Accessing Relational Databases from the World Wide Web

Tam Nguyen
IBM Santa Teresa Laboratory
555 Bailey Avenue
San Jose, CA 95161
minhtam@vnet.ibm.com

V. Srinivasan
IBM Santa Teresa Laboratory
555 Bailey Avenue
San Jose, CA 95161
srini@vnet.ibm.com

Abstract

With the growing popularity of the internet and the World Wide Web (Web), there is a fast growing demand for access to database management systems (DBMS) from the Web. We describe here techniques that we invented to bridge the gap between HTML, the standard markup language of the Web, and SQL, the standard query language used to access relational DBMS. We propose a flexible general purpose variable substitution mechanism that provides **cross-language variable substitution** between HTML input and SQL query strings as well as between SQL result rows and HTML output thus enabling the application developer to use the full capabilities of **HTML** for creation of query forms and reports, and **SQL** for queries and updates. The cross-language variable substitution mechanism has been used in the design and implementation of a system called DB2 WWW Connection that enables quick and easy construction of applications that access relational DBMS data from the Web. An end user of these DB2 WWW applications sees only the forms for his or her requests and resulting reports. A user fills out the forms, points and clicks to navigate the forms and to access the database as determined by the application.

1 Introduction

The World Wide Web (Web) is fast becoming the most popular way of accessing the internet due to its easy to use graphical interface and the ubiquitous HTTP communication protocol. Figure 1 illustrates how workstations are connected together using the World Wide Web. Many universities, governmental agencies, and business organizations have already realized that there is an enormous potential in the Web, especially since the internet already has tens of millions of users and continues to grow exponentially in recent years.

Typically, an organization makes itself accessible to the Web public by maintaining a **home page** on a web server that can be accessed from any other location world-wide on the Web using a **uniform resource locator** (URL). For example, the URL for the IBM home page is `http://www.ibm.com`. This home page can be used to provide up-to-date information regarding existing products and services, new products and services, software downloads, as well as to get feedback from the Web public regarding various matters like product support.

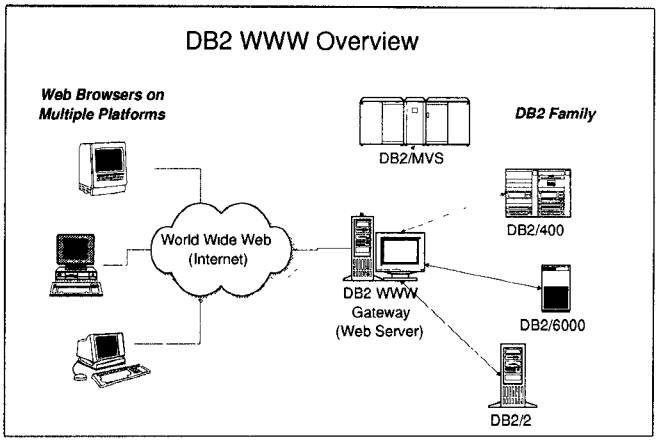

Figure 1: The World Wide Web

Business applications almost always require a database management system (DBMS) for storage and retrieval of the organization's valuable data. More precisely, Web applications for accessing a DBMS typically involve the following steps:

1. Create an HTML[1] fill-in form for the user.
2. Extract user inputs from HTML fill-in forms and access any necessary data from the DBMS (both read and/or update access is possible here).
3. Format the query results into a desirable HTML report form.

[1] HTML (HyperText Markup Language) [LEMA95] is the markup description language of the World Wide Web and every Web page is essentially a static or dynamically generated HTML page.

SIGMOD '96 6/96 Montreal, Canada
© 1996 ACM 0-89791-794-4/96/0006...$3.50

4. Allow for additional accesses to the DBMS and HTML pages, possibly from the hyperlinks embedded in the reports.

So far in our discussion, we have assumed that a Web page accessed using an URL is static (i.e., Web pages are merely files stored in the server location). To implement Web applications that access a DBMS, we need to dynamically create Web pages as the result of the user input and the database access. Indeed, the Web provides a standard protocol for dynamic creation of Web pages called the **common gateway interface** (CGI) [CGI95]. The CGI protocol enables the calling of any executable program recognized by a Web server using the URL syntax. Inputs from the Web client are passed to the executable program, and the program can generate Web pages that are displayed back at the calling Web client. (The CGI interface is described in more detail in Section 2.3 and is illustrated in Figure 4.)

