Proceedings of International Conference
on Networking and Multimedia

GWB - A Web/RDBMS Gateway Program Generator

Wei-Jyh Lin, Computer Science Department, National Chengchi University, Taipei, Taiwan, R.O.C.
Kung Chen, Information Management Department, National Taiwan Institute of Technology, Taipei, Taiwan, R.O.C.
Chih-Hung Hsieh, Microtec, a Mentor Graphics Company, 2350 Mission College Blvd., Santa Clara, CA 95054, U.S.A.
C. Jay Chen, Trilogy Technologies, Inc., 3F, No. 3, Alley 181, Section 2, An her Road, Taipei, Taiwan, R.O.C.

Abstract

This paper describes GWB, a WWW to RDBMS gateway CGI generator. GWB is an HTML-centric tool that generates ODBC code from an HTML-like template. It is simple yet powerful enough for most intranet as well as internet applications. This paper surveys the current approaches; describes the design goals, the language, and several intranet and internet applications using GWB. This paper also discusses future enhancement in terms of scalability, session and state management, transaction management, and high-performance.

1. Introduction

World Wide Web (WWW, or Web) browsers have been called "The GUI's of the 90's." The multimedia nature and the input form make Hypertext Markup Language (HTML) [18] a good vehicle for application user interface. The simplicity of HTTP [19] and HTML spurs implement of Web browsers that virtually all platforms have several implementations. The universal interface and availability of Web browsers make them perfect graphical user interface for internet and most intranet IS applications. Using the Web to develop client/server applications involves writing HTML as front-end user interface and Common Gateway Interface (CGI) [20] programs for back-end database access. This paradigm has many advantages over traditional methods of hand crafting user interface and database code using C/C++, or any GUI builder. Among the advantages are quick prototyping, instant deployment, and easy modification.

This paper describes *GWB*, a Web/RDBMS gateway application generator. GWB is an HTML-centric tool that generates ODBC (Open Database Connection) code from an HTML-like template. We describe the design goals, the language, and several GWB applications. Specifically this paper is organized as follows. The rest of this section gives a brief introduction to the mechanism of Web clients accessing databases. Section two surveys current approaches. Section three identifies the GWB design goals. Section four gives a detailed discussion of the GWB language. Section five gives two GWB example applications. Section six discusses future work.

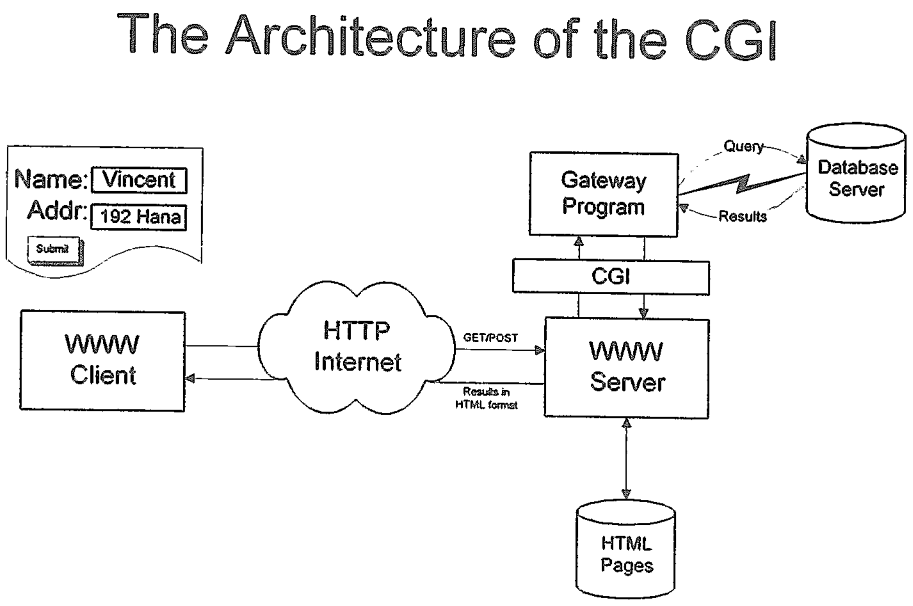

Figure 1

The mechanism that a Web client accessing a database is depicted as in Figure 1. Users submit a form or click on an anchor in an HTML page which represents a user interface to databases. This action triggers the client to send to web servers a GET or POST HTTP message specifying a CGI program to run with arguments from user input. The web server executes the gateway program which is really a database gateway to service SQL requests embedded in input arguments. The database gateway connects to the database server and sends the request for execution. The results are then passed back to clients along the same route, from database servers, to gateways, to web servers, and finally to clients. Other mechanisms of accessing databases from the Web exist. Please see section 2.2 below.

In the above CGI mechanism the database gateway program has to do several tasks: (1) it has to decode the parameters passed from a web server and validate them; (2) it has to compose SQL statements according to input values, including necessary binding of SQL parameters to host variables; (3) it has to performs database connection, sending requests, examining execution status, and retrieving result sets if any; and finally (4) it has to formulate results back into HTML, which includes HTML-escaping database texts, numeric values, and blob data before sending them back to clients.

