Wiley Encyclopedia of Electrical and Electronics Engineering

Volume 18

John G. Webster, Editor
Department of Electrical and Computer Engineering
University of Wisconsin–Madison

A Wiley-Interscience Publication
John Wiley & Sons, Inc.
New York · Chichester · Weinheim · Brisbane · Singapore · Toronto

theorems in point stochastic process theory. The Renyi–Kninchine–Ososkov theorem (8–10) states that the procedure of "thinning" any point process (under a suitable normalization condition) asymptotically leads to a Poisson process. The Grigelionis–Pogozhev theorem (11,12) states that a superposition of point stochastic processes (under some not-so-restrictive conditions) also results in a Poisson process asymptotically. The first theorem is effectively used for reliability analysis of highly reliable redundant systems where system failures are "rare events". The second one is a background for the use of a Poisson process for the description of failure process of multicomponent renewable series systems. Discussion on application of these approaches to reliability theory can be found in Ref. 4. We also find heuristic methods for analyzing renewable systems there.

For a fairly complex renewable systems like communication network, analytical results are difficult to obtain. Monte Carlo simulation can be recommended (5,13) for such systems. For highly reliable systems, whose Monte Carlo modeling takes too much computer time and demands huge computer memory, accelerated methods of modeling have been developed (4) by various authors.

DATA COLLECTION

We need to pay careful attention to collection and analysis of field data. Effective reliability analysis of a repairable system demands developed engineering intuition and experience owing to many details. This is the proven way to move from reliability theory to real-world engineering applications. We remind the readers of the GIGO (Garbage In Garbage Out) principle in a jocular vein!

There are several ways of reliability data collection. Because the availability coefficient of a repairable system is one of the main reliability indices, statistical data collection begin as special tests to confirm the required availability level. Tracking the history of each individual failure is important because system maintenance (spare supply, operation monitoring, preventive maintenance, etc.) are based on current reliability data. For these purposes, reliability data must be supplied with all relevant information: environmental condition at the time of failure, level of loading, regime of its use (hot or cold), and so on. Individual failure report should then be consolidated to obtain statistical summary, which might be used for reliability analysis of newly designed system of a similar type.

There are two main ways to collect reliability data. First is recording the failure history for each type of unit (time between current and previous failure plus additional related information). Statistical inference based on the unit data gives objective information about the units of this type. This information is usually collected by unit vendors. Another type is tracking each repairable unit: from warehouse (as a spare) to installation, then to failure, repair and back to sparing or installation. This records the individual unit behavior placed in a particular set of circumstances and is useful for recognizing possible weak points in the system.

Data on mean time to repair may be obtained from special control experiments or from real usage. The MTTR value is often assigned to a unit on the basis of previous engineering experience.

Formal reliability analysis of a complex system is only as useful as the model and the approximations used. If the ultimate goal of reliability analysis of a repairable system is to ensure some overall reliability threshold for the minimum cost, we need to understand which components are more important in the reliability sense. Sometimes increased component level reliability is more effective than a subsystem redundancy (3). Given a concrete set of objective functions, ingenious analysis and judicious use of redundancy can deliver a reliable (fault tolerant) system with inexpensive and less reliable components. In our opinion, RAID (Redundant Array of Inexpensive Disks) is one such example (14).

BIBLIOGRAPHY

1. H. Ascher and H. Feingold, Repairable system reliability: Modeling, inference, misconceptions and their causes, in D. B. Owen (ed.), *Lecture Notes in Statistics,* Vol. 7, New York: Marcel Dekker, 1984.
2. B. Gnedenko, Yu. Belyaev, and A. Solovyev, *Mathematical Methods in Reliability Theory,* New York: Academic Press, 1969.
3. E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing: Probability Models,* New York: Holt, Rinehart and Winston, 1975.
4. B. Gnedenko and I. Ushakov, in J. Falk (ed.), *Probabilistic Reliability Engineering,* New York: Wiley, 1995.
5. I. Ushakov (ed.), *Handbook of Reliability Engineering,* New York: Wiley, 1994.
6. D. R. Cox, *Renewal Theory,* Methuen Monographs on Applied Probability and Statistics, London: Methuen, 1962.
7. W. L. Smith, Renewal theory and its ramifications, *J. Roy. Stat. Soc. Ser. B,* **20** (2): 243–302, 1958.
8. A. Renyi, Poisson-folyamat egy jemllem zese., *Proc. Inst. Math., Hungarian Academy of Sciences,* **1** (4): 11–24, 1956.
9. A. Ya. Khinchin, On Poisson stream of random events, *Theory Probab. Appl.,* **1**: 1–23, 1956.
10. A. Ososkhov, A limit theorem for flows of similar events, *Theory Probab. Appl.,* **1** (2): 89–101, 1956.
11. B. Grigelionis, On the convergence of sums of step stochastic processes to a Poisson process, *Theory Probab. Appl.,* **8** (2): 27–31, 1963.
12. B. Progozhev, Evaluation of deviation of the equipment failure flow from a Poisson process, in B. N. Bruevich and B. Gnedenko (eds.), *Cybernetics in the Service of Communism,* Vol. 2 (in Russian), Moscow: Energia, 1964.
13. B. Gnedenko, I. Pavlov, and I. Ushakov, in S. Chakravarty (ed.), *Statistical Reliability Engineering,* New York: Wiley (forthcoming).
14. D. Patterson, G. Gibson, and R. Katz, *A Case for Redundant Arrays of Inexpensive Disks (RAID),* University of Berkeley Report no. UCB/CSD 87/391.

IGOR USHAKOV
SUMANTRA CHAKRAVARTY
QUALCOMM, Inc.

REPORT GENERATOR

Report generators are tools that provide an effective way to present data retrieved from databases in a customizable and printable format. Most of the information in a report comes from an underlying table, query, or structured query lan-

guage (SQL) statement, which is the source of the report's data. Other information in the report is stored in the report's design. Users can manipulate the data, perform necessary mathematical calculations, create charts, and more. Because one has control over the size and appearance of everything on a report, an end user can display the information the way he or she wants to see it. The trend today is to generate reports via Web browsers, and many businesses and information providers have regarded the Web as the most powerful and extensible solution for providing up-to-date information for their clients. One way to do this is to link the information power of the database to the Web and to provide a tool to manage all the information needs (1). Through a Web-based report generator, one can have access to the database in customizable reporting capabilities and to every field in the database. In addition, because the user can have total access to the database, he or she can even make custom database modifications using the report generator. This article reviews the state of the art of database report generators, particularly the current trend of the Web-based report generators. The article first surveys the field of report generators and discusses the emerging trend of the Web-based reporting tools. Then it provides some definitions and concepts related to our topic and discusses the principles of Web-database interaction to report generators. It finally presents a case study of three report-generating tools developed at the High Performance Database Research Center (HPDRC) at Florida International University.

The case study later in this article presents three techniques and tools that provide database connectivity for HTTP Web servers running on Unix and Windows. Each tool offers different ways to construct an SQL query, extract data from a database, and generate Hypertext Markup Language (HTML) pages to produce interactive and real-time Web reports. Moreover, each tool follows a different strategy and has certain characteristics. The first tool, *WebRG,* allows database integrators to develop Web forms and reports for any open database connectivity (ODBC)-compliant database (most relational databases as well as Web-ODBC semantic database management systems [DBMS]). This tool merges HTML documents with database functions to create a powerful dynamic access to databases using designer-defined macro files. End users can then query the database through a series of predefined forms and reports provided by the application developer. Thus, users can easily publish data from their databases in the form of Web-enabled reports. The second tool, *Web-SQL,* is most effective for users who are familiar with SQL. Users can edit an SQL query, process it, and retrieve results in a tabular format at runtime. These tools are also useful for batch scripts, for production of printed reports, and for data import/export and post processing. The third tool, *Sem-Access*, allows end users to have automatically generated forms and standard or customizable reports derived from the conceptual schema of Sem-ODB. This tool is generic in the sense that it provides a simple and effective method to retrieve and manipulate the semantic database and generate reports without requiring predefined forms, and to define quickly report content and format. At runtime, the Sem-Access extracts data from the semantic database and generates HTML pages to produce interactive and real-time Web reports. This application is built implicitly for the semantic object database management system to generate reports and to modify the data of the database. Furthermore, the end user can extract information about the schema of the database itself. This can be enhanced to include whatever information end users and database developers need to know about the database. This information is provided through a sequence of the user's interaction with the information provided though the Web browser.