One approach for building a Web application that accesses a DBMS is to implement a stand-alone program that accesses DBMS data and invoke it directly as a CGI application from a URL. This approach has the following disadvantages:

1. the application programmer has to be knowledgeable of the CGI protocol details and the DBMS programming interfaces. The application program is affected by any change in the CGI protocol or the DBMS API.
2. The HTML text is intermixed with complex datastructures and programming logic, making it less readable.
3. Since CGI applications need to produce HTML output, it is not easy to switch to newer HTML versions with many new useful features like HTML 3.0 which is now beginning to be introduced. Changing an application's output will involve changes to the code even though the application logic and database access remain unchanged.
4. Many applications in a client-server environment have little or no application logic – they typically need to generate SQL statements based on user input and execute these SQL statements dynamically against the DBMS. It is not known exactly what (or how many) SQL statements will be executed before the user provides input. To implement applications like these, the formatting efforts might be significant if one needs to write code to generate output forms.

We propose a general purpose solution to build a large class of Web applications that access a DBMS. Our solution has the following characteristics which we feel are necessary in any approach to building Web applications that access a DBMS.

1. New applications must be easy to build, preferably no significant coding effort should be involved.
2. Applications must be easy to maintain and enhance with new HTML versions. In addition they must be shielded from changes to the CGI protocol.
3. The full power of HTML (including the latest versions) for creating input and result forms must be available to the application developer. Ideally, the application developer must be able to use a visual HTML editor to construct the HTML forms.
4. The full power of SQL to access the relational DBMS must be available including using a visual query tool to construct the SQL queries needed to access the DBMS.
5. A mechanism for transferring input variables from the Web client (i.e., the user) to the SQL query (or queries) that is accessing the DBMS.
6. A mechanism for substituting the result of a SQL query into a report form for viewing the result. It must be easy to redesign report formats using new HTML features.
7. Allow for additional (related) queries and HTML forms, possibly from the hyperlinks embedded in the reports.

We propose a general purpose solution for building Web applications that access databases using a page layout paradigm, which encapsulates HTML, the standard markup language of the Web, and SQL, the standard query language of relational databases [SQL92]. To bridge the gap between HTML and SQL, we propose a flexible, general purpose variable substitution mechanism that provides cross-language variable substitution between HTML input and SQL query strings as well as between SQL result rows and HTML output, thus enabling the application developer to use the full capabilities of HTML for creation of query forms and reports, and SQL for queries and updates. The variable substitution mechanism that we describe is quite general and has been already used for other purposes like communicating between HTML and REXX [GERM94] (and can be extended to be used between HTML and PERL [WALL91]). We will focus here, however, exclusively on HTML and SQL.

The cross-language variable substitution mechanism mentioned above has been used in the design and implementation of a system called DB2 WWW Connection that enables quick and easy construction of applications that access relational DBMS data from the Web. The application developer creates HTML forms and SQL commands, and stores them in files (called macros) at the Web server. Embedded variables are used to link the SQL commands and the HTML forms within the same macro. These macros get processed by the DB2 WWW Connection run-time engine. Since DB2 WWW Connection uses native HTML and SQL languages, various visual tools may be used for creation of HTML forms and for generation of the SQL query. An end user of these DB2 WWW applications sees only the forms for his or her requests and resulting reports. A user fills out the forms, points and clicks to navigate the forms and to access the database as determined by the application.