All the above tasks are tedious and repetitive in developing Web/Database programs. They are low level and should be automated. There is a clear need for Web/Database gateway program generators to automate these tasks.

GWB is designed to automate these tasks. The following diagram depicts GWB architecture. GWB users write HTML-like source programs. GWB first compiles the source programs into ANSI C code. It then invokes C compiler to compile the generated code and link with GWB and ODBC libraries to produce a database access gateway program. A separate HTML form can be authored using any WYSIWYG HTML editor to invoke the gateway program (see Figure 2.)

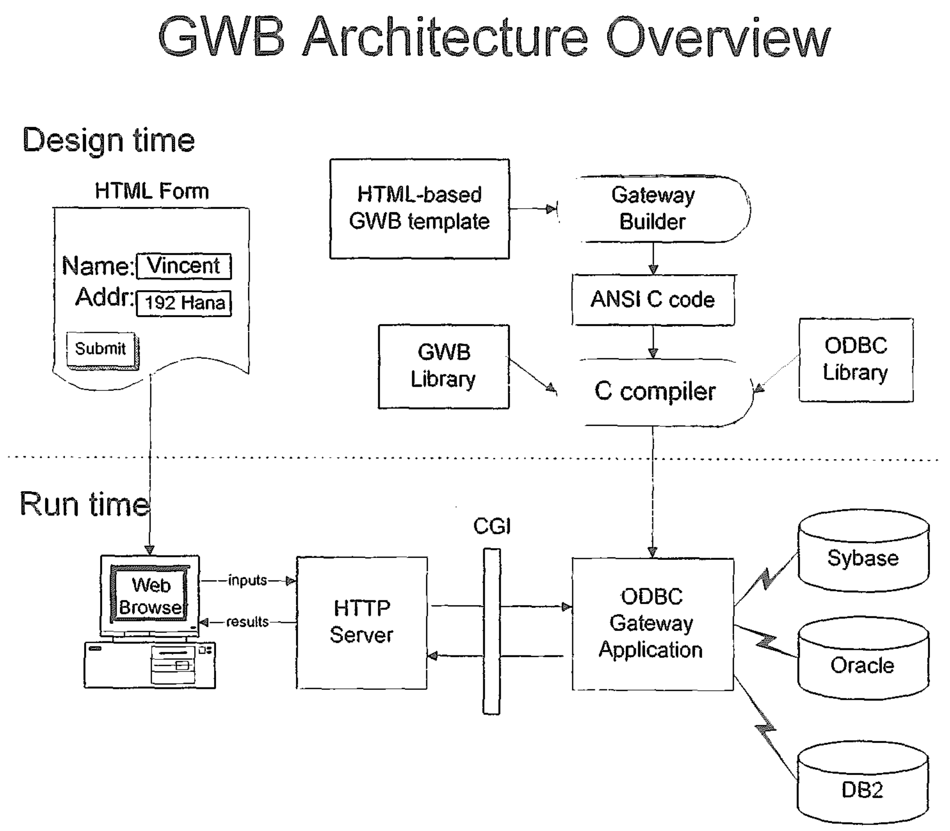

Figure 2.

2. Current approach survey

Since Arthur Secret's work on WWW access to relational databases at CERN in 1992 [1], there has been a growing interest in this area. Individual researchers, database middleware vendors, and database vendors all have their own tools to address this issue. Richmond [2] and Kim [3] maintain comprehensive lists of Web/DBMS tools and products. Frank [4] explores various techniques and products that are used to access databases from the Web. Brian [5] wrote a good tutorial of application development on the Web.

For the purpose of designing a Web/RDBMS gateway program generator we surveyed existing tools and products from three different perspectives: *programming model, architecture, and intended usage.*

2.1 Programming model

By *programming model* we mean the abstract model of a language in which applications are programmed. A language's programming model determines how naturally the intended Web database processing and presentation can be expressed. The language constructs and database access primitives determine how general applications can be written. We classified programming models of existing tools as *HTML-based, Perl-based, Script-based, and other-programming-language-based.*

HTML-based tools generally take HTML as a base language and add a few extended tags for application processing logic. A source program, called a *template*, is simply the intended HTML output with control constructs and SQL statements embedded. Special syntax is provided for variables that act as place holder for parameters passed from Web servers. Variables can be printed as HTML texts, incorporated in SQL statements, or used in branching or looping constructs. These tools also provides mechanism to iterate through and HTML-escape retrieved data before presenting to clients. Since a source template bears the look of HTML documents, this approach turns a Web page into an "*application page*."

This approach has the advantage of simplicity. A WYSIWYG HTML editor can be used to write the base HTML part of a template. Control structures can then be added. The output presentation of an application is thus clearly separated from its processing. Typical tools in this category includes Cold Fusion [6], WebBase [7], and WebDBC [8].