The objective of the aforementioned tools is to provide the user with easy access to databases and to generate forms and reports across all platforms without requiring the developer to code a complex application for the task at hand. This increases the speed of development. In addition, using the Web browsers, the end user can retrieve data from databases directly whenever needed and do any necessary modifications and manipulations of data using the Web-formatted forms and reports, platform independently, and remotely, without having to open a different application or learn to use anything but the Web browser.

Other features of these form and report generators are as follows:

1. Provide automatic SQL generation capabilities.
2. Allow the operator manually to edit and prepare SQL statements.
3. Provide access to a large variation of databases by the use of the ODBC protocol.
4. Allow the operator to prepare, store, maintain, and modify report templates.
5. Allow the operator to generate reports from the Web on the fly.
6. Allow the operator to update database reports.
7. Allow the operator to generate multiple queries in a single report.
8. Provide an authentication against the database from the Web and present the operator with his or her view of the database schema.
9. Allow import/export and postprocessing of ASCII data.
10. Produce printing-friendly HTML and allow batch printing of reports.
11. Provide an interactive ad hoc report generator.
12. The generators are Web-browser-independent (i.e., can be run with any high-graphic or low-graphic browser).

REVIEW OF REPORT GENERATORS

Linking a database to the Web to generate Web-formatted forms and reports presents a challenge to many companies. The challenge is to make the data available on the Web efficiently and reliably and to provide the user with the ability and flexibility to create, modify, and generate forms and reports using a Web browser dynamically. Another challenge is to adopt a Web Interface language that would make data available on the fly cheaply and easily in the form of Web-enabled forms and reports. A number of publications have addressed different strategies to connect the database to the Web and presented different ingredients of the reporting tool. In Ref. 2, the author discusses a number products for connecting the database to the Web, some of which offer greater scalability since multiple connections can be run to the database instead of just one as in the case when the Web server is linked directly. For instance, IBM offers the Net.data Web application, a tool to generate reports from the Web. This tool

is invoked by using Common Gateway Interface (CGI), Netscape API (NSAPI) from Netscape communications, and Internet Server API (ISAPI) from Microsoft Corp.

Report generators are designed taking into consideration the database in question and the benefits and supported features they can provide. We will list some of the report generators that were developed by different companies and we will discuss the functionalities and other related features of these report generators. FlexQL (DataFlex Corp.), for instance, is a relational database writer that can produce reports from DataFlex, Paradox (Borland Inc.), dBase (Ashton-Tate Corp.), Lotus 1-2-3 (Lotus Development Corp.), and Btrieve (Pervasive Software Inc.). The generated reports can be viewed immediately as they are created and modified. In addition, FlexQL combines data from different file formats within a single report. It utilizes the power of SQL with report definitions automatically translated into American National Standards Institute (ANSI) standard SQL scripts. DB-Tech Inc. has also developed a reporting tool that can bring reports directly to the Web and allow the user to interact with the reports. DBPower is another report writer that gives users the advantage of a friendly user graphical interface (GUI) for creating presentation quality reports. These reports can include images, multicolumn text, and business graphics. DBPower transforms the details of database structure into a graphical representation of data called views. Based on the view, a report layout can be built by placing report objects in a report page. Report objects can be labels, database columns and expressions, runtime parameters, images, tables, and charts.

Another reporting tool, PLAS (product line asset support), provides the command center operator the capability to perform database queries and extract information from the database in a tabular format that can be used to develop reports and documents. The IQ/LiveWeb and IQ/SmartServer, for IQ Software, provide query processing, such as searching, sorting, data manipulation, output processing, and production reporting, and allow complex and repetitive queries or reports to be created on the desktop. In addition, they allow users to

1. Query data from heterogeneous data sources.
2. Combine charts, multidimensional crosstabs, and free from layouts on a single page.
3. Use parent/child reports to pass results from one query object to another.
4. Add watermarks, bitmaps, and audio and video clips.

Informix Inc. offers Web-DB Publisher, a report generator that lets users easily publish data from their databases in the form of Web-enabled reports. Web-DB Publisher lets users extract data from the Informix database and generate HTML pages to produce interactive reports. This is accomplished through the execution of embedded SQL statements directly within the HTML pages, which retrieves data dynamically from the Dynamic Server database. Furthermore, it provides a set of utilities for database management purposes, such as creating tables, views, and stored procedures. Additionally, an SQL scheduler is included to automate the execution of recurring tasks such as report generation.

EasyReporter, by Speedware Corp., is a tool that allows nontechnical end users to create and run reports. Reports can be generated from the midrange system and sent straight to the printer, or downloaded to PCs for more processing with popular PC packages such as Lotus 1-2-3, dBase, Microsoft Excel, or Microsoft Word. EasyReporter also creates sophisticated reports using calculations. EasyReporter's data retrieval methods are transparent. End users do not need to know the database file structures' setup. All they need to know is what information they want and how they want their report to look.

MS Access, by Microsoft, is a relational database that contains tools to design graphical forms and reports. Users can use SQL to query, update, and manage the interaction with MS Access database and other relational databases by the use of the ODBC. When a user creates a query in query Design view, behind the scenes Microsoft Access constructs the equivalent SQL statements. The user can view or edit the SQL statement in SQL view.

RepGen is a powerful report generator that can access and combine information from a relational database. It allows users to incorporate data from different modules into simple "column" reports or more complex "panel" reports. A report type, "grid" reports, allows data to be analyzed numerically. A graphical "field navigator" allows the user to locate the information required from the various data tables, while powerful sorting and querying functions allow complete and subtle control of the final output. To facilitate the production of commonly needed lists, RepGen is supplied with a library of over 50 predefined reports. These can be edited or modified according to the user's own needs. In addition, the user can use elements of these reports, such as sort orders or queries, as components of his or her own report definitions. RepGen provides an exceptionally powerful tool for interrogating a database.

The report generator, REPORT, is a comprehensive tool for preparing a wide variety of printed reports. REPORT contains the tools for preparing simple lists, multilevel hierarchical subtotals, form letters, bills and checks on preprinted forms, cross tabulations, and schedules. REPORT contains facilities for record selection, sorting, computation, table lookup, record linkage between files, and complex procedural logic. REPORT reads the report instruction file (REP) and produces a report according to the instructions found there. There are two modes of formatting usable in REPORT: namely, automatic format and explicit format. Automatic format mode is appropriate for detailed listings in columnar format and/or summary reports. Explicit formatting, on the other hand, is used when the person writing the report needs to specify the exact output formats.

To support different databases, Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) have been used to permit a Web server to pass data to any SQL database. Obviously, the only problem with this methodology is that the ODBC has to be translated to the native language of the database. However, ODBC provides a uniform access to heterogeneous databases. For instance, Aspect Software Engineering has released dbWeb 1.0, which is a software tool that provides database connectivity for HTTP Web servers running on Microsoft Windows NT. Essentially, dbWeb is a gateway between ODBC data sources and Web servers, such as Purveyor and WebSite. DbWeb offers full insert, delete, and update capabilities.

CONCEPTS AND DEFINITIONS

To detail the Web/database interaction to generate reports through the Web, it is first necessary to define and explain

briefly the key concepts that are used in this article. These key concepts help set the stage for the proceeding discussions.

Hypertext Transfer Protocol (HTTP). The protocol that is used by Web servers. Client programs that can speak HTTP are known as browsers. Web browsers are used to connect to HTTP servers and to view or retrieve information.

Hypertext Markup Language (HTML). The formatting language used with the Web. It defines how authors can format information that will be presented on the Web. For example, if the user wants to write the phrase "the database and the Web integration" in the center, he or she needs to enclose it within two tags: ⟨center⟩ the database and the Web integration ⟨center⟩.

Uniform Resource Locator: (URL). Provides the ability to specify addresses for particular objects on the Web.

Common Gateway Interface (CGI). Has some limitations; the main one is its inability to present dynamic information on the Web. Therefore, we needed a way to keep dynamically changing information current. This required the invention of the Common Gateway Interface and Java. A CGI program directs its output to an HTTP client. In other words, it dynamically generates HTML code.

Structured Query Language (SQL). Provides a way to access and manipulate data within a database. It selects columns and rows in a tubular format that match certain criteria.

Open Database Connectivity (ODBC). Provides a standard interface with heterogeneous DBMSs. The ODBC interface allows applications to access data in database management systems using SQL as a standard for accessing data. Thus, a single application can access different DBMS without targeting a specific DBMS. Users can then add modules, called database drivers, which link an application to their choice of DBMS.

