An Efficient Web-based Semantic SQL Query Generator

Naphtali Rishe*, Khaled Naboulsi*, Ouri Wolfson**, Bryon Ehlmann***
*High Performance Database Research Center
Florida International University, ECS354, University Park, Miami, FL 33199
**University of Illinois at Chicago
***Florida A&M University
hpdrc@cs.fiu.edu

Abstract

The Internet can provide transparent access to distributed databases via users' web browsers. The client-server communication is accomplished using the hypertext transfer protocol (HTTP). However, the stateless architecture of the HTTP causes the client-server communication to be costly. Such communication requires a new database connection each time the user wishes to access the database. Applications typically access databases by submitting an SQL request to the server. Prior to submitting the actual query, the application typically submits a series of requests needed to help the user formulate the query. Reducing the amount of data between the clients and the server during query formulation enhances client response time and offloads SQL application server and database engine from frequent requests. Herein, we provide a model and an algorithm to efficiently retrieve database schema objects and offload the SQL application server by distributing the application processing between the client and the server. SQL queries may be constructed using a tree-like structure and are submitted to the SQL application server only when complete and validated syntactically as well as semantically. The model is unique in terms of simplicity and ease of SQL statement generation. [*]

1. Introduction

The Internet provides a number of advantages: low cost PC's, standard browsers, elimination of expensive remote networks and routers. However, the phenomenal growth of the Web traffic has sparked much research activity on improving the Web performance and scalability. The key performance factors are to reduce the volume of network traffic produced by Web clients and servers and to improve the response time for Web users.

Hooking the database to the Web produces another source of Web traffic where response time plays a key role in client-server communication. Users can access the databases through the Internet by posing SQL queries.

We provide a model to integrate the semantic database to the Web and a means for the Web clients to generate SQL queries with Web browsers using a tree-like structure. First, the user submits a request to retrieve the schema from a database. A SQL statement is then built via reference to the downloaded database schema, which is composed of categories, relations, and attributes. Thus database "hits" are eliminated by locally addressing the database schema. In addition, to minimize network traffic, the database schema is downloaded only when clients need to build a SQL query statement such as INSERT, DELETE, UPDATE, or SELECT. Schema navigation becomes efficient by creating a data structure of the schema at the client during the download of the database schema. The schema objects are loaded in a hashed heap data structure thus achieving a constant time access to any schema object. A user can then generate SQL queries and validate them syntactically and semantically without contacting the database engine over the network.

In short, the processing tasks are offloaded from the database thus eliminating database hits. More importantly, offloading minimizes the number of requests that must travel across the network between the client and the application server. The result is a reduction in traffic between the client and the application server. For instance, the SQL application could run entirely on the client's desktop without having to go out across the network and retrieve additional chunks of information to complete the building of the SQL query. Consequently, with less traffic between the client and server, the overall performance in the network is improved. Throughout our study the automatic generation of the SQL query and performance enhancements were emphasized.

[*] This research was supported in part by NASA (under grants NAGW-4080, NAG5-5095, NAS5-97222, and NAG5-6830), NSF (CDA-9711582, IRI-9409661, and HRD-9707076), ARO (DAAH04-96-1-0049 and DAAH04-96-1-0278), AFRL (F30602-98-C-0037), BMDO (F49620-98-1-0130 and DAAH04-0024) DoI (CA-5280-4-9044), NATO (HTECH.LG 931449), and State of Florida.

The remainder of the paper is organized as follows. Section two provides a brief background on the World Wide Web. Section three discusses the different methods to integrate the database with the Web. Section four presents the network programming languages to connect the database to the Web and the pros and cons of each. Section five gives an overview of the semantic object-oriented SQL. Sections six and seven discuss the model we adopted to generate SQL queries and database reports across the Web. The paper concludes with a brief summary of our reporting model and a discussion of security issues that should be considered when connecting the database to the Web.