Tools in this category vary in how new tags and variables are introduced syntactically. They also differ in whether general operators and expressions are provided. Limited expressions constraint the scope of applications that can be written. Some tools do not allow multiple SQL statements in one template. Virtually none of these existing tools offer complete data types and database access primitives such as those provided by ODBC API.

The second category includes tools that use Perl as their programming language. Perl-based tools are among the first Web/RDBMS tools. Michael Peppler in Switzerland and Kevin Stock in the UK have developed sybperl [9] and oraperl [10], which are perl implementations of the C library routines for Sybase and Oracle databases. Since Perl has complete control constructs, arbitrary complex Web applications can be written using these libraries. As a result this approach has been very popular in publishing databases on the internet.

Although these tools provides all necessary primitives for writing database applications they are not specifically designed for Web/Database access. The presentation of application output typically disperses in Perl statements. Other tools, such as Sybase's web.sql [11] and WDB [12], removes this drawback by providing another layer's programming paradigm on top of these Perl libraries.

The third category contains script-based tools that define their own specific script languages. These tools generally assume particular access patterns and presentation styles and optimizes the script languages accordingly. For example, Web/Genera [24] assumes that most database access from Web is to query information from a complex database. It thus provides a schema description language that states the information to be displayed, including their data types, column and table name, etc. WDB [12] is another example of script-based Web/Database tool that uses *form definition files* to describe which tables and fields should be accessible through each query form. Writing applications using these tools involves writing script statements to access databases and format results.

Since these tools assumes access patterns and presentation styles, applications that do not follow these patterns or styles would be awkward to implement. These script languages generally do not have primitive control constructs. This further limits the kind of applications that can be programmed. None of these tools provide database access primitives.

Other-programming-language-based tools use Java, C++, etc. as their programming languages. These tools share the same characteristics of Perl-based tools: complete language constructs and database access primitives. These tools, however, differ from Perl-based tools in that they generally have a visual development environment. They are mostly designed to support large scale intranet applications and thus all provide mechanism to maintain state or perform load-balancing. See the following section for tools in this category such as [14], [15], [16].

2.2 Architecture

By *architecture* we mean the execution mechanism and environment of generated applications. The most popular architecture's are CGI, server API, special web server, and three-tier architecture.

CGI is the simplest form of Web/Database applications. A CGI application is simply a program that is forked and executed from a web server. It can be written in any language and virtually every web server supports CGI. There are two approaches of using a CGI architecture. In the first approach the Web/Database tool is an interpreter which is run by web servers as a CGI program. Applications, whose paths are passed in URL query strings, are executed by the interpreter. Template-based and Perl-based tools, such as Cold Fusion and WebDBC, fall into this category. In the second approach the Web/Database tool compiles an application from a source format to a CGI executable. Script-based tools fall within this category.

CGI-based architecture suffers from poor scalability and performance. The performance is not optimal because each time a CGI is requested from clients it has to be forked and executed by web servers. When there are many requests arriving at the same time web servers will be overloaded with CGI programs.

FastCGI from Open Market, Inc. [13] addresses this problem by providing a network protocol library which implements the CGI specification. Instead of running an external process, web servers compiled with this library communicate with a FastCGI program using this network protocol for each client request. This solves the performance issue. The scalability issue is also solved because FastCGI applications can run on multiple hosts which could be different from the one running web servers.

Netscape [21] is the first to pioneer *server API architecture*. The goal is the same as FastCGI: to avoid running a separate process for each client request. However, server API architecture does not seek to preserve the CGI mechanism between web servers and application programs. Instead, application programs are compiled as loadable modules, such as shared libraries and DLL's, that are loaded by, and linked with, web servers at run-time. The performance is improved because each client request becomes an internal function call within web servers. Web.sql falls within this category.

Although server API architecture improves application performance most API's are complex and difficult to use. Since application code is linked with web servers, immature applications can corrupt servers. In addition, applications do not scale up well since they must run with web servers on the same host.

Using special web server is another way to increase performance. Special web servers that can complete application requests, in addition to servicing HTML pages, are used in place of a regular web server. When a request arrives, special web servers determine if it is a request for an HTML page or for an application. In case of the latter they would execute the application as part of server functions without running a separate process. Some HTML-based tools, such as WebBase, fall within this category.

Special web server architecture shares the same problems as server API based architecture: applications and web servers can interact in unpredictable ways, and the applications do not scale up well. Special web server approach is in general easier to use than the complex server API's.

Three-tier architecture has become an increasingly important architecture recently. A three-tier architecture involves web servers and a number of application servers and database servers working together in servicing requests. The application servers and database servers run as daemon processes and wait for requests from web servers. When a request arrives web servers pass the request to an available application server through a small CGI or server API. The application server would run a Java class or interpret a template or a script as specified by the request. For each databases access embedded in the request the application server asks an available database server to access the desired data. The results are then processed by application servers before sending back to clients. NetDynamics [14] and dbKona/T3 [15] are representatives of three-tier architecture.

The three-tier architecture solves performance and scalability problems simultaneously by employing multiple application and database servers. In addition, transaction, session, and state management can all be addressed in this architecture. See section six for a discussion on these issues and how a three-tier architecture solves these problems.