REPORT GENERATOR: SYSTEM ARCHITECTURE

On the Inner Design of a Form and Report Generator

In this article, we reflect the design strategies of three reports generating tools originally developed in conjunction with the semantic database project, Sem-ODB. Two of the tools, WebSQL and WebRG, interact with the databases via ODBC-SQL protocol and, thus, work with any ODBC-compliant database. Another tool, Sem-Access, is specific for the semantic database as it builds ad hoc SQL queries while guiding the user through the rich semantic structure of the database schema (3). The tools can be viewed at http://hpdrc.cs.fiu.edu/demos.

The reporting tools access the semantic database in two different ways:

1. We implemented ODBC Application Programming Interface (API) functions that handle the operations on the semantic database.
2. We implemented semantic API functions to handle the interaction and the data access to the objects in the semantic database. The Semantic API functions is then the layer that is used as a gateway from the form generator to the semantic database, as shown in Fig. 1.

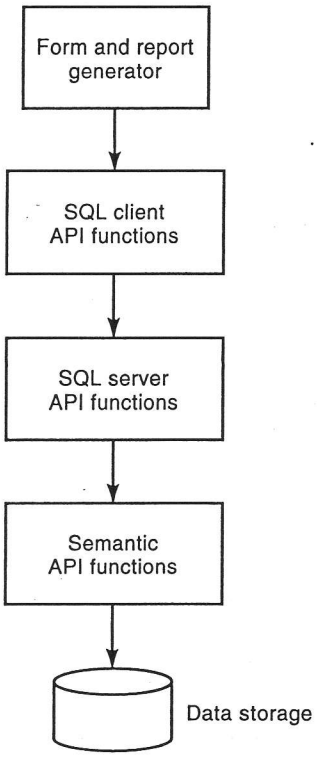

Figure 1. A representation of the layers that interface with the semantic database.

In Figure 1, the form and report generator is used as an interface tool to build and produce forms and reports as well as send requests to a remote SQL server. The SQL server accesses data from databases that are distributed along the Wide Area Network (WAN). Developers make calls to the SQL client standard API functions to communicate with the remote SQL server. The client portion contains enough functionality to retrieve the database schema and construct or submit SQL statements. The SQL server processes clients' requests by making direct calls to the semantic database API functions, converts the standard ODBC SQL query to the equivalent semantic API functions, and sends the results back to the client.

Architecture of the Front End of Database Engines

The front end of semantic database is divided into four layers:

User Interface Layer. This layer defines the user interface for our front end. It includes the following three components:

1. *Form generator.* Allows users to create, design, and modify forms.
2. *Report generator.* Gives the user the ability to create, modify, and print reports.
3. *Query.* Provides user the ability to generate ad hoc queries.

Conceptual Layer. This layer defines the conceptual data types that are needed by the user interface layer. These conceptual data types include the following:

1. *Form.* A form is based on a query, and it has the format information needed for creating a form-view and some basic control methods in order for the user to browse the records through a form.
2. *Report.* A report consists of queries and format (or structural) information needed for creating a report.
3. *Query.* An SQL query such as SELECT, DELETE, INSERT, and UPDATE.
4. *View.* A view gives a partial information of a schema or a subschema.

We can look at these concepts as a group of objects that work interactively with each other, transform operations from the user interface layer to operations of the data model layer, and pass results from the data model layer to the user interface layer.

Data Model Layer. This layer defines the semantic data model needed by the upper layers. It interacts directly with the Sem-ODB engine, gets all the information needed by the upper layers, and structures them together. This layer consists of objects needed to access the Sem-ODB, such as SCHEMA, DATABASE, CATEGORY, and RELATION.

Front End: Database Engine Interface. We have defined an abstract layer between the data model layer and database engine to tolerate any modification of the engine's interface in the future. This can also bring a benefit such as a uniform programming interface for the front-end product, an idea similar to ODBC. This architecture is shown in Fig. 2. The data access functions to the database can then be partitioned into three distinct areas: client access, application server, and data source, as shown in Fig. 3.

The SQL Query Builder gives the end users the ability to access database schemas and build various SQL statements. The user does not need to have any background knowledge of the database schema or to be previously exposed to the SQL world. The SQL query builder provides intuitive methods to access in a constant time algorithm any schema objects and data on the remote database. The SQL query builder walks the user through a sequence of interactive screens. After query building is done, the user can submit the constructed query to the remote SQL server where results are transferred to the client's desktop. The query builder gives the user the illusion that the query is executed locally because of fast response time in query generation and screen navigation. The form/report designer gives the users the opportunity to design customized forms and reports on the fly. Syntax verification and formatting is done locally on the client side without consulting the server. This results in a reduction in the amount of data that transfers between the client and the server. The form/report presentation is responsible for displaying the outputs resulted from executing forms or reports on the server. The server sends the results back to the client where results are formatted according to the specification of the forms and reports.

The server is composed of four components: SQL Query Execution, Form/Report Formatting, SQL Generation, and Data Access. SQL Query Execution is responsible for executing SQL queries supplied by the clients. Form/Report Format reads user specified templates, parses the lines in the template file, and creates an HTML formatted Web page. The SQL Generation component is responsible for generating SQL queries based on a set of commands supplied by the clients. Finally, the Data Access component is responsible for accessing data from databases using ODBC drivers or by making a direct call to the semantic database engine.

REPORT-GENERATING TOOLS

The recent popularity of the World Wide Web (WWW, or Web) has created a great increase in demand for applications that support access to databases from the Web. The purpose of these applications is to enable users to access databases remotely and to generate Web-based forms and reports in a simple and efficient way. That is, the user should be able to access or manipulate the data of a database without having to write a complex application that would use database APIs for every specific task. A powerful solution to this is to use the database as a back-end or data source for Web applications and provide a set of functions that would interface with the database and incorporate those functions with the HTML statements (4). Combining the Web with the database maximizes the strengths of its components and achieves two goals:

1. From the application developer's perspective, this is ergonomic since the HTML is the formatting language that is used with the Web and most programmers are acquainted with this language. In addition, it offers cross-platform compatibility and high-speed prototyping capabilities.
2. From the end user's perspective, it offers easy and ergonomic access to databases and manipulation of data as well as generation of database reports.

This section describes the underlying theory and tools used to create the Web-based forms and reports generators, using as a case study our tools WebRG, Web-SQL, and Sem-Access.

Basic Theory

Although different methods have been proposed and used as a means for integrating the database and the Web in order to generate reports on the fly from the Web, our tools used CGI to link the database and the Web. The underlying key to database integration is to create an application that runs on the Web server, connects to a database through an SQL server, performs a specific operation such as SQL processing and report construction and formatting, and outputs the results in an HTML format. Integrating this application with CGI creates an interface that can send the output of a database query to the client. This is achieved in seven steps, as shown in Fig. 4.

1. The client machine on a Web browser requests information from a URL. This URL is the home page for the databases that are provided to the clients. After the page is retrieved, the client can select any database (usually by a click).
2. The Web server receives the request from the client and runs an instance of the requested CGI application.