The rest of the paper is organized as follows. In Section 2, the basic CGI architecture is described along with how variable names are passed from the Web client to the CGI application through the Web server. Section 3 describes the cross-language variable substitution scheme. In Section 4 we describe the system, DB2 WWW Connection, that we built using the cross-language substitution scheme. In Section 5, we describe how DB2 WWW Connection handles real world issues like security, multi-lingual Web pages, etc., that need to be addressed while building any application for the Web. Section 6 describes related work in this relatively new area. Finally, in Section 7 we present our conclusions.

2 World Wide Web Fundamentals

Figure 1 illustrates a typical distributed computing system using the Internet to connect client systems executing Web clients (a.k.a browsers) to server systems executing Web servers (a.k.a http daemons). Web clients communicate with Web servers using the http protocol. For the purpose of designing Web applications that access a DBMS, it is necessary to have knowledge of (i) how applications work on the Web, (ii) the methods used to pass inputs from the Web client to the server, and (iii) the support available in the web for writing applications that create a Web page dynamically.

2.1 Steps in Using a Web Application

A Web application basically consists of a sequence of accesses to Web pages based on interactive input from a user. All accesses start with a user providing a URL to a Web client that enables access to a certain Web page on the Web. An application, therefore, consists of the following steps which may be repeated any number of times in a single application.

1. A user fires up a Web client (e.g., Mosaic, Netscape, WebExplorer) and uses it to access a URL.
2. The Web client uses the internet address of a host (and a port number) which is present in the URL to communicate with the Web server at that host and port. The Web client provides the server with the following information:
   (a) the URL itself, portions of which are used by the server to determine the Web page to be returned to the client,
   (b) user provided values for HTML input variables if the URL was present in an already instantiated Web page at the client, and
   (c) other information (e.g., an encrypted password or other security information if the Web client and Web server are operating in a secure mode).
3. The server uses the URL and input variables provided by the Web client to get at a Web page that is shipped back to the client.
4. The Web client parses the Web page received from the server and performs appropriate display operations displaying the page to the user. Note that the server can actually communicate certain special types of data other than HTML to the client (e.g . images, voice, video, and, lately, executable JAVA [GOSL95] byte-code programs). The Web client might use viewers to render such specialized data to the user's screen (e.g., a Postscript [ADOB90] viewer is started if a Postscript file is returned by the server on accessing a URL).
5. The user on viewing the resulting form can start the process all over again by clicking on another hypertext link in the current form

```
<TITLE>DB2 WWW URL Query</TITLE>
<h1>Query URL Information</h1>
<p>
<P>
<FORM METHOD="post"
 ACTION="/cgi-bin/db2www.exe/urlquery.d2w/report">
Please enter a search string:
<INPUT TYPE="text" NAME="SEARCH" SIZE=20>
<p>
Please select what field(s)
to search for the string above:
<P>
<INPUT TYPE="checkbox" NAME="USE_URL"
       VALUE="yes" CHECKED>URL<br>
<INPUT TYPE="checkbox" NAME="USE_TITLE"
       VALUE="yes" CHECKED>Title<br>
<INPUT TYPE="checkbox" NAME="USE_DESC"
       VALUE="yes">Description
<P>
Please select what field(s) to see in the report:
<br>
<SELECT NAME="DBFIELD" SIZE=3 MULTIPLE>
<OPTION VALUE="url">URL
<OPTION VALUE="title" SELECTED>Title
<OPTION VALUE="desc">Description
</SELECT>
<hr>
Show SQL statement on output?
<INPUT TYPE="radio"
       NAME="SHOWSQL" VALUE="YES"> Yes
<INPUT TYPE="radio"
       NAME="SHOWSQL" VALUE="" CHECKED> No
<p>
<INPUT TYPE="submit" VALUE="Submit Query">
<INPUT TYPE="reset" VALUE="Reset Input">
</FORM>
```