2. World Wide Web Overview

The World Wide Web or WWW is based on a client-server interface model. A user can gain access to the information on the Web by using a Web browser. Client and server communication is always in the form of request-response pairs. Web clients and servers communicate using the HTTP protocol [1-2-3]. HTTP runs on the top of TCP (Transfer Control Protocol), a protocol on the transport layer. Communication is transported in the following manner: user selects a document to retrieve; resultantly, the browser creates a request that is sent to the Web server. Next, a TCP connection is established between the client and the server. This enables exchange of the client's request and the server's response. Once the request has been sent to the server, the client machine awaits the response. When the response arrives, the TCP connection is closed and the browser parses the reply. This process is repeated each time a client contacts the Web server with any request. Although the information and/or documents may be stored virtually anywhere around the network, the Web provides transparent access to these documents, so that users have the illusion that the documents are stored in a central location.

A Web server can respond to requests from many Web clients. The server usually listens on the designated port 80 for a request from a client to establish TCP connection. When the server responds to a client, it closes its TCP connection with the client and begins listening for the next requests.

The overall performance of the World Wide Web is affected by the clients, the servers, and the capacity of the network links that connect clients and servers. Web performance can be enhanced by using the client cashing approach and improving the client-server communication. The latter will be one focus of this paper.

3. Database and Web Integration

The main question is whether there is a value in integrating the database with the Web. A diversity of the client platforms such as Windows 3.1, Windows 95, Windows 98, Windows NT, OS/2, and Macintosh are installed on computer systems with wide variation in memory, disk space, and processor speed. Supporting all these platforms through a traditional application development requires time and money. Using Web access as the client interface of an application shifts that compatibility burden to Web browser vendors. In addition, access to the Internet is available to anyone, which eliminates the need for companies to extend their networks to accommodate all potential users. Web browsers can be used by every application that provides a Web gateway translator between hypertext markup language (HTML) and the database server's [4-5-6-7] application programming interface (API). The look and feel of the client presentation can be enhanced using Java applets to build powerful applications. Additionally, viewing partial information as the page is loaded, is particularly useful when retrieving a large volume of data from a database, a feature that is supported by HTML and Java.

Accessing a database from the Web using the HTTP communication is quite expensive [8]. In order for a database to communicate with a Web application server, a TCP connection has to remain open so data can be passed between the two machines. However, Web connections are short-lived because links open and close quickly. Each new request from a Web browser requires that a new connection be opened; thus the user must await a new connection to the database. Opening and closing a connection requires extra CPU cycles on the server application and the database server or engine. If each query to a database through the Web browser required a connection and login to the database, the accumulated overhead would have a withering effect on the database server and consequently on the user. By maintaining the connection to the data sources and keeping track of each client messaging in from the Web browsers, the whole interaction is controlled, getting around the stateless problem of the Web. Since there is no way to poll the client to determine if it still alive, we propose some sort of time out process where a connection to a given thread is dropped after a certain period of inactivity. The application server listens to a port that is known to its clients. The server keeps track of the clients that are currently connected using a connection manager. The connection manager runs as a separate thread. It keeps track of the live connections and manages the connections making sure that the clients are still active by sending a "keep-alive" flag message. The connection manager keeps the following client information: IP address, port, user

name, password, transaction cursor, last time response, and transaction information. In case a client connection to the server times out, the server sends a "keep-alive" message to the client. If the client fails to respond to this message, the server frees the client's connection and deletes the client entry from its hash table maintained by the connection manager. We will explain in depth the model of generating SQL query and client-server communication.

4. How to get the database talk to the Web

How to get the database to talk to the Web? Accessing the database from the Web can be done using any of the three approaches: CGI, Java, or proprietary APIs.

The CGI (common gateway interface) protocols for message passing from Web servers are used as middleware for initiating application processing. The Web browser and the server represent the application presentation layer and the CGI server handles application processing. The CGI scripts are started by the Web server upon client request. The CGI scripts convert HTML requests into a format that the database understands and translate the results to the clients. Thus, a CGI can generate dynamic HTML pages. The pages are composed of the data retrieved from the databases and HTML codes.