2.3 Intended Usage

Existing Web/Database tools can also be classified according to their intended usage's. The simplest ones are those intended for browsing a database. These tools generally lack the flexibility of building arbitrary queries or presenting results in arbitrary formats. Script-based tools generally fall within this category. The second category contains tools that are designed with internet database access in mind. They are more flexible in building queries and presenting results. Oraywww [22] and Decoux gateway [23] can be considered among this category. Tools in the third category are those that are designed to support intranet client-server as well as internet database applications. WebBase, Cold Fusion, NetDynamics are among this category.

For programming simplicity, GWB uses HTML-based programming model. GWB is designed to support both intranet and internet applications. Although currently GWB has a CGI architecture for fast implementation, it can be changed to use a three-tier architecture in the future without changing programming interface and semantics of existing GWB applications.

3. The GWB design goals

An ideal Web/RDBMS tool should be easy to use without requiring specific programming expertise. It should be designed for both internet and intranet applications. Ideally, it should also scale up without scarifying performance and ease-of-use. Specifically the design goals of GWB are to fulfill the following requirements.

1. Simplicity: Many of the tedious and low-level work should be simplified. For example, a simple mechanism should be provided to access arguments without having to worry about decoding. High-level constructs should be provided to validate user input against various rules. Presenting results should be straight-forward without having to worry about HTML-escaping of data and whether the piece-meal

composition of HTML texts would produce the desired layout.
2. Power: Not only should simple client/server applications be easy to develop but writing complex applications should also be possible. It should be easy to specify multiple complicated queries using user inputs or results from previous queries. Basic constructs, such as branching, looping, and assignment, should be provided for arbitrary processing logic. It should be able to handle various data types in heterogeneous databases. It should also make presenting blob data easy.
3. Extendibility and easy to learn: The provided constructs should be uniform and easy to learn. It should be easy to add new capabilities without conflicting existing features. The architecture should also be scalable for high performance.
4. Flexibility: The generated application should be able to work with any database, web server, and web browser. It should also be easy to bridge to legacy code.

GWB was designed to satisfy these requirements. To free developers from details of presenting results in HTML texts, GWB provides an HTML-based template language. GWB adds only a few extended tags to introduce essential programming structures into templates. Three system predefined record are provided to access program arguments easily in templates. GWB also provides a rich set of data types and operators that, coupled with extended tags, makes complicated database application possible.

GWB has a rich set of functions to encapsulate high-level operations such as data formatting and input validation. It also allows user-defined external functions, which is called in the same way as system functions. GWB generates CGI programs that access database through ODBC libraries, but can be extended to use a three-tier architecture with separate application and database servers without changing semantics. Because GWB generates ODBC codes and follows CGI standards, it works with any RDBMS and any web server.

4. The GWB language

GWB is a template based language for automating Web/database gateway application generation. The language is based on HTML, with eight extended tags introduced to bring database connection and processing logic into a template. Conceptually a GWB template is an "application page" which is executed by a Web server. In reality, a template is translated into efficient ODBC code, which is compiled into a CGI program and executed by a Web server.

GWB introduces variables into templates. A GWB variable is a dollar sign leading identifier, e.g., $customer_id. A variable can be a record of heterogeneous named values. Three system records, $GWB_FORM, $GWB_URL, and $GWB_CGI are provided to access the three sources of parameters into a CGI program. Dotted notation is used to qualify individual components in a record; e.g., $GWB_FORM.customer_id denotes an input form field whose name is 'customer_id'. An unqualified name can represent a member of either $GWB_FORM, $GWB_URL, $GWB_CGI, or a local variable. GWB automatically searches $GWB_FORM, $GWB_URL, $GWB_CGI, and local variables, in that order, for an unqualified name. A database query result is the fourth source of values into a CGI program and is also presented as a record which contains members for query execution status, error, and results.

GWB is a dynamically typed language. A variable can hold values of different types throughout its lifetime. GWB provides twelve primitive data types and two compound types.

GWB also has functions. A GWB function is a dollar sign leading identifier immediately followed by a left parenthesis, zero or more parameters, and a closing parenthesis, e.g., $GWB_is_date_format($datefield). GWB provides a rich set of functions for various HTML formatting, input validation, type coercion, etc. GWB also has a C function hook for legacy code and for extending GWB's capability.

In their simplest usage, GWB variables and function values can be printed to HTML output streams. But GWB offers operators and expressions. Expressions can be composed from variables, literals, functions and fourteen operators. Values of expressions are the sources of processing logic provided by GWB extended tags.

The following sections describe data types, functions, operators, expressions, extended tags and give examples to show the power of this compact language.