Figure 2: A Sample HTML input Form

2.2 HTML Input Variables

An example HTML input form is given in Figure 2 and Figure 3 shows how this form appears to a user on a Web client. This HTML form has `INPUT` and `SELECT` sections which are used to define input variables for user input. The form contains six input variables defined in the various `INPUT` and `SELECT` tags using the `NAME` parameter. The variable `SEARCH` is used to get text input from the user, the variables `USE_URL`, `USE_TITLE`, and `USE_DESC` are used to indicate the three types of searches that can be done, the variable `DBFIELD` is used to enable the user to select a list of things to view, and finally, the variable `SHOWSQL` is used to set a flag. Note that the Web user who is viewing the form (as in Figure 3) does not need to know about the mechanism of setting variables – the Web user merely

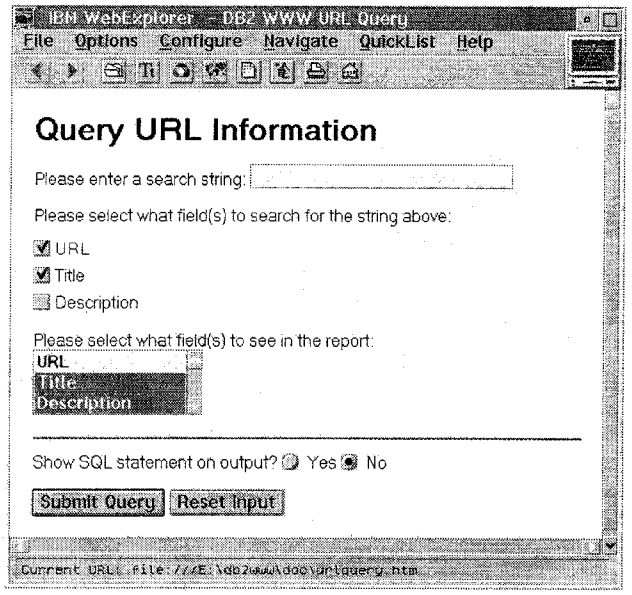

Figure 3: A sample Web Page

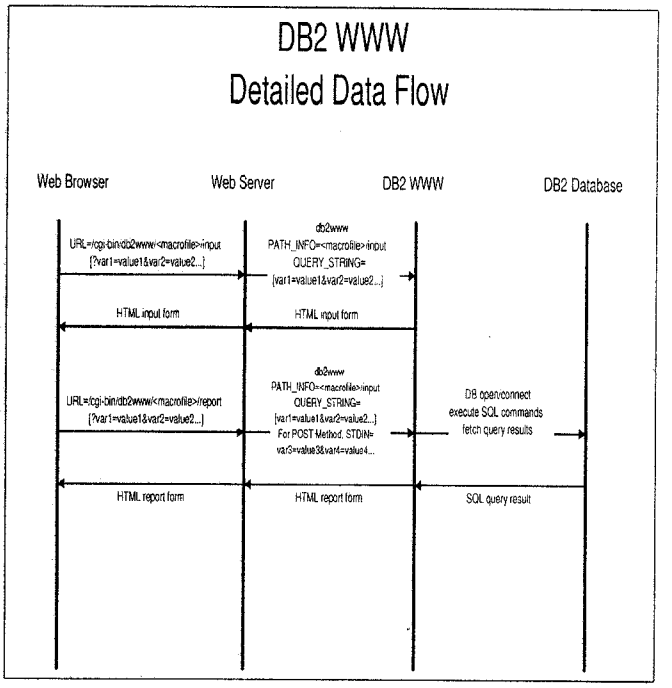

Figure 4: The Data Flow Using the CGI Interface

points and clicks on the various buttons, enters text in the text box and finally submits the query by clicking on the button named `Submit Query`. The Web client will then package the variable values as indicated by the user's screen clicks and passes these onto the Web server. These inputs are sent to the server using a predetermined format. For example, for the selections that the user has made in Figure 3 the following is the value of the input variables that the Web client sends to the server when the user clicks on the button named `Submit Query`.

```
SEARCH    = ""          USE_URL   = "yes"
USE_TITLE = "yes"       USE_DESC  = ""
DBFIELD   = "title      DBFIELD   = "desc"
SHOWSQL   = ""
```

When variables are passed from a Web client to a Web server, the case where a variable is not defined and the case where a variable is defined to have its value as the null string are treated identically. Finally, the variable `DBFIELD` is what we call a **list** valued variable, since the user can make multiple selections on the `SELECT` box to which this variable is attached. When multiple selections are made to `DBFIELD` (as is the case in Figure 3), multiple values for `DBFIELD` will be returned by the Web client to the Web server as shown above.