The CGI [9] scripts receive the parameters supplied by the client in the form of $NAME_1$=$VALUE_1$&$NAME_2$=$VALUE_2$&…&$NAME_N$= $VALUE_N$. The advantage of CGI is that it can be written in any development language supported by the Web server's operating system. The beauty of it is that the best development language can be chosen for any given situation. However, the CGI protocol has many inherent faults such as stateless session management and multiple process instantiation. In addition, CGI is limited by the fact that almost all CGI programs processing takes place at the server. In other works, in order to provide the user with any feedback, a trip has to be made between the server and the client. This results in poor performance in terms of network bandwidth and server CPU cycles.

Proprietary APIs tend to perform better because they are precompiled whereas CGI is converted to an executable form at run time. There are three main proprietary APIs: Netscape API (NSAPI), Internet server API (ISAPI) from Microsoft Corp., and Web Request Broker from Oracle. These APIs are not as portable as CGI since they support a specific API, but they are more efficient because programmers can code directly to the API itself.

The CGI and APIs are considered to be server site approaches because they are executed on the server site. Briefly, when the server receives an incoming call from a client, it processes the call with default defined steps (authorization, send back the result, log, etc). Here the programmer is allowed to customize, add, and specify a particular processing for a step. The server will call the customized service(s) by defined API's. These services are shared code, loaded once in memory and remaining resident, awaiting calls by the application(s). So, when the server needs to make such a call it's faster than CGI. Moreover, application developers can perform initialization tasks such as opening connection to the database and when a request arrives, it can be placed in the already opened connection.

Java, on the other hand, allows overcoming some of the limitations encountered in CGI such as the stateless HTTP protocol and Proprietary APIs such as the server-concentric communication overhead. Java [10] is an object-oriented programming language designed to be used on networks of heterogeneous computers. Thus, Java allows programmers to write applications that are platform independent. Java applets are downloaded over the network to the client's Web browser. The applets are transferred as byte codes from the Web server to a browser using the HTTP protocol. The Java applets are then executed by a virtual machine that runs on the browser. The beauty of Java language applications is that processing is distributed between the client and server. The key is to be able to run a broad range of programming interfaces and languages (such as Java) regardless of the vendor to communicate with the database.

The basic limitation of Java is that the byte codes are downloaded each time the client wants to access the server. This introduces a delay before the client can perform any kind of transaction against the server. Our approach is to save the byte codes on the client side and each time the client decides to access the server, the local copy of the bytes codes is compared with the server's version. At that time, a decision is made of whether to download the server's version of the ".class" byte codes to the client or to use the client's saved version. This is accomplished in the following sequence of actions:

1. First, the user goes to the HTML page that points to the server where the Java applets ".class" exist.
2. Java applets are downloaded over the network to the client's Web browser. The user is then prompted to save the Java classes to his/her local machine.
3. The next time the user wants to run the Java applets, s/he executes the Java applet saved on the local machine. Upon execution of the Java applet by a virtual machine that runs on the browser, a message is sent to the server requesting the server version of the applet.
4. The returned version is then compared with the client version. If the server version does not match the client version, Step 2 is repeated; however, the downloaded byte codes replace the

previously saved version. Otherwise, if the versions match, the user proceeds interacting with the report or the application server.

5. Semantic Object-Oriented SQL Interpretation

SQL queries have become the standard de facto language to access and manipulate data in relational databases. The SQL was also adapted by the semantic object-oriented database (Sem-ODB) [11]. The purpose of this adaptation was to be able to communicate with relational database tools. Thus, in order to access the semantic database, a SQL query is submitted to the database server. The Semantic Object-Oriented SQL syntax is compliant with the standard ODBC SQL (with null values). However, our SQL queries refer to a virtual schema. This virtual schema consists of an inferred table (T) defined for each category (C) as a spanning tree of all the relations reachable from (C).

5.1. Data Model

We define a formal representation of the data model with the following sets:

- A virtual table T.
- A finite set of attribute names A.
- A finite set of category names C.
- A finite set of relation names R.