4.1 Data types

Many template based Web/Database gateway application generators ignore data types and treat all values as character strings. This approach has many disadvantages:
1. The values retrieved from databases can only be displayed. Only limited operators, such as equality operator, can be performed on retrieved values.
2. It is difficult to handle NULL and BLOB data in databases.
3. Since values are converted to character strings when retrieved from databases, lost of precision might occur if the values are to be inserted back into databases. For example, a floating point value may be truncated to 10 digits of precision silently when retrieved from databases and converted to character

strings. When this value is inserted back to databases it may have less precision than the original value.

4. Due to lack of operators it is impossible to provide embedded SQL level flexibility needed in a complex application.

GWB provides twelve primitive types for database and error values, and two compound types for basic data structure.

4. 1.1 Primitive types

GWB has twelve primitive data types (table 1.) The NULL type has a single value, 'GWB_undefined', which is the undefined value of a GWB variable. Existent GWB_FORM or GWB_URL member whose value is not supplied at run-time has a GWB null value. Retrieved database data whose values are database NULL also has GWB null value.

Error type also has only one value: 'GWB_error'. Referring a non-existent variable, applying operators on wrong types of operands, or calling a function with parameters evaluating to 'GWB_error' results in 'GWB_error'. 'GWB_error' carries an error cause string which is printed by GWB at run-time.

Note the distinction of NULL type and Error type. If a form field is defined but no value supplied at run-time the value is 'GWB_undefined'. If a form field is not defined but referred the value is a 'GWB_error'.

GWB distinguishes between signed and unsigned integer since some RDBMS distinguish between these two types. GWB has Bigint type for 64-bit integer. This data type is for RDBMS on 64-bit architecture's.

GWB has Numeric types for NUMERIC and DECIMAL types in some RDBMS. Since precision and scales in these values may carry special meaning, GWB numeric types keep original database precision and scale for faithful reproduction of values when presented. GWB numeric types can be involved in computation and comparison.

GWB has Date, Time and Timestamp types whose values can come from a database column, or converted from a string value. Date, Time and Timestamp values can only be compared to values of the same type. GWB has blob type for database BLOB (Binary Large Object) data. GWB provides special functions to automatically put blob data into file systems and generate proper (in-line image) links to blob files.

4.1.2 Compound types

GWB provides two compound types for basic data structure: List and Record. A list is an ordered collection of GWB values. A form field with multiple values supplied at run-time has a GWB list type value. These multiple values form the elements of a list. A record is a special list where the elements are paired as (name, value). Each pair denotes a named value within a record. The "name" elements all has string types while the "value" elements may be of any GWB types.

4.1.3 Data type determination

Values come into a GWB application from input forms, query strings, CGI environment variables, or SQL results. Each source introduces a record variable with fields for individual value in the source. Since CGI provides only character string passing between web servers and CGI programs, GWB determines the data type of a variable to be NULL, Integer, Float, String, or List depending on if the value is supplied, the format is appropriate, and multiple values are supplied.

A set of coercion functions are provided for cases when the determined types are not desired. For example, a user input “1996-06-09” for an input field called 'date' would have a String type. It can be coerced into a date type in order to be compared with a date value retrieved from databases. Using a simple rule and a set of coercion functions, instead of a more complicated one to determine more types such as Date, Time and Timestamp, makes GWB a compact language while providing maximal flexibility. For example, the $GWB_date() function is flexible enough to accept date format such as “1996-06-09”, “1996-6-9 ”, “1996/6/9”, etc. This also has the advantage of letting GWB users apply these functions consciously instead of coercing silently.

For variables corresponding to values from a database, GWB is able to query the database for data types through ODBC API. Most appropriate GWB data types would be assigned for such variables.

4.2 Functions

Functions are essential to express abstract operations. They adds extendibility to a language without complicating its syntax. GWB is designed to be a compact language with a rich set of system functions for various needs in typical Web/Database applications. GWB also allows user-defined external C functions. External C functions follow the same calling convention as system functions, e.g., $my_legacy_fcn($GWB_FORM.ship_no). The first function call serves as a forward declaration for the called function. This language design decision not only has the benefit of uniform function interface but also results in simple implementation for system and user functions.

NULL	Error	Boolean	String	Integer	Bigint
Numeric	Float	Date	Time	Timestamp	Blob

Table 1.

Input validation functions		
GWB_is_date_format(expr)	GWB_is_time_format(expr)	GWB_is_timestamp_format(expr)
GWB_is_integer_format(expr)	GWB_is_float_format(expr)	
GWB_name_exist(quoted_name)		
HTML formatting functions		
GWB_format_html(expr)	GWB_format_url(expr)	GWB_format_blob(var, mime)
Coercion functions		
GWB_signed_integer(expr)	GWB_float(expr)	GWB_unsigned_integer(expr)
GWB_string(expr)	GWB_date(expr)	GWB_numeric(expr, pre, scal)
GWB_bigint(expr)	GWB_timestamp(expr)	GWB_list(expr)
GWB_time(expr)	GWB_boolean(expr)	GWB_record(name_expr, value_expr)
Data type validation functions		
GWB_is_signed_integer(var)	GWB_is_unsigned_integer(var)	GWB_is_float(var)
GWB_is_string(var)	GWB_is_numeric(var)	GWB_is_date(var)
GWB_is_bigint(var)	GWB_is_timestamp(var)	GWB_is_list(var)
GWB_is_time(var)	GWB_is_boolean(var)	GWB_is_record(var)
String, list, and record manipulation functions		
GWB_string_length(expr)	GWB_sub_string(expr, i, j)	GWB_list_length(expr)
GWB_list_value_to_string(list_var, separator, pattern)	GWB_instr(str_expr1, str_expr2)	
GWB_record_value_to_string(rec_var, separator)		
Miscellaneous functions		
GWB_resultset_returned(query_var)	GWB_concate_datetime(date_expr, time_expr)	
GWB_sql_table(query_var, max_count, start_row, Next_label, Prev_label, offset, suffix, ...)		