2.3 Dynamic Generation of Web pages

In order to enable dynamic creation of Web pages, the Web provides the **common gateway interface** (CGI) protocol [CGI95] that can be used by Web users to specify an executable program in the URL. When presented with an URL that contains the name of what is known as a CGI application (i.e., the executable program), a Web server that implements the CGI protocol will start the CGI application as a separate process while passing to this new process the user input that the server received from the Web client along with the URL. In addition, all of the input sent by the Web client to the Web server (discussed in the previous section) is formatted to fit into a string and passed to a CGI application using the `QUERY_STRING` environment variable. The data flow during the CGI protocol is illustrated in Figure 4. In this figure, two different scenarios are shown where an executable program called DB2WWW is invoked twice with different inputs. Note that, in Figure 4, any other executable program can be invoked in place of DB2WWW.

The executable program being run as a CGI application accesses the HTML input variables from the `QUERY_STRING` environment variable performs the application logic including accessing and manipulating any data from the database, and finally generates the output. The output generated by the CGI application is collected by the server and is used to construct the Web page that is sent back to the Web client after the CGI application completes its execution.

3 Cross-Language Variable Substitution

The key challenge in writing Web applications that access a DBMS is to understand both the HTML and SQL languages. In their simplest forms, basic knowledge of SQL and HTML can be easily acquired. However, these languages can be quite complex and tedious to write in order to utilize their advanced functions. Fortunately, there are existing HTML editors

and SQL query tools that can help to greatly reduce the complexity of generating HTML and SQL. Since we wanted application developers to be able to continue using their existing HTML and SQL development tools, we designed a simple macro language that directly includes HTML and SQL sections while tying these two together using a cross-language variable substitution mechanism. The cross-language variable substitution mechanism extends the HTML input variable support described in Section 2.2 by providing a mechanism for defining new variables as well as using the existing HTML input variable support in novel ways to construct DBMS applications.

The macro language itself has the following characteristics:

1. It requires very little extra effort by the application developer other than the use of HTML to create forms and SQL for queries and updates against the database.
2. It is sufficiently flexible for a variety of Web applications that do not require extensive programming logic.
3. It is easily portable to multiple server platforms. In fact, a macro written on one system works as is on another system.
4. It is usable with existing Web HTML editors and DBMS query tools

A macro contains a number of SQL and HTML "sections" tied together via variable substitution. Each macro file typically contains four types of sections:

1. One or more variable definition sections that can be used to define and manipulate variables in the macro.
2. One or more SQL command sections, that each contain one SQL statement as well as (optional) user-defined report formats for the SQL statement.
3. An HTML input section that can be used to get input variable values from the user.
4. An HTML report section that will be used to generate reports from executing SQL statements that are constructed using input variables.

Each section is marked by a reserved keyword with the prefix symbol % (e.g., %SQL), and can contain one or more lines of text. The multiple lines of text are marked enclosed between "{" and "%}". Unless explicitly specified below, section blocks may not be nested. The keywords are **case insensitive** (may be upper or lower case), but the variable names are **case sensitive** except in certain special cases like implicit variables that represent database column names. In the rest of the section, we will describe the macro language features in detail while also illustrating the use of these features using example macros.

3.1 Variable Definition Section

A DEFINE section can be used for one of two purposes: (a) to assign value strings to variables and (b) to define characteristics of a variable (e.g., a variable can be a conditional variable, list valued variable, etc.). Variables are defined in macros using a DEFINE section which contains one or more define-statements that have the following syntax[2]:

```
syntax:
  %DEFINE define-statement
 |
  %DEFINE{
  [define-statement]+
  %}
```

A "define-statement" may be one of four types, namely, a simple assignment, a conditional assignment, a list variable declaration, or an executable variable declaration.