In general, a *Relation* R is a descriptive of a set of pairs of objects that are related. For instance, *works-in* is a relation relating instructors to departments. More specifically, *works-in* is a direct relation from the category *INSTRUCTOR* to the category *DEPARTMENT* as shown in Fig. 1. Therefore, we need to keep information about the categories that could be reached by traversing a relation inversely and directly. For instance, the relation *works-in* will be hashed with the direct category name *DEPARTMENT* and the inverse category name *INSTRUCTOR*.

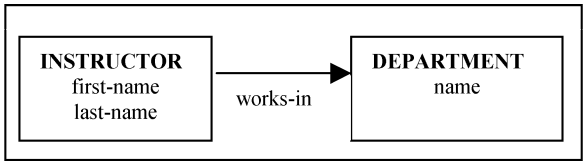

Fig. 1. An example of a database sub-schema

A *Category* C, on the other hand, is a unary property of objects. Objects could be concrete (i.e. number) or abstract (i.e. *STUDENT*). A category contains a set of attributes. For instance, the category *INSTRUCTOR* has the attributes *first-name* and *last-name*. In addition, a category has a direct and/or inverse relationship with other categories. For example, the category *INSTRUCTOR* has the inverse relation *the-instructor* with category *COURSE-OFFERING* and the direct relation *works-in* with category *DEPARTMENT*. In addition, a category can have super-category or sub-category. Based on this information, we create the SQL tree structure as shown in section 6.

5.2. The virtual table T(C) for a category C, recursive definition:

1. The first attribute of T:
   - C - attribute of *T*, range: *C (m:1* or many-to-one relationship*)*

2. For every attribute A of T, for every relation r whose domain intersects with the range of A:
   - A_ _r - attribute of T, range: range(r) *(m:1)* provided the depth of recursion does not exceed the system variable $MAXDEPTH. If the original relation r is many-to-many or one-to-many, the new attribute would be many-to-one, but many virtual rows would exist in the table T, one for each instance of the tree. If r has no value for an object, a null value will appear in the virtual relational table. The relation r may be inferred as defined below. The range of a virtual attribute may be of multi-media type: numbers with unlimited varying precision and magnitude, texts of unlimited size, images, etc. The name of T is the same as of C.

5.3. Inferred relations.

- Inverted relations: for every relation R, its inverse is called, by default, R_.
- For every category C, the identity relation, also called C: x.C=if x in C then x else null.
- For every category C, the attribute Isa_C: x.Isa_C=if x in C then "y" else null.
- For every category, a combined attribute C_ _, which is the concatenation of all attributes of C that are represented by printable strings (this includes numbers, enumerated, Boolean). The concatenated values are separated by slashes. Empty strings replace null values.
- Infinite virtual relations representing functions over space time, which in the actual database are represented by a finite data structure.

5.4. Definition of the Extension of a Table

The virtual table T for a category C T(C) is logically generated as follows:

1. Initially, T[C]=C, i.e. T contains one column called C whose values are the objects of the category.

2. For every attribute A of T, for every schema relation or attribute r whose domain may intersect range(A), let R be the relation r with its domain renamed A and range renamed A_ _r. Let T be the natural right-outer-join of T with R. (Unlike a regular join, the outer join creates A_ _r=null when there is no match.)

5.5. User-specified Tables

It is used only by generic graphical user interfaces; not needed by users posing direct ODBC SQL queries.

Let C be a category. Let S={A_1,...., A_k} be some unabbreviated attributes of the table C of type Abstract-object (i.e. no attribute A_i ends with an actual concrete attribute of an original semantic category). (Recall that the name of C is a prefix of each A_i).

We define a virtual table T(S) as the projection of the table C on one of attributes SPP comprised of the attributes S, their prefixes, and done-step extensions of the prefixes.

(An attribute A is a prefix of an attribute in S iff A is in S or A_w is in S for some string w. An attribute B is a one-step extension of an attribute A iff B=A or B=A_ _w where w contains no underscores.)

The name of T is generated as follows: for each A_i let B_i be the shortest synonym of A. The name of T is: B_1__B_2__...__.B_k.