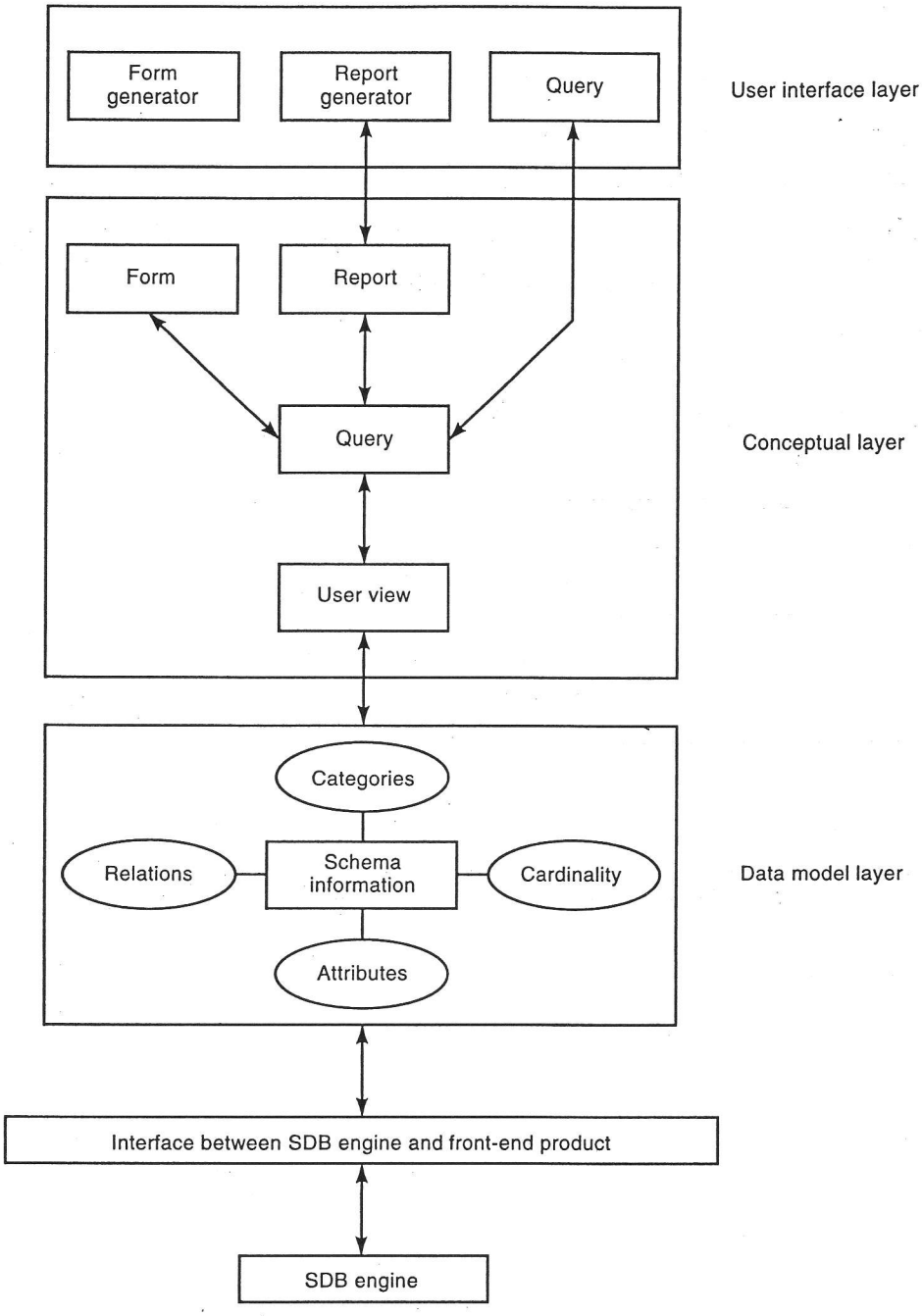

Figure 2. Architecture of the front end of SDB.

3. The CGI application establishes a connection with the SQL server and sends to it the SQL statement.
4. The SQL server executes the specified SQL statement from the specified database.
5. The SQL server retrieves the output of the SQL query sent back from the database.
6. The SQL server then relays the information back to the CGI application. The CGI program does a further processing and massaging to the data, such as adding HTML codes that the browser requires to display the data before it is returned to the client.
7. Finally, the Web server sends the results that were dynamically generated by the CGI application back to the client. The data returned to the client might be a list of categories, attributes, and relations of a specified database or the results of executing the SQL query formatted in a Web-based form or report.

In addition, because HTML allows words in a text document to become hypertext links to other URLs, the CGI program can output hypertext links that call the same CGI program with different options or other programs, not necessarily CGI, to do further database operations.

The Web-Wired Database and Configuration

Essentially, the SQL server is a gateway between the Web server and the database source as well as a gateway between

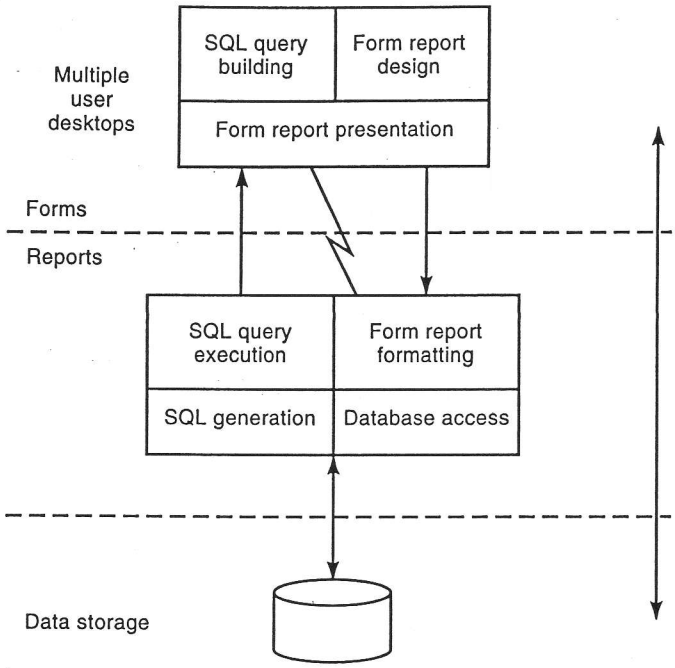

Figure 3. Form/report data access and generation.

the ODBC data source and the Web server. The Web server handles the communications and simply passes the data back to the client. Multiple instances of the SQL server can be run by specifying different TCP port numbers.

The Web server needs only one connection from the client to the SQL server. The SQL server, in turn, handles the transactions with the database transparently. In other words, the client does not need to worry about how to deal with the database APIs to complete the specified request. All other clients connect using standard browsers that are designed to handle the networking to the Web server. Therefore, the graphical user interface and other networking drivers have already been provided and the developer does not need to concern himself or herself with this aspect in the process of the application's design. The developer needs only to manage the information that is sent back from the SQL server. Figure 5 shows the configuration for a database-linked SQL/Web server. The Web server and the SQL server can be dispersed geographically. Upon invoking the CGI application on the Web server, the CGI program establishes a connection with the SQL server by specifying the IP address of the SQL server and the database name.

Techniques Employed in the Case Studies

WebRG. WebRG is a Web (Internet/Intranet HTTP) application allowing application designers to create dynamic documents easily. The documents that are created have the simplicity of HTML and the functionality of SQL and CGI. WebRG makes it easy to add live data to static Web pages. Live data includes information stored in databases. Moreover, WebRG has the functionality of creating simple dynamic Web pages or complex Web-based applications.

With WebRG, a Web macro interface tool, the database application designer defines the user interface as "macro file" that contains SQL, HTML, and control statements. When the Web server receives a URL that refers to WebRG and a macro file, the Web server starts an instance of WebRG and passes initial information to it, including the name of the macro file. WebRG reads and parses through the macro and interprets the statements. After all parsing is done and language environment processing is completed, all that remains is an HTML text that can be processed and interpreted by any Web browser. Then the HTML text is passed back to the Web server and WebRG terminates. The resulting HTML text is passed to the Web browser, where the user interacts with it. This text may be a form requesting user interaction, which results in the process repeating itself from the beginning.

WebRG can be invoked either from an HTML anchor reference or form or directly as a URL. WebRG starts when an HTML form input for a WebRG application is sent to the Web server. The Web server passes to the WebRG application the name of the Web macro file, the name of the HTML section in the Web macro file, and any other input variables.

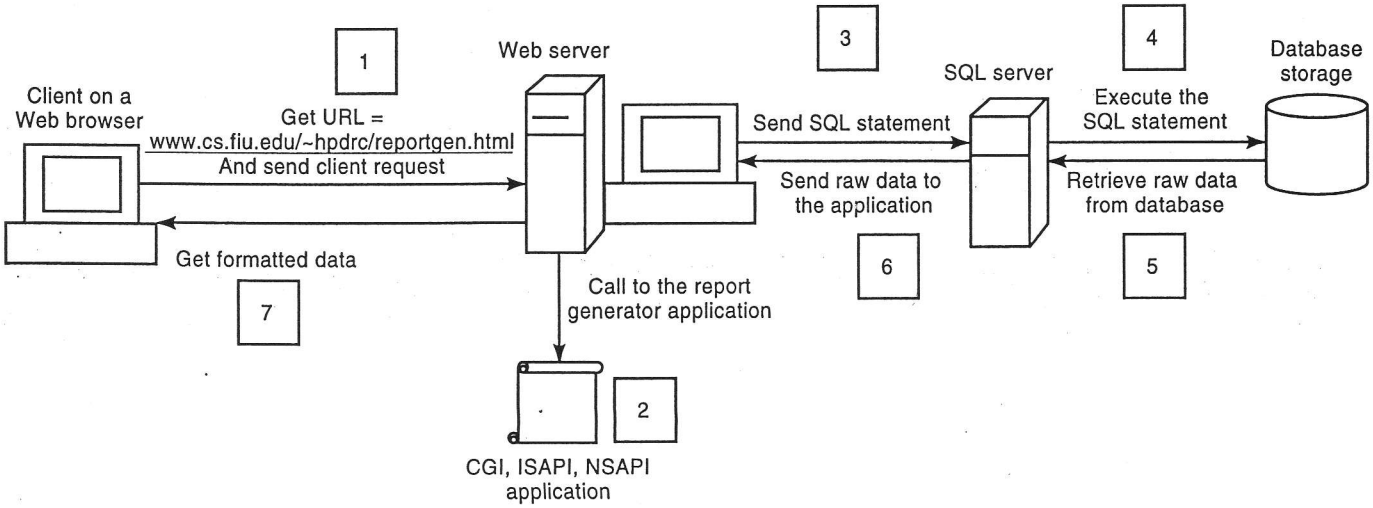

Figure 4. Client/server communication.