3.1.1 Simple Variable Assignment

A simple variable assignment in a macro is a way to associate a variable name with a value string. Variable names must start with a letter ([A-Z][a-z]) or underscore (_), followed by a variable number of alphanumeric characters or underscore(_). Variable names are **case sensitive**.

```
syntax:
  varname = "value-string-on-one-line"
  varname = {value-string-on
             multiple-lines %}
```

The value string assigned to a variable can contain references to other variables that might be defined in the macro itself or variables that will be input from the Web client using the CGI mechanism. A variable **varname** can be referenced in a value string (as well as in other portions of a macro) using the expression `$(varname)`. When a variable is evaluated to get its value, any variables referenced in its value string are also recursively evaluated to obtain the required value. For example, `%DEFINE var1 = "$(var2).abc"` is permitted. If one wants to get a literal string of the form `$(varname)` to be the value of the variable, then the value should be prefixed with another `$`. For example, `%DEFINE a = "$$(b)"` will result in the variable a being evaluated to the string `$(b)` at runtime. This escape mechanism can be used in extremely useful ways to hide unnecessary information from the user in an application program (for an illustration

[2]The notations used for the syntax descriptions in this document are as follows:

- **UPPERCASE** – keyword
- **lower-case-with-dashes** – a description of what is to be written
- [...] – parts inside [] appear once or not at all
- [...]* – parts inside [] appear zero or more times
- [...]+ – parts inside [] appear one or more times
- **A | B** – choice of one of the items A or B

of this, see the example application in Appendix A). Circular references among variables are not allowed and result in an error.

Simple variable assignments are typically used to set default values for HTML input variables, and also to define variables necessary for database access like the name of the database, the userid, etc.

3.1.2 Conditional Variable Assignment

```
syntax:
  (a) varname = testvar ? "value-string1"
                        : "value-string2"
  (b) varname = ? "value-string"
  (c) varname = testvar ? {value-string1-on-
                           multiple-lines%}
                        : {value-string2-on-
                           multiple-lines%}
  (d) varname = ? {value-string-on-
                   multiple-lines%}
```

The value string assigned to a conditional variable at run time is dependent on whether other variables are defined. In cases (a) and (c) above, `$(varname)` is substituted with `value-string1` if `testvar` exists and is not null, otherwise it is substituted with `value-string2`. The value strings may contain other variable references which are dereferenced when `$(varname)` is processed. In cases (b) and (d) above, `$(varname)` is substituted with `value-string` if this value string does not contain any undefined (or null) variables, otherwise `$(varname)` is set to null.

3.1.3 List Variable Declaration

```
syntax:
  %LIST "value-separator" varname
```

A list variable is declared in the DEFINE section with the list (`%LIST`) property. Multiple value strings assigned to this list variable will be concatenated together with the value-separator in between. Note that the value-separator can in turn contain references to other variables and hence we can have dynamically varying delimiters (An example is to get the delimiter from the user for `AND` or `OR` conditions for performing searches). By default, a multiply assigned variable returned from an HTML form in the `QUERY_STRING` is a list variable with the comma (,) as the list separator (the default is particularly useful for SELECT and FROM clause lists of a SQL query). The conditional and list variables can be used together to construct portions of the SQL clause based on HTML input variables from user selections on an HTML form. An example is shown below:

```
%define{
  %list " AND " where_list
  where_list = ? "custid = $(cust_inp)"
  where_list =
    ? "product_name LIKE `$(prod_inp)%'"
  where_clause = ? "WHERE $(where_list)"
%}
```