6. SQL Query Generation Algorithm

Writing a semantically and syntactically valid SQL query is not an easy task. Because many users are not exposed to the SQL world and the difficulty of writing SQL statements, we provide a model to construct and generate SQL statements such as INSERT, DELETE, UPDATE, and SELECT. This model is not only capable of generating SQL queries efficiently, it also reduces the client/server communication. In order to reduce the traffic between the client and the server, query construction and generation is handled at the client site such that schema navigation and attribute format validation are supported locally without having clients to consolidate the results with the server. The model works as follow: The client requests the database schema the first time s/he contacts the application server. Upon receiving the database schema from the server, further queries or references to the database schema are handled locally. The server keeps track of connected clients and sends "read invalidate" message to the connected clients whenever the schema is altered. This will cause the client to request a new copy of the database schema. Therefore, the response time is improved because of the reduction of messages that are exchanged between client and server. Users can create queries by interacting with the database schema objects presented to them. After the user is done with the creation of the query, the query is assembled to produce a valid SQL query. The query is then sent to the application server, where query execution is targeted to the database server or engine.

6.1. SQL Tree Construction

Given a Database D, a *Selection Predicate* is a predicate of the form X θ Y, where X is an attribute name, Y is an attribute name or a constant value, and θ is any of the following SQL operators {>, <, ≥, ≤, ≠, LIKE, IN, BETWEEN}. Let S={A_1, …, A_k} be the set of attributes of table T(C). A_i could have a long prefix or path expression E that indicates the path to attribute A_i.

The "_ _" operator in the path expression E invokes a path traversal of an attribute. Path expressions are used to navigate through the object graph. A path expression E is a variable name that represents a category name followed by a sequence of zero or more relation names separated by a sequence of two or three "_" operators. The path expression E comes in two forms:

- For every attribute A directly derived from a category C (A is an attribute of C), then E=<category-name>_ _<attribute-name> or C_ _A.
  For simplification, we can eliminate C and have A only.
- For every attribute A that is derived from a category C along a sequence of one or more relations R, then E=<category-name> <relation-names> <attribute-name> or C_ _R_ _A. For simplicity, we can remove C to obtain R_ _A. For every relation R, its inverse is denoted by R_. Therefore, if R is an inverse relation to attribute A, it is denoted by R_ _ _ A.

Every category C contains zero or more attribute and relation names. During user interaction with the schema objects, a tree is built to construct the SQL that is composed of the category name, attributes list with path expression, and condition on the query. The tree is rooted with the category name. Relations are the intermediate nodes (siblings of the root nodes), and the leaves are the attributes as shown in Fig. 2. The attributes or the leaves of the tree represent two SQL predicates:

- The attribute predicates S (A_i, …, A_k) of the table T(C).
- The condition predicates of the form X θ Y. where X, Y ∈ S.
- A link to a sub-table or a sub-query (for recursive queries).

An interpreter translates the SQL tree to a syntactically correct SQL query that is used as input to the application server, which submits it to the database server.

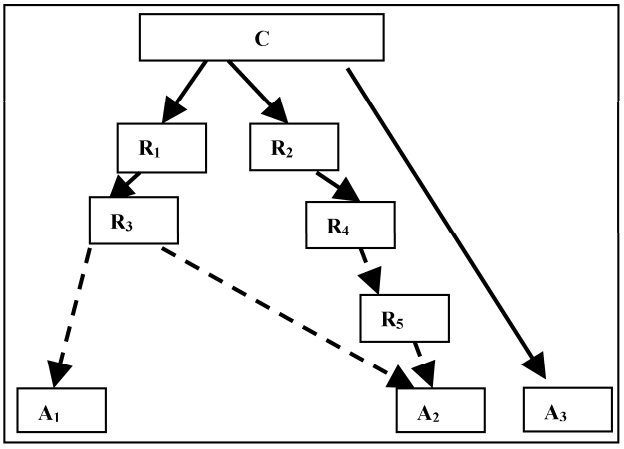

Fig.2. An illustration of the query execution tree

The tree is constructed in the following way:
Initially, a category C is selected. The attributes and relations of the selected category C are expanded. Upon user selection of any of the expanded attributes and/or relations, the selected objects are then added to the tree. Thereafter, if a relation R is selected, the range of the relation is expanded in turn to show the attributes and relations of the category C' (assuming category C has a relationship R with category C'). The database objects are built in the following way:

1. For a given abstract object x, find what category the object belongs to (suppose category C).
2. For a given category C, finds its objects.
3. For a given abstract object x and relation R, retrieve all abstract object y such that xRy.
4. For a given abstract object x, retrieve all of its inverse and direct relationships.
5. For a given relation R and a given concrete object y, find all abstract objects such that xRy.

