("Gallons")

       'Perform operations on the extracted data. Here, GetThresholdFor()
       'is a function to obtain the warning threshold of a m           etering device
       Threshold = GetThresholdFor( TheID )

       Write #1, TheID, TheDate, TheTime, TheGallons
       rst.MoveNext
Next k
Close #1

'Close the database connection
rst.Close
dbs.Close
```

Figure 4: An example of VB and SQL queries

ent vendors and even different platforms. An early solution to this problem was the ODBC (open database connectivity) [4] technique provided by Microsoft. The ODBC module sits between applications and vendor-specific databases to provide the necessary connectivities.

Microsoft has recently replaced ODBC with universal data access (UDA), which provides access to all kinds of data sources like ODBC databases, traditional SQL data, non-SQL data like spreadsheets, etc. UDA is the database access part of Microsoft's component object model (COM), which is an overall framework for creating and distributing object-oriented programs in a network. UDA consists mainly of the high-level application program interface (API) called activeX data ob-

jects (ADO) and the lower-level services called OLE DB. SQL queries are sent to ADO interfaces from applications and then forwarded to OLE-DB interfaces. OLE-DB communicates with vendor-specific data providers to retrieve information from physical databases. The retrieved information is then sent back to the applications. This database access strategy is shown in Figure 5.

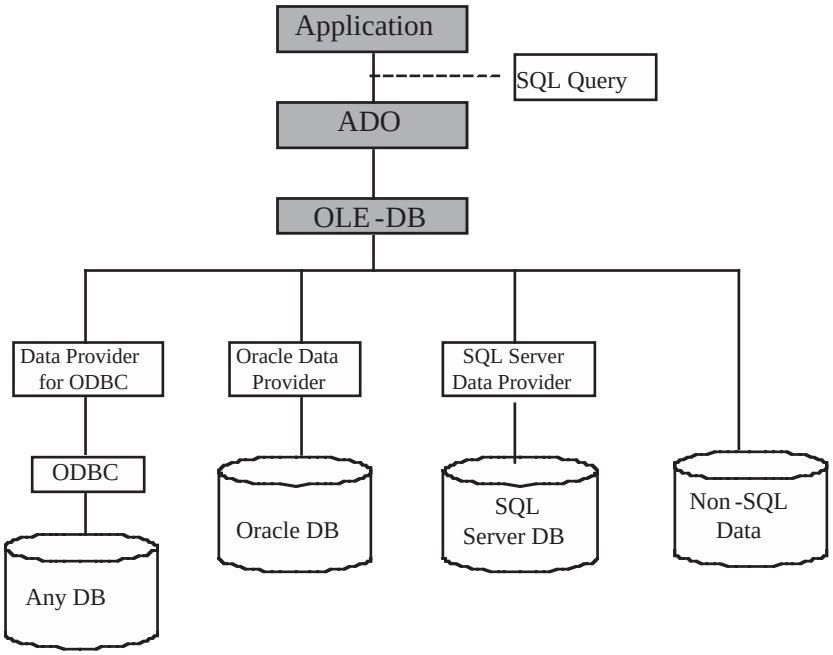

Figure 5: Architecture of Microsoft's UDA

Sun Microsystems presents another data access technology, Java database connectivity (JDBC) [5], which is an API that can be used to access almost any tabular data source from the Java programming language. The JDBC interface provides cross-DBMS connectivity to a wide range of SQL databases. The latest JDBC API also provides access to other tabular data sources, such as spreadsheets.

JDBC architecture contains a driver manager and database-specific drivers to provide transparent connectivity to databases from different vendors. This is shown in Figure 6. A JDBC driver translates standard JDBC calls into a protocol that the underlying database can understand.

This translation function makes JDBC applications able to access many different databases. There are four distinct types of JDBC drivers. Details about the four drivers and their mechanisms are beyond this article, but can be found in reference 5.

An interesting and noteworthy point is that Sun's JDBC technology supports multiple operating systems but is restricted to the Java programming language. As opposed to JDBC, Microsoft's UDA technology is restricted to the Windows platforms but supports multiple languages like VB, C/C++, J++, and the latest .NET technology.