Table 2.

4.2.1 System functions

GWB provides several groups of functions. The first group is for validation. It is essential to validate input values before using them in a SQL statement. GWB provides functions for validating if a variable exists, or if it has the right input format. Since GWB is a dynamically typed language it also provides type validation functions to test the type of variable.

Since most data in databases are not of HTML format they need to be properly HTML-escaped before sent to browsers. A GWB variable is automatically HTML-escaped before output to HTML streams. If this is not desired, for example, if the variable contains HTML texts, $GWB_format_html(expr) can be called to prohibit HTML-escaping. Another function, $GWB_format_url(expr), is provided to perform special URL-escaping, which is needed in the SRC attribute of an anchor or in-line image.

BLOB data generally contain images, audio, video, or documents. Before BLOB data can be sent to clients, proper in-line image tags or anchors need to be generated in HTML output streams. GWB provides a special function, $GWB_format_blob(var, mime), to automatically save a blob variable's data into a file on the server and generate a proper tag with the SRC attribute invoking a *blob handler* program. This program is passed with the saved file name and the mime type and is responsible for sending the BLOB data file to clients and removing the data file after being used.

As mentioned above, it is possible that a variable's data type is not desired and needs to be coerced. Coercion functions are provided for such purpose. One of them is of particular interest: $GWB_List(expr). This function returns a list of a single member which is the value of an expression. Used with the overloaded '+' operator (discussed in a later section,) arbitrary lists can be composed from prime values.

GWB provides several string, list, and record manipulation functions. Arbitrary complex SQL strings can be composed by examining variable values, and use branching statement and the overloaded '+' operator for string composition. Other miscellaneous convenient functions are provided, e.g. one is to automatically formulate the result set of a SQL into an HTML table.

4.2.2 User-defined external C functions

User-defined functions are useful to incorporate legacy code and to extend GWB's capability. They are called identically as system functions. No forward declaration is needed. A set of C utility functions are provided to create, retrieve, and set GWB values. User functions and system functions are treated the same by the GWB compiler. This provides a simple mechanism to link with legacy code. Users can even provide a new implementation and override a system function at link time.

4.3 Operators and expressions

Operators are essential in providing flexibility in applications. GWB provides five arithmetic operators: '+', '-', '*', '-', '%'; six relational operators: 'EQ', 'NEQ', 'GT', 'GE', 'LT', 'LE'; and three logical operators: 'AND', 'OR', 'NOT'. The operator '+' is overloaded for string and list types. Expressions can be constructed by applying operators on variables, literals, and function values. All operands are checked for correct types at run-time. Expressions with incorrect operand types or function parameters evaluate to a GWB_error. Error causes would be printed and the containing statement would be skipped in case of error.

4.4 Statements

Statements are the extended tags that GWB introduces into HTML-based templates. Statements are the missing control and database access operations that are needed to turn an HTML page into an "application page." In addition to database access statements, GWB provides conditional and iterative constructs, which are essential in programming languages. Conditional and iterative constructs in GWB can be nested arbitrarily with virtually no limitations of the nested level.

A GWB template thus can be viewed as a base HTML file with GWB statements intermixed. The part of a template that are not GWB extended tags are also considered as statements in GWB grammar. They are called "non-GWB-phrases." Thus a GWB template is a sequence of GWB statements and non-GWB-phrases.

4.4.1 SET statement

The SET statement corresponds to assignment in imperative programming languages. Assignment bring convenience into a programming language. A variable can be set to the value of any expression. The SET statement has the following form:

```
<GWB_SET var=expr>
```

4.4.2 IF statement

Branching constructs are essential in any programming language. It brings processing logic into a CGI program. Any branch in a IF statement can contains non-GWB HTML phrases or any GWB statement. The IF statement has the following form:

```
<GWB_IF (expr)> {any-statement}*
{<GWB_ELSEIF (expr)> {any-statement}*}*
{<GWB_ELSE> {any-statement}*}
</GWB_IF>
```

4.4.3 FOR statement

Looping constructs are also essential. GWB provides a FOR statement to iterative through all list elements. The body of a FOR statement can also contain non-GWB HTML phrases or GWB statements. The new identifier introduced is only valid within the body. The FOR statement has the following form:

```
<GWB_FOR ident in list_var>
{any-statement}*
</GWB_FOR>
```