The goal of this data model or query language is to be able to navigate, query, manipulate, and restructure the graphs of trees. This model is known as "Query by Tree".
Example: Suppose we want to generate the following query (schema is shown in Fig. 3):

select first-name, last-name, major_ _name, the-student_ _ _the-offer_ _the-course_ _name FROM STUDENT where last-name = 'Rishe'. The constructed tree is shown in Fig. 4.

Using the CGI protocol, a server-based protocol, the database schema should be referenced each time the client wants to navigate the schema objects. For instance, to obtain the attributes and the relations of a category, a request should be sent to the database server. The more the user wants to know about any particular object in the schema, the more the database server is contacted. Thus, the network bandwidth is used poorly. In order to reduce the messages between the client and the server, we adopted the Java programming language. Using the Java applets, the user can build SQL queries without having to go along the network to request any further information about the schema objects. This is accomplished by shipping the database schema to the client, where further references to any schema object are done locally. This allows us to reduce computational load for the servers and to implement a very fine granularity, thus improving the overall server performance.

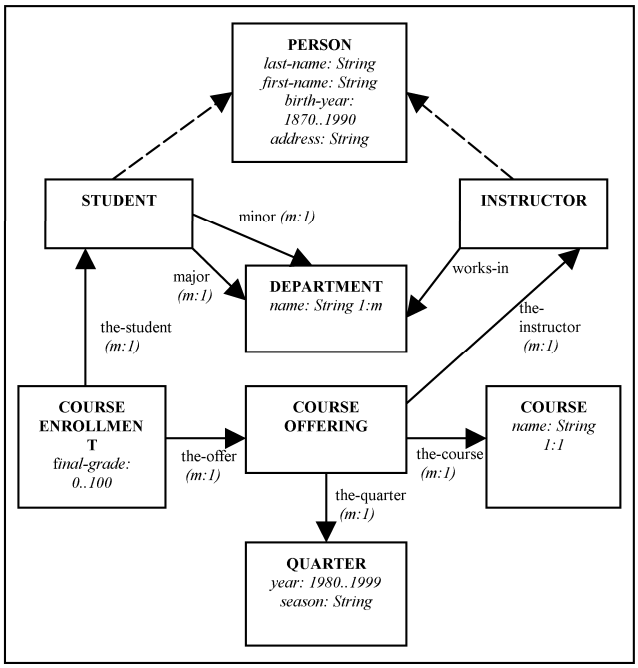

Fig. 3. Semantic schema for the university database.