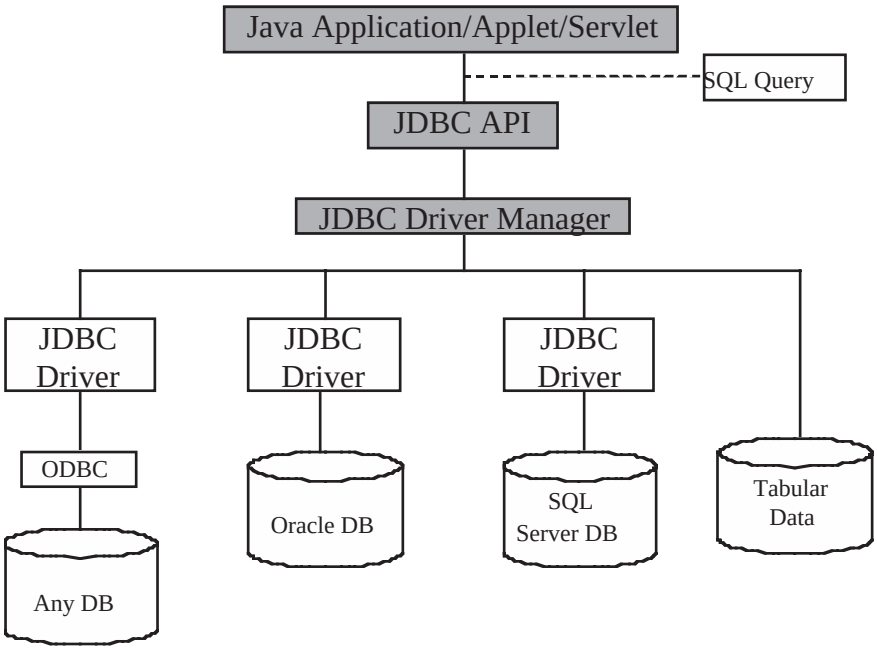

Figure 6: Architecture of Sun's JDBC technology

Although a thorough discussion of the technologies on database access interfaces is beyond the scope of this article, it is helpful to readers to understand the basic concepts. With the assistance of these interfaces, applications can essentially "talk" with physical databases and efficiently retrieve the information contained in various databases. Also, database access interfaces like ADO and JDBC can minimize application

developers' efforts, because developers only need to deal with the ADO or JDBC rather than do the detailed work that the interfaces perform.

DATABASES AND WEB APPLICATIONS

Like stand-alone applications, web applications [6-8] may rely on database technologies for efficiency. The architecture of a database-driven web application is illustrated in Figure 7. The information flows and activities are also illustrated with the arrow-lines. Here is the description of the activities in the figure.

1. A client browser sends an HTTP request to a web server for a specific web page that may contain regular HTML code as well as code written in server-side script (SSS).
2. The web server receives the request. If the requested web page involves SSS code, the web server invokes a SSS engine to process the SSS code.
3. If the SSS code involves database operations, the SSS engine queries the database through database interfaces to obtain the needed results, and may generate part of the returning HTML code based on query results.
4. The web server creates a response HTML page that is a combination of the returning HTML code from the SSS engine and the regular HTML code in the original web page.
5. The response HTML page is sent back to the client browser and the browser displays it for users.

The server-side script (SSS) is employed to generate dynamic web pages, which may require database manipulations. Since HTML is a markup language designed mainly for information display, it cannot handle complicated computation and database access. To make a web server more powerful, some scripts are usually embedded into an HTML page to direct the web server to perform some specific tasks. The web server invokes the SSS engine to handle complicated tasks like database access. The SSS engine passes the database queries to the database interface/driver that communicates with the physical databases.