4.4.4 SQL statement

The SQL statement brings database connection into a template. The SQL statement has attributes to specify data source, login id, password, and a query string to be executed. It also has a name, which refers to a record that carries the query execution result. The SQL statement has the following form:

```
<GWB_SQL NAME="sql_name" SOURCE=
"Data Source" ID="id" PASSWD="passwd"
QUERY="query-string">
```

The following example select from a 'title' table all rows whose 'author' column starts with the prefix specified in variable $author. A record variable, $search_book, would be produced that contains the executing results.

```
<GWB_SQL NAME="search_book"
SOURCE="INF_BOOK_DB" ID="guest"
PASSWD="public" QUERY="select * from
title where author like '$author%' ">
```

4.4.5 RESULTSET statement

The RESULTSET statement is used to iterate through the result set of a query. The NAME attribute specifies a corresponding GWB SQL statement whose result set is to be iterated through. Since result sets could be very large, attributes are provided to limit the number of rows to retrieve. GWB has the nice mechanism of automatically supplying HTML buttons for scrolling through the result set. Three more attributes are provided to specify the offset between previous and next scroll screens, and the label on the buttons. The RESULTSET statement has the following form:

```
<GWB_RESULTSET NAME="sql_name"
ITERATOR=id MAX_COUNT=int_expr
START_ROW=int_expr OFFSET=int_expr
NEXT="labal-text" PREV="label-text">
{any-statement}*
</GWB_RESULTSET>
```

The following example tests if a SQL statement 'serach_book' was executed successfully. It prints out an error message with status code if the SQL was not executed successfully. When successful, the 'author', 'title', and 'year' column values are printed using iterator 'b':

```
<GWB_IF ($search_book.status NEQ "OK")>
     Database accessing failure :
$search_book.status
<GWB_ELSE>
     <GWB_RESULTSET NAME="search_book"
ITERATOR="b">
     Author: $b.author Title: $b.title
Publish: $b.year <BR>
     </GWB_RESULTSET>
</GWB_IF>
```

4.4.6 Other statements

There are three other statements in GWB. `<GWB_INCLUDE SRC=template_path>` is for template source inclusion. `<GWB_OUTPUT SRC=html_path>` is for inserting an HTML file to output streams at run-time. `<GWB_EXEC COMMAND=host_command>` is for executing an external host command whose output is inserted into the HTML output stream. This is useful for including system related information, e.g., system date and time.

5. Example GWB templates

5.1 Data analysis

This application analyzes CD titles buying group classified by customer ages. The kind of titles to analyze and the classification scales are supplied at run-time from an input form. This shows the flexibility of GWB in data manipulation.

```
<html><head>
     <title>CD buying pattern classified by
age</title></head>
<body>
<GWB_SQL NAME="sel" SOURCE="Sales" ID="xxx"
     PASSWD="yyy" QUERY="select * from
     CD_SALES where kind='$kind'">
<GWB_IF ($sel.status NEQ "OK")>
     Error accessing Database: $sel.status;
<GWB_ELSE>
     <GWB_SET b1=0 b2=0 b3=0>
     <GWB_RESULTSET NAME="sel" ITERATOR="i">
          <GWB_IF ($i.age LT $scale1)>
               <GWB_SET b1=$b1+1>
          <GWB_ELSEIF ($i.age LT $scale2)>
               <GWB_SET b2=$b2+1>
          <GWB_ELSE>
               <GWB_SET b3=$b3+1>
          </GWB_IF>
     </GWB_RESULTSET>
     The $kind CD bought by group 1: $b1<BR>
     The $kind CD bought by group 2: $b2<BR>
     The $kind CD bought by group 3: $b3<BR>
</GWB_IF>
</body></html>
```

5.2 Internet electronic shopping

This application is a typical internet database access example. Here the partial name of an electronic product is entered by users from an input form. This template would retrieve matched items from databases. Audio and picture of the items would be automatically formulated as anchors and in-line images. In case of error a technical person is automatically notified by running a host command to call a beeper number. This example demonstrates that it is very easy to access databases and presenting multimedia information in GWB.

```
<html><head>
     <title>Product Information</title>
</head><body>
<GWB_SQL NAME="sel" SOURCE="Items" ID="xxx"
     PASSWD="yyy" QUERY="select * from Items
     where name='%$name%'">
<GWB_IF ($sel.status NEQ "OK")>
     Sorry, database is inaccessible at this
     moment. Technical persons have been
     notified. Please try again later. <BR>
     <GWB_EXEC COMMAND="beep 9-456-7766">
<GWB_ELSE>
     <GWB_RESULTSET NAME="sel" ITERATOR="i"
          MAX_COUNT=1>
          Product Name: $name<BR>
          Price: $price<BR>
          <GWB_IF ($pic NEQ GWB_undefined)>
               Picture: $GWB_format_blob(
               $pic, "image/gif")<BR>
          </GWB_IF>
          <GWB_IF ($music NEQ GWB_undefined)>
               Music: $GWB_format_blob(
               $music, "audio/wav")<BR>
          </GWB_IF>
     </GWB_RESULTSET>
</GWB_IF>
</body></html>
```

6. Discussion and future work

GWB in its current implementation fulfills its design goals: it is easy to learn and easy to use; it is powerful enough for most internet and intranet applications; it is flexible to work with any database, any web server, and any web browser; it is also extensible by adding more system functions and user-defined external functions easily. The next generation would tackle the problems of scalability, session, state, and transaction management.