In the above example, it is assumed that the variables `cust_inp` and `prod_inp` are HTML input variables that are passed through the CGI interface. When the variable `where_clause` gets evaluated at run-time (run-time variable evaluation is explained in Section 4.3) the variable `where_list` used in the definition of `where_clause` gets evaluated in turn. From the definitions, it is clear that `where_list` is a list variable that is a concatenation of two conditional value strings, the first containing a reference to the variable `cust_inp` and the second containing a reference to the variable `prod_inp`. If the CGI input values are such that `cust_inp = "10100"` and `prod_inp = "bikes"` the variables `where_list` and `where_clause` respectively evaluate to the following strings.

```
custid = 10100 AND product_name LIKE `bikes%'
WHERE custid = 10100 AND product_name
                                LIKE `bikes%'
```

If `cust_inp = ""`, the first value string of the variable `where_list` conditionally evaluates to null and `where_list` itself evaluates to the clause `custid = 10100` (the list variable evaluation is intelligent enough to add delimiters only if the individual value strings are not null). The variable `where_clause` therefore evaluates to `WHERE custid = 10100`. In case both the variables `cust_inp` and `prod_inp` are either not defined or evaluate to null, both value strings of `where_list` evaluate to null and hence `where_list` itself evaluates to the null string. This in turn forces the conditional evaluation of `where_clause` which also evaluates to the null string. In other words, there will be no WHERE clause in a SQL statement constructed using the variable `where_clause`.

3.1.4 Executable Variable Declaration

The execute variable feature allows the invocation of any program from the macro file and passing to it the values of variables defined in the macro.

```
syntax:
  varname = %EXEC "commandstring"
```

The "command-string" gets executed each time `$(varname)` is encountered in an HTML input or output section, including the SQL report block (see Section 3.2.1). The error code, if any, resulting from the execution is returned in `varname`. If there is no error, `varname` will be set to NULL (equivalent to undefined). An executable variable can be used with a conditional variable for printing of error messages (see the DB2WWW Application Developer's Guide [D2W95] for a detailed discussion).

3.2 SQL Section

```
syntax:
  %SQL [(sql-section-name)] {
    any-valid-sql-command
    -on-multiple-lines
    [ %SQL_REPORT{ ... %} ]
```

```
  [ %SQL_MESSAGE{ ... %} ]
%}
```

A macro file may contain multiple SQL sections, with each section containing exactly one SQL command to be executed against the database. A SQL section can be of a line format or a block format (we only discuss block formats here) and each SQL section may optionally be named with a unique `sql-section-name`. contain a valid SQL command on one line. Note that the SQL command string specified by the SQL section can contain variables and therefore the exact SQL string can only be determined at run-time after evaluating the HTML input variables described in Section 2.2. A SQL section block must contain a valid SQL command and may contain a SQL report section (`%SQL_REPORT` block) and/or a SQL message section (`%SQL_MESSAGE`). The SQL commands in the SQL sections are executed when the HTML report section (`%HTML_REPORT` block) is processed (HTML reports are discussed in Section 3.4). **All** unnamed SQL sections are executed by an execute SQL command (`%EXEC_SQL` directive) in the HTML report section, and *each* named SQL section is executed by a corresponding named execute SQL command (`%EXEC_SQL(sql-section-name)` directive) in the HTML report section.

3.2.1 SQL Report Block

A SQL report block may be written inside a SQL section to provide custom report formatting of data resulting from the associated SQL query. The format of the section is defined below:

```
syntax:
  %SQL_REPORT{
    report-header-(any valid html text),
    with column name variables
    resulting from query
    %ROW{
      any-valid-html-text, with column name
      and column value variables as each
      row is fetched
    %}
    report-footer-(any valid html text)
  %}
```

The SQL query is initiated before the SQL report block is processed, and the names of the columns are retrieved. The report header, which is any HTML text in the SQL report block preceding the ROW block (`%ROW` section), will be printed once before the first row of data is fetched. Special report variables for the table are available for use inside the SQL report block for formatting purposes:

1. `Ni` – contains the name of the i^{th} column retrieved from the SQL query,
2. `N_column-name` – is set if a column named `column-name` is retrieved by the query, and
3. `NLIST` – contains a string that is created by concatenating the names of all the columns retrieved.