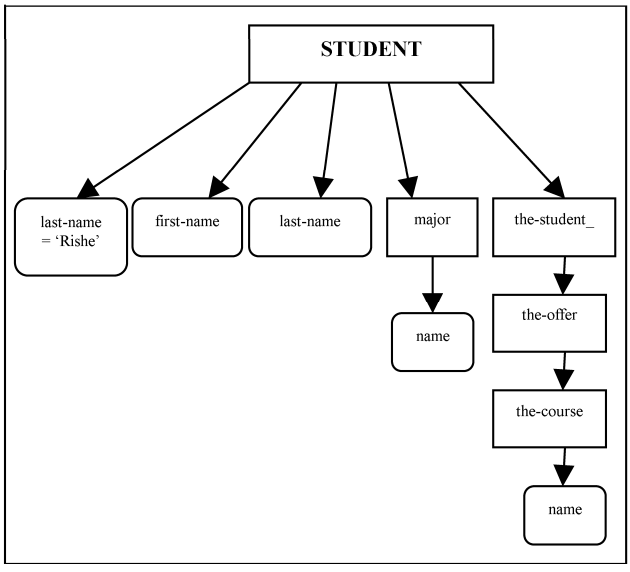

Fig. 4. An example of semantic SQL

7. Client/Server Design

The client SQL query designer guides the user through the process of developing a query. As query building is finished, queries are assembled on the client and then scheduled for execution on the server. The application server then sends the query to the appropriate database server. Thereafter, the application server retrieves the results from the database server, formats the results as a Web page. The application server then delivers dynamically generated reports and pre-assembled reports, served as Web pages in HTML, to the Web client as shown in Fig. 5. If the data being extracted were not just a few rows of data, but several hundred pages, formatting in HTML would be ineffective for displaying the results. Therefore, the application server retrieves the results in chunks, keeping track of the client's transaction and the Session State. This way the client can retrieve and view the SQL query results in chunks. The application server maintains its own knowledge about the state of every client session by keeping track of all actions in the client browser.

The Web was not intended to preserve the browser's state from request to request, other than being able to return to a previously displayed screen or page that has been cached by the browser. The system sometimes depends on information from previous screens, especially when receiving query results in chunks. Generating and preserving the session recent state is one of the main issues when connecting databases to the Web. If the client goes *back* or *forward* from a screen, the *Session State* will be inconsistent with the server associated state. In addition, accessing the databases from the Internet where screens are generated on the fly by the application server presents an issue since the pages are generated dynamically and the status of query execution is not known until its execution is terminated with success. This issue will be addressed thoroughly in a different article.

In summary, Web clients and servers communicate using the HTTP. The basic procedure when the clients communicate with the server is that the client submits a query request to the server. The server sends a response back to the client, supplying the status code as well as the resource requested. The client then disconnects because the HTTP is stateless and connectionless. The Web server cannot originally maintain information about the resource being requested, the client requesting the resource, and the status of the server's response. However, by storing the session state on both the client browser and the server, we can determine if the server can service the request or not by comparing the session states. If the session states do not correspond, an error message is generated and the query's recent status result is forwarded to the client browser.

Data manipulation and data caching are done on the application server, while the client applets provide interactive formatting, visualization, schema navigation, and query generation. Fast response time are achieved because formatted report pages are sent over the network only as needed, which reduces network traffic and the server load.

We use a connection manager to cache and reuse database connections. The connection manager keeps track of clients that are currently connected. The clients' information is placed in a hashed heap table. The connection manager determines when to send a keep-alive message to a client and when to free a connection space. The connection manager spawns a unique thread of execution for each client and thus we achieve maximum concurrency on the server side.

The application or report server and the database server can co-exist together on the same machine or run on separate machines. Coexistence of the report/application server and the database server maximizes the speed of data transfer between the report and the database servers, but it could degrade report and DBMS performance because both servers would compete for resources such as memory and CPU. Use of separate machines gives a server its own dedicated resources, but it requires additional hardware and operating system software. Using separate servers also lengthens the data transfer time between the two servers. Therefore, it would be a good idea to keep the two servers adjacent to each other.