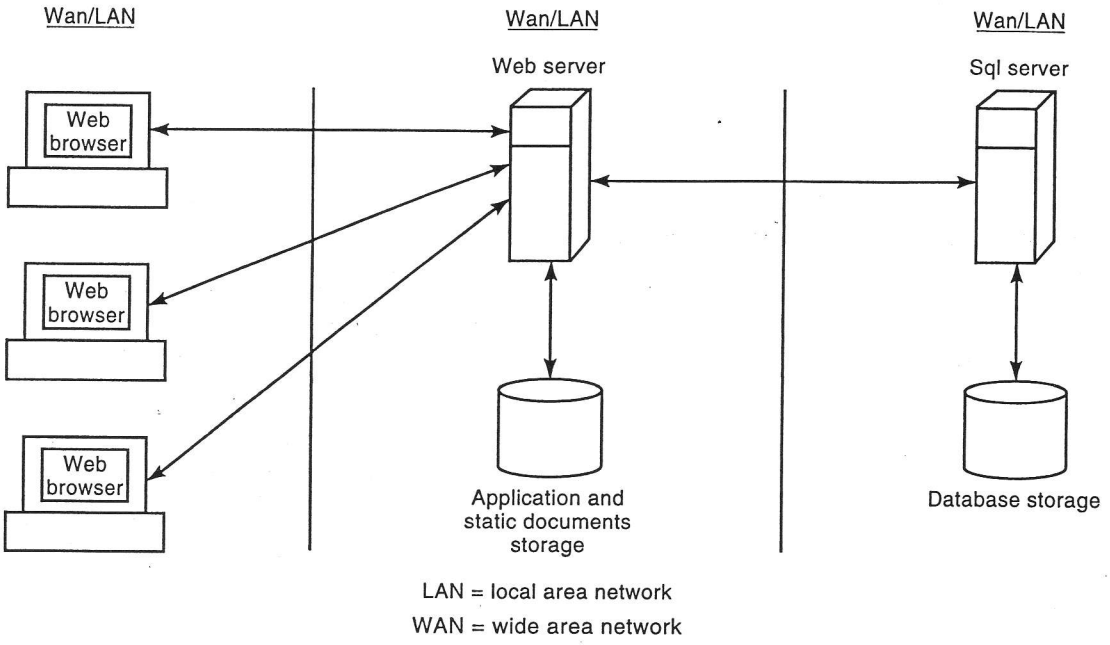

Figure 5. Web/database configuration.

WebRG takes yet another approach to produce the resulting HTML document. In this approach, the appearance of the interface and the database operations are specified through a series of user-defined forms. Since WebRG form is a macro file written in HTML in addition to a set of database interface statements, HTML lines and all WebRG statements form a database Web interface language for design. WebRG forms are processed by an interpreter/compiler to produce a final HTML document. Thus, a database application developer can create a series of forms and associated SQL statements for the end user's own needs.

The WebRG can also hook other databases, such as relational databases, to the Web by the use of the ODBC. The ODBC protocol permits us to connect to any SQL database by offering uniform API functions that are supported by those databases. A query, submitted by the client from the Web, is transferred over the network in ODBC-SQL format. WebRG extracts data from the data source by connecting first to the ODBC driver manager. The driver manager, in turn, loads the driver on behalf of an application. Then the driver processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to the syntax supported by the associated DBMS.

To use this package, the application designer should have some basic knowledge about HTML and SQL. After getting familiar with the WebRG statements, the user then can create various form-based Web interfaces to the databases, which are fully user customizable. Whenever the user wants to change the interface, he or she has only to change the relevant form specifications. As a result, the WebRG tool provides a framework for both rapid building and easy maintenance of Web to database interfaces. A macro file consists of five principal parts that can be interlaced and nested:

1. The Define statements—Declare variables.
2. The HTML Input statements—Specify interactive forms.
3. SQL statements—Specify databases operations to be performed.
4. The HTML Report statements—Specify outputs.
5. Control statements—Specify execution flow.

These parts contain familiar HTML tags, which makes writing macro files easy. If the user is familiar with HTML, building a macro simply involves adding macro statements to be processed dynamically at the server. Although the macro looks similar to an HTML document, the Web server accesses it through WebRG as a CGI program. WebRG incorporates data from the database into the Web application. WebRG supports any SQL statements. Users can connect to one database at a time for each Web macro. However, multiple queries and updates can be incorporated in a macro file.

The Define Statements. WebRG lets the designer define and reference variables in a macro file. In addition, the user can pass these variables from the macro to another macro and so on. Macro variables are categorized into three types:

1. *User-defined variables.* These are variables the user defines and references in the macro file.
2. *Special macro variables.* These are variables that the user still has to define but that have special meaning to WebRG.
3. *Implicit variables.* These are variables that are implicitly defined by WebRG and may be referenced by the user.

The user must always define the variable DATABASE for each macro that accesses a database. In addition, the user can define all the different types of variables supported by WebRG.

Syntax Notation. Items to be replaced are in *italics*. { item } denotes repeatable item, [item] denotes optional item. [item | item | item] denotes a choice of one of the items. There is a one-line and multiple-line syntax:

```
%DEFINE varname = value
  and
%DEFINE { {varname = value} %}
```

The value specified can be a numeric value (e.g., 5.21) or a quoted character string (e.g., ''John'').

HTML Input Statements. The HTML input statements define the HTML form where end users can specify information they want from the database by entering values in the form using their Web browsers. Input is dynamically placed in the user query. WebRG macros do not require an HTML input section for simple queries. The syntax is

```
%HTML_INPUT[{HTML-text-on-multiple-lines %}
```

The HTML text can contain and allow assigning values to variables.

SQL Statements. The macro file can have multiple SQL statements or sections, but each section can contain only one SQL command with an optional report specification and error message handler. Optionally, the user can name one or more sections to call them from anywhere in the HTML report using the section name. The syntax is

```
%DB(sql-section-name){Sql_Statement
  Report_specification Error_Handling %}
```

- Sql_Statement: any SQL statement on one or multiple lines.
- Report_specification: optional

```
%DB_REPORT{ header_specification row_handling
  footer_specification %)}
```

- Header_specification & footer_specification: are HTML text, which may contain variables.
- Row_handling:

```
%ROW{ HTML-text-to-display-once-for-each-row-
  returned %}
```

Report specification gives the user the ability to customize the query's output using HTML formatting. If the macro has no DB_REPORT section, a default table is displayed with column names at the top. All text and graphics before the %ROW declaration are header information and are displayed at the top of output. Upon SQL execution, the column names are placed in special variables Ni, column-name, and NLIST (number of columns), which can be used in header specification and row handling. The row handling contains information displayed once for each row returned by the SQL query. The text in the row handling would normally use variables containing the row output: V1. . .Vi. Information, including text and graphics, following the ROW subsection is footer information and is displayed once after all rows are displayed.

- Error_handling: optional

```
%DB_MESSAGE{ {DBCODE: ''warning message'':
  [exit | continue]} [default: ''default
  message''] %]
```

The error handler allows the user to customize error and warning messages from SQL commands. If the user places this declaration inside a SQL statement, it is local only to the SQL command in that section. If it is outside of all SQL statements, it is global to the entire macro. Furthermore, the user can create a table of DB codes and specify the information to display following each DB code. The default error message is shown when a DB code not in the declaration is returned by the special variable DBCODE. For positive DB codes, the user has the option of exiting or continuing.

If an error occurs in a SQL command, the execution terminates and a return code is given. The warning or error message can be any HTML text, including links to other URLs.

HTML Report Statements. An SQL statement is executed when it is called by %EXEC_SQL in the HTML Report statement. The Report statement is executed when WebRG is started in the report mode.

```
%HTML_REPORT{ HTML-text {%EXEC_SQL (SQL-
  section-name) HTML-text }
  HTML-text %}
```

The user can specify any HTML and include any variables from the DEFINE statement section in the HTML code. User's input from the HTML form overrides variables in the %DEFINE statement. When a %EXEC_SQL line is encountered, the DB section matching the name or defined variable is called. Using a variable for the SQL section name is an easy way to allow users to select a query to perform.