To address the scalability issue the next generation of GWB would use a three-tier architecture to distribute the requesting load. One possible approach would configure GWB application servers as server-side Java applications. The GWB templates could be compiled into Java classes, which are dynamically loaded and run by application servers. The protocol between application and database server would be carefully designed to facilitate caching database connections and cursor states within database servers.

The issues of session, state, and transaction management originate from the stateless of HTTP. Two-tier client-server applications are able to maintain states and manage transactions because they constantly maintain a session with the server. This is impossible, at least not efficient, for web applications which use HTTP.

To support session a Web/Database tool needs to provides *virtual session* for applications. A virtual session maintains session related information, such as database connections and user privileges, from web pages to web pages. A popular approach is to generate a unique session key that is kept on the client page with associated data stored on the server. A *session manager* can be added into the three-tier architecture to perform the job of generating session key and storing session data.

GWB in its current implementation supports client state management by the SET command. Local variables can hold the state information and can be carried from page to page using hidden form fields or URL query strings. Server state management can be provided by the session manager which keeps state information along with the session data.

Finally, to support transaction management GWB simply has to add functions that export ODBC's SQLSetConnectionOption and SQLTransact functions. Next generation of GWB will export more ODBC functions, such as SQLTables, SQLStatistics, etc., to extend the range of applications that can be written.

7. Acknowledgment

Many thanks to Addison Lin of Trilogy Inc. for working on the implementation of GWB. He has been very helpful in implementation and testing which lead to early capture of some original design flaws. Thanks also goes to Jichang Tan of NTU for his contribution of the original database access code and the experience of building an earlier version of GWB.

References

[1] W3C, WWW Access to Relational Database, http://www.w3.org/pub/WWW/RDBGate/.
[2] Alan Richmond, Web Developer's Virtual Library: Database, http://www.stars.com/Vlib/Database.html.
[3] Pyung-Chul Kim, WWW-DBMS Gateways, http://grigg.chungnam.ac.kr/~uniweb/documents/www_dbms.html.
[4] Maurice Frank, Database and the Internet, DBMS magazine, December 1995.
[5] Marshall Brian, Web Development Series, Interface Technologies, Inc., http://www.iftech.com/classes/webdev/webdev4/htm.
[6] Cold Fusion Profession 1.5 White paper. http://www.allaire.com/allaire/pages/cfwhiteppr.htm.
[7] WebBase User's manual, http://www.webbase.com/docs/WB23/UserVars.htm.
[8] WebDBC 2.0 Smart Introduction, http://www.ndev.com/qref/default.htm.
[9] sybperl, http://www.sybase.com/WWW/Sybperl/index.html.
[10] Perl and Oracle, http://www.bf.rmit.edu.au/~orafaq/perlish.html.
[11] Sybase's Web tools strategy, SunWorld on line, March 96.
[12] WDB - A Web to Database Interface, http://arch-http.hq.eso.org/bfrasmus/web/intro.html.
[13] Open Market, Inc., FastCGI: A High-Performance Web Server Interface, April 1996, http://www.fastcgi.com/kit/doc/fastcgi-whitepaper/fastcgi.htm.
[14] Spider Technologies, Inc., NetDynamics - Delivering Applications in Web Days instead of Man Year, http://www.w3spider.com/ndwhite.html.
[15] WebLogic dbKona/T3 White Paper, Weblogic Technologies, 1995, http://Weblogic.com/t3.html.
[16] Nick N. Duan, Distributed Database Access in a Corporate Environment Using Java, Fifth International World Wide Web Conference, May 1996, Paris, France, http://www5conf.inria.fr/fich_html/papers/P23/Overview.html.
[17] PageBlazer Overview and Features, http://www.PageBlazer.com/overview/main.htm.
[18] T. Berners-Lee and D. Connolly, Hypertext Markup Language - 2.0, September 22, 1995, http://www.w3.org/pub/WWW/MarkUp/html-spec/html-spec_toc.html.
[19] Basic HTTP, http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html.
[20] The Common Gateway Interface, http://hoohoo.ncsa.uiuc.edu/cgi/.
[21] The Netscape Server API, http://home.netscape.com/newsref/std/server_api.html.
[22] Orawww, http://www.nofc.forestry.ca:80/oraywww/.
[23] Decoux' extension of Sectret's work, INRA, Paris, http://moulon.inra.fr/oracle/www_oracle_eng.html.
[24] What Web/Genera Does, http://gdbdoc.gdb.org/letovsky/genera/genexamples.html.