Typically, the associated database interface/driver is located at the server side. This makes many database-related operations at the server-

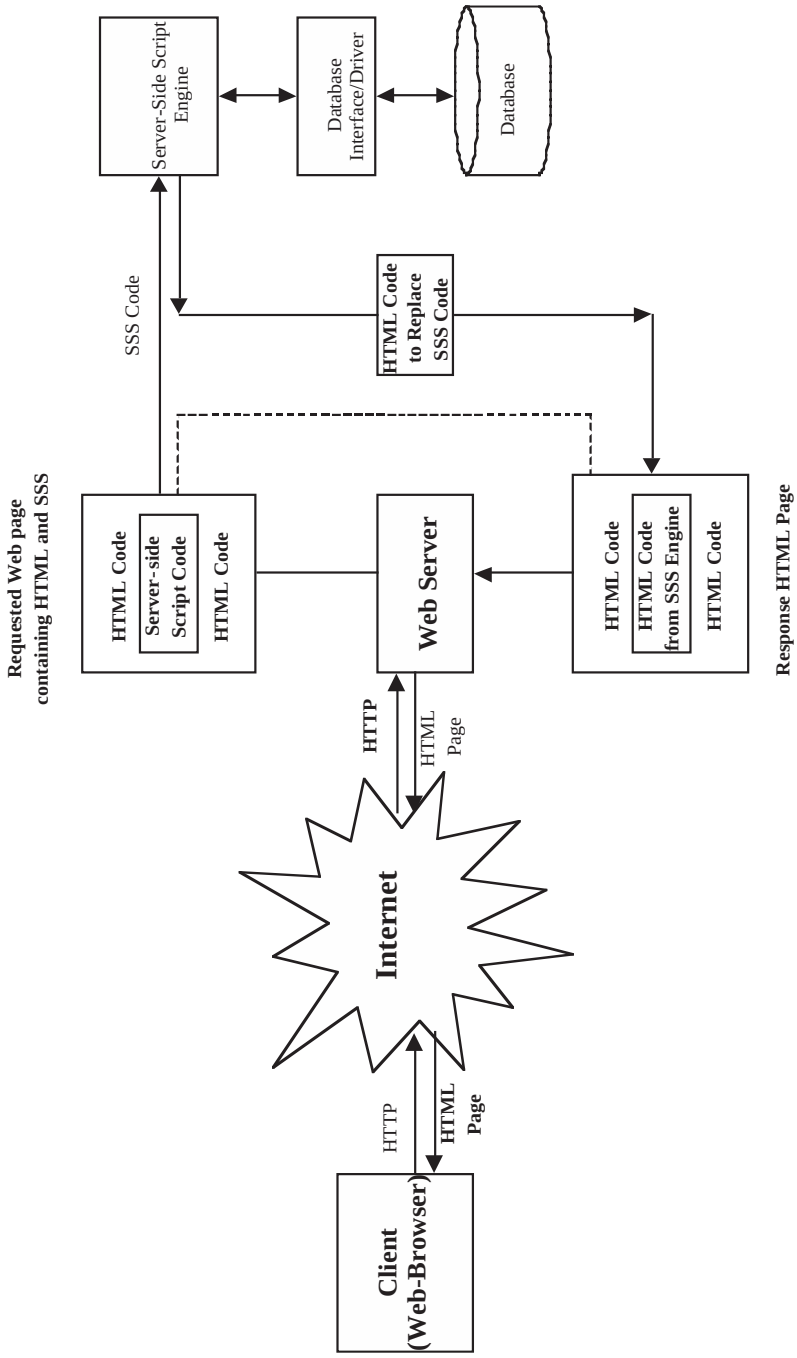

Figure 7: Generic architecture of a database-driven Web-based application

side transparent to the client side. This is an advantage of web-based applications, because users do not need to worry about the complicated database access and set-up processes. Once everything at the server side is set up, users from anyplace with Internet access can benefit from the web application.

SERVER-SIDE SCRIPT TECHNOLOGIES FOR DATABASE-DRIVEN WEB APPLICATIONS

There are several server-side script technologies that can assist developers in implementing database-driven web applications. Two of them, Active Server Pages (ASP) and FoxWeb, are briefly reviewed in this section.