If the user does not specify a section name, all unnamed SQL statements are executed in the order they appear in the macro.

Control Statements. Control statements are used in order to control the flow of the execution of the statements in the macro files. We have three types of control statements: IF statement, assignment statements, and list statement.

IF Statement. The IF statement allows conditional branching. The IF statement can be incorporated in the HTML_INPUT section and the DB_REPORT section. The syntax of the IF statement is

```
%IF(expression) THEN{ {statement} %}
{ %ELSIF(expression) THEN{ {statement}%} }
[%ELSE{{statement} %}] ENDIF
```

One can have zero or more %ELSIF inside the %IF statement and zero or one %ELSE statement. The *expression* is an arithmetic expression of type Integer, Real, or Boolean. *statement* are executable statements in the same program unit as the IF statement.

Assignment Statement.

- Assignment statement: *varname = value*. Assigns a value to a variable.

Conditional statements are used to determine if a string is empty. Conditional variables have two main forms:

- *varA=varB?value_1: value_2*
  If *varB* is defined or not empty, *varA=value_1*, otherwise *varA=value_2*.

- *varname* = ?*value*
  varname is null if *value* is empty, otherwise varname is set to *value*. The *value* may contain variables. If any of the variables is empty, then this is an assignment of the empty string to *varname*. Otherwise, *value* is evaluated and assigned to *varname*.

The application designer needs to replace the quotes with braces {. . .%} if the values cover more than one line.

List Statement. The list statement concatenates several items with delimiters. This is useful when constructing an SQL query, header lines, output rows, and so on. The designer can select different columns from a table. The syntax is

```
%LIST ''delimiter'' variable_name
  {assignment_statments}
```

The list statement must be defined in the DEFINE statement with the associated values. *Delimiter* separates the different values that are assigned to a variable.

Examples of assignments and list statements are as follows:

```
%HTML_INPUT{
<FORM METHOD=''post'' ACTION=''http://
  www.cs.fiu.edu/sdbwww/example.mac/report''>
Select one or more: <BR>
<INPUT TYPE=''checkbox'' NAME=''cond1>Smith<BR>
<INPUT TYPE=''checkbox'' NAME=''cond2''>Lee<BR>
<INPUT TYPE=''submit'' VALUE=''Submit Query''>
</FORM> %}
```

The variables *cond1* and *cond2* are HTML input variables passed from CGI, usually from an HTML_INPUT section.

```
%LIST ''OR'' wherecondition
wherecondition = ? ''last_name LIKE
  ''$(cond1)''''
wherecondition = ? ''last_name LIKE
  ''$(cond2)''''
WhereClause=wherecondition ? ''WHERE
  $( wherecondition)'': '''' %}
%DB SELECT last_name, first_name, birth_year
  FROM STUDENT $(WhereClause)
%HTML_REPORT{
%EXEC_SQL %}
```

- The preceding *%LIST* statement declares the variable *wherecondition* with the OR delimiter.
- A conditional statement is made to the *WhereClause* variable. The *wherecondition* is null if no boxes are checked (where we supply the user with a form that contains two check boxes with cond1 and cond2), so *WhereClause* is set to the second value (null). If a box is checked, Conditions, is defined, then *WhereClause* is assigned the first value *WHERE $(Conditions)*.

The value of *$(WhereClause)* depends on which two boxes are selected on the form. There are four possible combinations, which result in four different WHERE clauses:

- Neither box is checked. *Null*
- Only *Smith* is checked. WHERE last_name LIKE "Smith"
- Only *Lee* is checked. WHERE last_name LIKE "Lee"
- Both boxes are checked. WHERE last_name LIKE "Smith" OR last_name LIKE "Lee"

Web-SQL. In many cases, a sophisticated user desires to send a request directly to the database (e.g., issuing an SQL request from a WWW client manually). For those users who have a solid knowledge about SQL, a tool for submitting a native SQL query directly to the SQL server is helpful. Typically, the user is first presented with a home page, which displays a list of database names in addition to their locations in the form of URL. The user then selects the database name of interest. Once the user has selected the database name, a request with the database name and the application name is sent to the Web server. The Web server, in turn, runs an instance of a CGI program, which retrieves the database schema. The results are then massaged and sent back to the client. The user would use the schema information provided as a guideline to submit an accurate and meaningful query.

In addition, a textbox is provided in the same page to give the user the ability to type a SQL query. After completion of the SQL query, the query is then sent to the Web server, where the CGI application resides. The CGI program then sends the SQL query to the SQL server. The SQL server runs the query on the intended database and sends the raw data back to the CGI program. Then the CGI program formats the data in a table or a Web report format that appears as a grid of rows and columns, its contents typically filled with the results drawn from a database table. Those results are then sent back to the client in a Web-formatted report. The user may return to the query prompt, at which time the user may enter a new query. The system can handle arbitrary SQL queries and updates. Binary data are treated differently. When the CGI application receives a SQL query and determines that some of the fields that are generated of binary values, automatically an HTML container for the binary data is created. The type of the container is determined by reading the first bytes of the binary data (i.e., a GIF format).

Accessing the database and query editing and processing is done at runtime using an SQL engine. Since the tool does not impose restrictions on the syntax and the semantic of the SQL query, it is a means for a sophisticated user to write powerful and efficient queries. Password and group management is supported when access needs to be restricted to certain users or groups. The access restriction is based on the following criteria:

1. Username/password authorization
2. Rejection or acceptance of connections based on Internet address of client
3. A combination of these

Having a preprogrammed interface (e.g., using WebRG) creates rich reports that can be tailored to a user's needs and preferences. However, Sem-Access provides a user-friendly generic ad hoc query tool that serves all kinds of users.

Sem-Access. Sem-Access is a tool that gives the user the ability to navigate through a semantic database schema, which causes the system to generate appropriate SQL queries, updates, and reports. It is a generic ad hoc querying tool.

Sem-Access builds SQL statements automatically. The SQL queries are built in a sequence of steps as a result of user interaction and selection of Categories, Relations, and Attributes.

A simple Web form, which is a static HTML file, contains a list of databases in addition to the user name and password. As a result of user selection of database fields, a Web form allowing the user to enter filed values and operators is automatically generated. The system derives such forms from the database schema and user's navigation. The user is provided with a set of common interface elements (e.g., check boxes, radio buttons, data-entry fields) and a way of binding these elements. As the user fills the forms, SQL segments are generated. If the sophisticated user wishes to formulate a very complex query beyond what is generated by filling forms, he or she can manually add SQL segments in a textbox provided. Sem-Access then assembles an SQL query and submits it to the SQL server.

All pages, including the front Web form page, are generated on the fly (i.e. based on current content of the database). Thus, whenever a new row enters the database through a normal insert operation, any resulting new keywords will automatically appear in the Web page. The returned results are from raw data stored in the database, and query processing is done at runtime with the SQL engine.

Queries Generation. A checkbox is associated with each category name. When the user clicks a category, a new form is generated detailing the attributes and the relations of that category. Each attribute has a listbox and a textbox. The textbox allows entering attribute values. The listbox contains some of the following operators: <, >, =, <=, >=, <>, Exact, Prefix, Suffix, and Infix. The attributes that are of numeric type have the operators ''<'', ''$<'', ''='', ''<='', ''>='', ''<>'' in the listbox as well as other user-enterable operators. However, attributes of type string have the operators ''Exact'', ''Suffix'', ''Prefix'', and ''Infix'' in the listbox.

The suffix operator, for instance, is used to indicate the parts of a database that end with some pattern. For example, consider the category STUDENT with attributes first_name, last_name, and birth_year. Suppose the user wants the list of students whose last name ends with "th." The user would enter 'th' and click 'suffix', which results in the SQL statement ''SELECT * FROM STUDENT WHERE last_name LIKE '%th'''.

The interpretation of user-supplied value in an attribute's textbox depends on user's operation: Query (SELECT, INSERT, DELETE, or UPDATE).