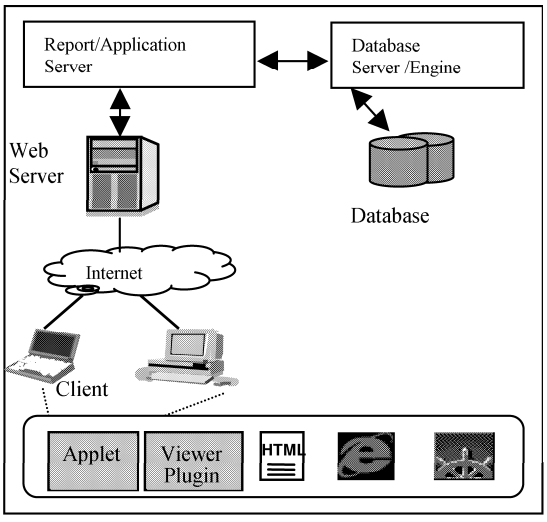

Fig. 5. Client-server architecture of the Web SQL generator

8. Conclusion

This paper presents a model that allows the automatic generation of SQL queries through the Web. The model, known as "Query-by-Tree", shares in concept the "Query by Example" and is intended for novice application developers with limited experience in SQL, and with only a basic understanding of the conceptual database schema. The focus of this application is on query building and information retrieval that are accessible through the Web interfaces. Query generation is the most important aspect and the main emphasis of this paper. The emergence of the Internet provides a tremendous new opportunity for the distribution of corporate data and reports. The Internet can connect disparate data sources with geographically dispersed users [12-13]; a claim that client-server promised but was never able to deliver. The Internet also facilitates the efficient publishing of corporate data to a large number of users, as opposed to the direct access of it by a multitude of individuals. We provide a client-server model to construct SQL queries and to improve the communication between the server and the client.

The client program converts requests into SQL queries, sends them to the application server. Report/application server runs on a second tier, and database server is the third tier. This architecture is intended to allow reports to execute efficiently against large databases and also to return large amounts of data. This reporting model of Internet access to databases can reduce the database impact of large-scale access by making the application and the database servers two separate tiers. We used a message-based architecture between the client and the server with a thin client model that takes advantage of intelligence on the client side.

The application server can be used to dynamically generate HTML reports. Using Java, the processing is distributed between the server and the client using a downloaded applet. Query execution and data manipulation are done on the server while the client applet provides query building features and query validation.

However, with this opportunity comes the risks of security. Database and report access and security must be an integral component of the reporting system. Internet access to reporting environments and databases—and the sensitive data that they often contain— requires security and selected control over database access as well as report production and viewing. Database and network security is not sufficient. This is an ongoing research problem that needs to be fully explored.

9. References

[1]. T.J. Berners-Lee, and D.W. Connoly. "Hypertext Markup Language –2.0.", HTML Working Group of the Internet Engineering Task Force, 1992.

[2]. D.W. Connoly. "HyperText Markup Language." URL, http://www.w3.org/hypertext/WWW/Markup/Markup.html.

[3]. Gettys, and H.F. Nielson. "HTTP-Hypertext Transfer Protocol." URL, http://www.w3.org/pub/WWW/Protocols/.

[4]. John K. Whetzel, "Integrating the World Wide Web and database technology", AT&T Technical Journal, v. 75, pp. 38-46, Mar./Apr. '96.

[5]. S.E. Dossick, and G.E. Kaiser. "WWW Access to Legacy Client/Server Applications." In Proceedings of the Fifth International World Wide Web Conference, Paris, France, 1996.

[6]. S.P. Hadjiefthymiades, and D.I. Nartakos. "A generic framework for the deployment of structured databases on the World Wide Web." In Proceedings of the Fifth International World Wide Web Conference, Paris, France, 1996.

[7]. C. Varela, D. Nekhayev, P. Chandraskharan, C. Krishnan, V. Govindan, D. Modgil, S. Siddiqui, O. Nickolayev, D. Lebendenko, and M. Winslett. "DB: Browsing Object-Oriented Databases over the Web." In Proceedings of the Fourth International World Wide Web Conference: "The Web Revolution", Boston, Massachusetts, 1995.

[8]. D. Flerescu, A.Y. Levy, and A. Mendelzon. "Database techniques for the World-Wide Web: A Survey". ACM the SIGMOD Record, 1998.

[9]. M. Grobe. "An Instantaneous Introduction to CGI Scripts and HTML Forms.", URL, http://kuhttp.cs.ukans.edu/info/forms/forms-intro.html.

[10]. Sun MicroSystems, "The Java Language." Technical Report, Sun Microsystems, 1995.

[11]. Naphtali D. Rishe, "Database Design: The Semantic Modeling Approach"

[12]. D. Lee, D. Srivastava, and D. Vista. "Generating Advanced Query Interfaces." In Proceedings of the 7th International Conference of the World-Wide Web, Australia, April, 1998.

[13]. H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J.D. Ullman, and J. Widom, "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS". In AAAI Spring Symp. On Information Gathering, 1995.