ASP

ASP is Microsoft's technology for building dynamic and interactive web pages. The overall architecture of ASP-centered web applications is similar to the generic architecture depicted in Figure 7. The main difference is that the so-called ASP script engine replaces the SSS engine in Figure 7. The ASP script engine can handle various requests including intensive database manipulations. The requests are coded in ASP scripts that are usually written in the VB script language. An ASP web page is a text file with the extension of *.asp that contains HTML code and ASP scripts.

FoxWeb

FoxWeb is another technology that enables developers to create dynamic web pages, especially if the pages involve underlying FoxPro databases. The overall architecture of FoxWeb-centered web applications is also similar to that depicted in Figure 7. The FoxWeb script engine is a specific SSS engine. Like the ASP script engine, the FoxWeb script engine can handle complicated database manipulations, especially for FoxPro databases. This makes FoxWeb very attractive to developers who need to convert legacy desktop applications powered by FoxPro databases to web-based, FoxPro-driven applications. Similar to an ASP web page, a FoxWeb web page is essentially a text file with extension of *.fwx that contains HTML code as well as FoxWeb scripts. Also, the latest FoxWeb scripting object is compatible with Microsoft's Active Server

Pages (ASP) objects. This makes it easier for developers who are already familiar with ASP to become familiar with FoxWeb.

There are also other similar server-side technologies such as JavaServlet, JSP, PHP, WestWind, etc., which use mechanisms similar to (but implementations different from) ASP or FoxWeb technologies to process server-side tasks. Since all of the above technologies are developed to carry out server-side tasks including database access, those technologies, together with database technologies, are the driving force of the evolution of web applications.

CONCLUSION

Like many other information systems, utility information systems usually employ database technology to store and retrieve data to achieve high performance and efficiency. Database technology is especially important to web-based information applications since a large amount of data needs to be processed at the server-side and distributed to geographically remote clients. As a quick tutorial and guide, this article reviews the basics of database technology such as relational databases, SQL, and database access interfaces. The article also provides an illustration about common architecture of database-driven web applications.

Acknowledgement

The author would like to thank Mr. David Green and Dr. Barney L. Capehart for their valuable comments and suggestions. The author would also like to thank Ms. Lynne Capehart for her careful editing and formatting of the final document.

References

[1] Raghu Ramakrishnan, *Database Management Systems*, McGraw-Hill, 1997.

[2] Jesse Feiler, *Database-Driven Web Sites*, Morgan Kaufmann Publishers, Inc., 1999.

[3] Paul Dorsey and Joseph. R. Hudicka, *Oracle 8 - Design Using UML Object*, Oracle Press, 1999.

[4] Kyle Geiger, *Inside ODBC*, Microsoft Press, 1995.

[5] Cay Horstmann and Gary Cornell, *Core Java*, vol. 2, The Sun

Microsystems Press, 2001.

[6] Chris Ullman, et al, *Beginning ASP 3.0*, Wrox Press Ltd., January 2000.

[7] Fangxing Li, Lavelle A.A. Freeman, Richard E. Brown, "Web-Enable Applications for Outsourced Computing," *IEEE Power and Energy Magazine*, vol. 1, no. 1, (Premier issue) January-February 2003.

[8] Dustin R. Callaway, *Inside Servlets: Server-Side Programming for the Java™ Platform*, Second Edition, Addison Wesley Professional, May 2001.

ABOUT THE AUTHOR

Fangxing Li is presently a senior R&D engineer at ABB Inc. He received his B.S. and M.S. degrees both in electric power engineering from Southeastern University, China, in 1994 and 1997 respectively. He received his Ph.D. degree in computer engineering from Virginia Tech in 2001. His main experience includes development of several power industry applications, such as Electric Power Research Institute (EPRI)'s Distribution Engineering Workstation (DEW) for power distribution planning and analysis, ABB's Power Distribution Optimizer (PDO) for Web-based distribution reliability planning, and ABB's GridView for energy market simulation. Dr. Li is a member of IEEE and Sigma Xi. He can be reached at fangxing.li@us.abb.com or fangxing.li@ieee.org.