If the user clicks a relation, a new form is generated traversing the relation to the next category. The related category is expanded with its associated attributes and relations. For example, consider the category ''STUDENT'' and its attributes ''first_name'' and ''last_name'', and its relations ''minor'' and ''major'' with the category ''DEPARTMENT''. If the user clicks on ''minor'', the category DEPARTMENT is expanded. The path, in this case the relation ''minor'', would be displayed next to each attribute and relation to indicate the path that the category is derived from. The user can continue from the expanded category to expand more categories in the query tree. A tree of the traversal of the categories and user previous selections and preferences is stored in every subsequent page. A tree that indicates the paths of every derived attribute is kept in every page in order to improve the processing time for further requests. Moreover, as the Sem-Access application executes, guiding the user from one entry field to the next and encountering fields that depend on values from previous forms, Sem-Access carries the information on which the fields depend to the current page. This nesting goes arbitrarily deep into the form stack, and the Sem-Access keeps track of the nesting path that is associated with each attribute. Figure 6 presents the process of interaction with the Sem-Access interface Web page. It displays a list of categories. The category "student" is expanded due to its selection. As shown, the "student" category contains a list of attributes and relations that are also expandable. The selection of any of these relations results in the expansion of other categories that are related to the selected relations.

The following four subsections explain and exemplify algorithms to generate SQL queries.

Semantic SQL Statements. The semantic database engine system Sem-ODB uses the industry-standard SQL language. It is interpreted against a virtual relational database making SQL programs much shorter than for an actual relational database. A virtual table is defined for each category C as a spanning tree of all the relations reachable from C. Therefore, joins, the main overhead of a relational SQL program are needless. Thus, a user can pose an arbitrary query against a semantic database by projecting on a single category. The tree spawns direct relations R and inverse relations $R\_$. We discuss the algorithm that is used to construct a semantic SQL query by prompting the user to traverse a semantic database schema.

SQL SELECT Statement. The syntax of the SQL SELECT statement is

```
SELECT attributes FROM category WHERE
  conditions.
```

The semantic database query can access multiple categories without the need to make join between categories. Therefore, we enable the user to select only one category from the schema. For instance, suppose the user marks the checkbox of the category STUDENT. This will lead to the generation of the FROM clause ''FROM STUDENT''. If the user checks the category STUDENT and then the relation ''minor'', the category DEPARTMENT will be expanded to show its associated attributes and relations on the path ''minor''. The category DEPARTMENT is expanded because of the direct relation ''minor'' from STUDENT to DEPARTMENT. Now suppose the user wants to display all the attributes of the category DEPARTMENT; this will result in the construction of the following query:

```
SELECT minor__ FROM STUDENT
```

If the user wants to display all the attributes of STUDENT by projecting on DEPARTMENT, the following query is generated:

```
SELECT ___minor__ FROM DEPARTMENT
```

The attributes that are checked are treated as parts of the *attributes* clause. To illustrate, suppose the user checks the category STUDENT and the attributes first_name and last_name. The category DEPARTMENT is directly derived from the relation major. If the user checks the attribute ''name'' of the category DEPARTMENT, the following SQL SELECT query will be generated:

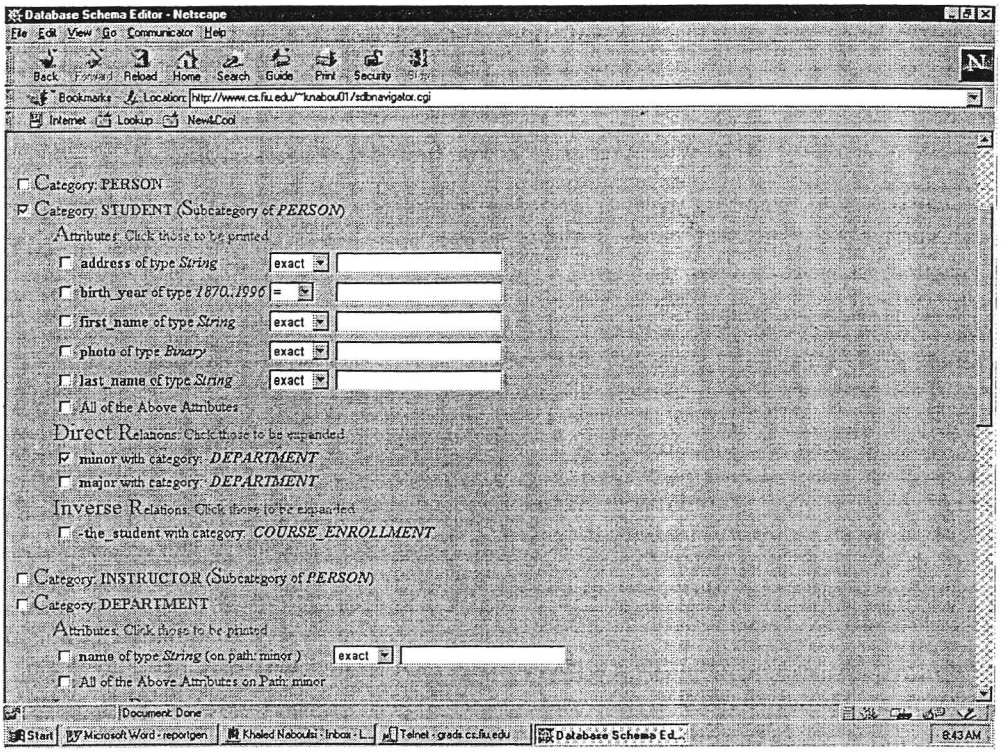

Figure 6. Sem-Access: Viewing query generation process.

```
SELECT first_name, last_name, major__name FROM
  STUDENT.
```

On the other hand, if the user types in the textbox associated with the attribute `last_name` `''Adams''` and selects `''Exact''` from the listbox, the SQL SELECT query will be the following:

```
SELECT first_name, last_name, major__name FROM
  STUDENT WHERE last_name = 'Adams'.
```

We will state some examples to illustrate the generation of the *conditions* clause. Suppose we want to get the list of students' names and majors where students' last names begin with the letter "A" and they have the computer science as a minor. In this case, the user needs to click on the category STUDENT and checks the box that indicates `''All Attributes''` and the relation `''minor''`. As a result, the two categories STUDENT and DEPARTMENT would be expanded with their associated attributes and relations. However, as has been noted, the attributes and relations of the category DEPARTMENT would have the path `''minor''` as an indication of the path where the DEPARTMENT category is derived. Moreover, the user needs to select `''Prefix''` from the listbox of the attribute `last_name` and type `''A''` in the textbox. The user also has to select `''Exact''` from the listbox of the attribute `''name''` in the category DEPARTMENT and type `''computer science''` in the textbox. The generated query is

```
SELECT STUDENT__, major__ FROM STUDENT WHERE
  last_name LIKE 'A%' AND minor__name =
  'Computer Science'.
```

suppose we want to retrieve all students whose birth year is greater than 1970. In this case, we need to mark the `''All Attributes''` field of the category STUDENT and select `''> ''` from the listbox of the attribute `birth_year`. Furthermore, the user needs to type 1970 in the textbox of the corresponding attribute `birth_year`. The following query will be generated:

```
SELECT STUDENT__ FROM STUDENT WHERE
  birth_year > 1970.
```

SQL DELETE Statement. The syntax of the SQL DELETE statement is

```
DELETE FROM category WHERE conditions.
```

The FROM clause is constructed by clicking on a category name. For example, if the user selects the category names STUDENT, this will be interpreted as `DELETE FROM STUDENT.`

The *conditions* clause of the SQL DELETE statement can be constructed in two ways.

1. Attributes whose check boxes are selected are treated as part of the where clause. Then an operator is selected for each attribute and a textbox is filled with the appropriate values. For example, suppose that the user selects the attributes `first_name`, `last_name` of the selected category STUDENT, selects the quantifiers `''Prefix''` and `''Infix''`, and types `''A''` and `''Brown''` accordingly. In addition, suppose that the user selects the attribute `''name''` of the category DEPARTMENT derived from the relation `''major''`, selects the operator `''Exact''`, and types `''Computer Sci-`

ence'' in the corresponding textbox. As a result of these actions, the following SQL DELETE statement will be constructed:

```
DELETE FROM STUDENT WHERE first_name LIKE
 'A%' AND last_name LIKE '%Brown%' AND
 major__name = 'Computer Science'.
```

This query states that we want to delete students whose first name start with ''A'' and last name contains ''Brown''.

2. The same query can be derived without having the user check all the boxes of certain attributes. In other words, by only typing some of the information in a textbox that belongs to certain attribute, the same action results by considering the information in the textbox as part of the WHERE clause. Considering the previous example, where the user selects the attributes first_name and last_name of the category STUDENT, but instead the user types ''computer science'' in the attribute ''name'' of the category DEPARTMENT without having to select the attribute ''name'', the same previous query will be generated.

SQL UPDATE Statement. The syntax of the SQL UPDATE statement is as follows:

```
UPDATE category SET attribute1=value1 [,
  attribute2=value2. . .] WHERE conditions.
```

The *category* clause is constructed similarly to the construction of the categories that we have already explained in the previous sections.

The *set* clause is a list of one or more attributes and values. In general, any attribute that is selected, whether the corresponding category is derived or directly implied (selected), becomes part of the *set* clause. However, the operators that are associated with the attributes are disregarded and not treated as part of the *set* clause. The reason for this is that the syntactic structure *set* clause has the attributes and the values separated by the operator ''='' . This operator (=) is the only valid one in the *set* clause. Therefore, the selected attribute and the value of the corresponding textbox are taken into consideration.

The *conditions* clause is constructed by typing data inside the textbox that belongs to some attribute without checking the attribute's check box. That is, the check box of a particular attribute should be unchecked and the user should select the appropriate value of the corresponding listbox and enter data in the textbox that belongs to that attribute. To illustrate, suppose we want to change the major of the student ''James Fox'' who was born in 1970 to Chemistry. We need to select the attributes first_name, last_name, birth_year and set their corresponding values ''James'', ''Fox'', and ''1950''. Then, without checking the attribute major, we need to type the value ''chemistry'' in the field corresponding to the attribute ''name'' of the category DEPARTMENT derived from the relation major. In addition, the user selects the operand ''='' from the listbox that corresponds to the attribute ''name''. The following UPDATE statement will be generated:

```
UPDATE STUDENT SET major_name = 'Chemistry''
  WHERE first_name = 'James', last_name = 'Fox',
  birth_year = 1970.
```

If an attribute should be both a part of the *set* and *conditions* clauses, the only way to specify a condition on an attribute that is also set is to put it in the ''Other Conditions'' box provided.

SQL INSERT Statement. The SQL INSERT statement has two forms:

```
INSERT . . . VALUES
```

and

```
INSERT . . . SELECT
```

The INSERT INTO statement enters data into a table one record at a time. The syntax is

```
INSERT INTO category (attribute1,
  attribute2,. . .) VALUES (value1,
  value2,. . .]
```

The basic format of the INSERT. . .VALUES statement adds a record to a table using the columns the user gives it and the corresponding values he/she instructs it to add.

We have chosen not to discuss the generation of the <category> clause due to the notable similarities found between the INSERT and the UPDATE statements.

The *attributes* clause is constructed by checking the appropriate attribute(s) of a particular category and entering data in its corresponding textbox. So, if the user checks the attribute ''last_name'' and enters ''Smith'' in the textbox that relates to a particular attribute, the following INSERT. . .VALUES statement will be generated:

```
INSERT INTO STUDENT (last_name) VALUES ('Smith').
```

The INSERT. . .VALUES query is useful when adding single records to a database table. However, to enter substantial amount of data efficiently, it is necessary to use the INSERT. . .SELECT statement. The syntax of the INSERT. . .SELECT statement is

```
INSERT INTO categoryX (attribute1,
  attribute2,. . .) SELECT attributes FROM
  categoryY WHERE conditions.
```

So the output of the standard SELECT query is then the input of *categoryX*.

The SELECT statement embedded in the INSERT. . . SELECT statement is obtained by having the user enter the SELECT statement in the big textbox that is provided at the bottom of the page. This textbox gives the user the flexibility to specify more sophisticated query.

Batch Printing and Import/Export of ASCII Data

Report-generating tools must be able to supply three kinds of output: viewable, printout, and ASCII data for postprocessing. Each output should be procurable either interactively or via batch script. Since printing is a vital feature of report generators, users should be able to print the results retrieved from the database in a nice and readable format. We have empowered users with various ways to print forms and reports. For instance, interactive procurement of printing can be accomplished using the print facility of Web browsers. Our reporting tools offer the following features:

1. Our tools make sure that directing outputs to a printer results in a nice printout (with other arbitrary HTML text, this is not the case as a document may be nicely displayed but not designed for properly paginated printout).
2. Batch printing is accomplished via a script that submits a request to the server via a non-interactive browser, like `''lynx-dump''`, and sends the output to the printer.
3. For postprocessing, the user can obtain a standard ASCII form of any report.

The data retrieved from submitting an SQL query to the database is composed of columns and rows. Every column represents an attribute of a table in a database. In ASCII rendition, tabs separate columns and new lines separate rows. The structure of the standardized ASCII file is

```
document-title
= /* a delimiter */
lines-of-column-headers
= /* a delimiter */
data
```

The data portion of the file is the rows results retrieved from the database. Users can then feed this file to any application tool that would do further processing of the data.

CONCLUSION

Combining the Web technology with the strengths of database presents a great challenge. The challenge is to provide an efficient, fast, and flexible method to link the database to HTTP servers and to offer a friendly user interface to generate interactive Web forms and reports. Three tools, developed in our labs, were discussed in this article as case studies, each tool with a different flavor and functionality. The reason for having different techniques is to accommodate all types of users who would utilize the form and report generator. One tool, WebRG, uses the HTML power and integrates it with SQL query ability. In WebRG, the designer defines forms and Web-based reports by constructing macro files. The macro file contains a series of macro language and HTML statements, invoked upon executing the WebRG application. We also provided open data access solution through ODBC support. Another tool, Web-SQL, allows users to generate standard formatted reports from the Web. This is accomplished through the execution of embedded SQL statements directly within the HTML pages. The user can write SQL statements based on the schema provided as a guideline and against which to pose queries. Yet another tool, Sem-Access, allows users to traverse a semantic database schema while automatically generating SQL queries. This eliminates the users having to struggle with the details of database design or having to learn SQL. Sem-Access provides the users with a view of the database schema. A view gives a partial information of a schema that is defined by the administrator to be granted to every particular database user. The view is determined upon authenticating the user name and password against the users defined in the database. The user then can construct Web reports by interacting with the schema components (categories, attributes, and relations). The tools then submits SQL SELECT, UPDATE, DELETE, or INSERT statements. This empowers the user to create reports without programming.

Accessing data from the Web is one way to increase the availability of data. However, data should be retrieved in a timely manner, a feature that is important to decision-making systems. Some queries require a large number of joins. Performing those complex queries on large databases can be very time-consuming (5). Distribute those databases among multiple machines, and the problems multiply. Multiple machines give us the ability to execute many operations in parallel, and we are now beginning to encounter multiprocessor computers that do parallel processing themselves, as well as new microprocessors that employ on-chip parallel pipelines. To take full advantage of this new multiprocessing capability, software should take advantage of parallel processing, a feature that is beginning to appear. The major database vendors are now offering parallel versions of their database engines (6–8). The goal is always to achieve a radical speed increase in query response. The area of parallel queries offers different strategies and theories to determine the most efficient way to execute queries. Our report generator offers a feature that allows clients to view the partial results of query as the query is executed. In other words, as the query is executed on the SQL server, the application server retrieves the partial results and sends them to the clients without having to wait until query execution is complete.

ACKNOWLEDGEMENT

This work was supported in part by National Aeronautics & Space Administration (NASA) (under grants NAGW-4080, NAG5-5095, and NRA-97-MTPE-05), National Science Foundation (NSF) (CDA-9711582, IRI-9409661, and HRD-9707076), Army Research Office (ARO) (DAAH04-96-1-0049 and DAAH04-96-1-0278), US Air Force Research Laboratory (AFRL) (F30602-98-C-0037), US Dept. of the Interior (DoI) (CA-5280-4-9044), North Atlantic Treaty Organization (NATO) (HTECH.LG 931449), and the state of Florida.

BIBLIOGRAPHY

1. W. Bosques et al., A spatial data-retrieval & image processing expert-system for the World-Wide-Web, *Comput. Indus. Eng.*, **33**: 433–436, 1997.
2. J. K. Whetzel, Integrating the World Wide Web and database technology, *AT&T Technical Journal,* **75**: 38–46, 1996.
3. N. D. Rishe, *Database Design: The Semantic Modeling Approach,* New York: McGraw-Hill, 1992.
4. L. Bruno, Web application servers dynamic dividends, *Data Commun. Int.* **V51**: 38–42, 1997.
5. M. J. Tucker, Managing your Web-to-database performance, *Datamation,* **43**: 106–108+, 1997.
6. K. Watterson, SQL Server 6.5: one for the Web, *Datamation,* **42**: 57–68+, 1996.
7. R. Grehan, Building SQL front ends, *Byte,* **18**: 238–242+, 1993.
8. O. Sharp, Databases get objective, *Byte,* **20**: 13–16+, 1995.

NAPHTALI RISHE
KHALED NABOULSI
OURI WOLFSON
Florida International University