The HTML text contained in the ROW block is printed out repeatedly as each row is fetched. Just as for the column names, special report variables for each column value are available for use inside the row block for formatting purposes:

1. `ROW_NUM` – contains the current row number being processed,
2. `Vi` – contains the value of the i^{th} column retrieved in the SQL query,
3. `V_column-name` – this contains the value of the column named `column-name` if that column was retrieved by the query, and
4. `VLIST` – contains a string that is created by concatenating the values of all of the columns retrieved.

The report footer, which is any HTML text following the ROW block, will be printed out once after all data rows have been processed. The special variable `RPT_MAX_ROWS` can be used to limit the maximum number of rows to be printed. As seen above, the special variable `ROW_NUM` contains the current row number as it is being fetched. After all rows have been fetched (`%ROW` block has been processed), `ROW_NUM` contains the total number of rows that result from the query, regardless of whether all rows were printed.

3.2.2 SQL Message Section

The SQL message section (`%SQL_MESSAGE`) allows customization of error or warning messages to be printed as a result of a SQL command. For more details, refer to the DB2WWW Application Developer's Guide [D2W95].

3.3 HTML Input Section

```
syntax:
  %HTML_INPUT{
    any-valid-html-text
    -on-multiple-lines
  %}
```

The HTML input form directive contains the HTML form asking for user inputs before generating the query. This section is needed only when user input is required to form the complete query.

3.4 HTML Report Section

```
syntax:
  %HTML_REPORT{
    any-valid-html-text-
    on-multiple-lines
  [ %EXEC_SQL(sql-section-name-or-variable) ]
    any-valid-html-text-
    on-multiple-lines
  [ %EXEC_SQL ]
    any-valid-html-text-
    on-multiple-lines
  ...
  %}
```

The HTML report form section contains the HTML report form for displaying query results. The report form contains the HTML text and execute SQL commands to execute the SQL statements (%EXEC_SQL). When a macro is processed in report mode, the HTML report form is processed. All HTML text in the report section is printed as is, with the variables deferenced to their run-time values. (Note that the run-time values of user inputs from the QUERY_STRING variable override any default settings in the DEFINE sections of the macro).

When an execute SQL command with no SQL section name %EXEC_SQL is encountered in the HTML report section, all unnamed SQL sections are executed sequentially, in the order of appearance in the macro. There can be at most one execute SQL command in the HTML report form. When a named execute SQL command (%EXEC_SQL(sql-section-name)) is encountered, the SQL command in the correspondingly named SQL section (%SQL(sql-section-name)) is executed. The SQL section name for the named execute SQL command may be stored in a variable that gets dereferenced at run time; e.g., %EXEC_SQL($(sqlcmd)) is allowed, where $(sqlcmd) gets deferenced to a SQL section name. This feature can be used to allow the end user to select which SQL command to execute at run time.

The HTML text before and after a %EXEC_SQL directive may contain hyperlinks to other HTML pages or to another macro file. The results from executing the SQL command in a SQL section (each SQL section has exactly one SQL command) are printed in a default table format if no SQL report section exists in the SQL section. If a SQL report section exists, then it is used for printing out the desired format using result variables substitution as specified in the SQL report section.

4 DB2 WWW Connection

Using the macro language described in the previous section as a basis to build Web DBMS applications, we designed and implemented a system that processes these macros and provides support to access a wide variety of DBMS. The system that we built is called DB2 WWW Connection (DB2WWW) and it can be used to access IBM DB2 databases on a wide variety of IBM and non-IBM platforms as well as other non-IBM DBMS on these platforms. The overview of the system environment that DB2WWW executes in is illustrated in Figure 5. As shown in the figure, DB2WWW is invoked using the CGI interface from a Web server using the URL provided by a Web client. (The CGI interface is described earlier in Section 2.3.) DB2WWW can be invoked from an HTML page in one of two ways:

1. <A HREF=
   http:[{web-server}]/cgi-bin/db2www[.exe]/
   {macro-file}/{cmd}[?name=val&...]>

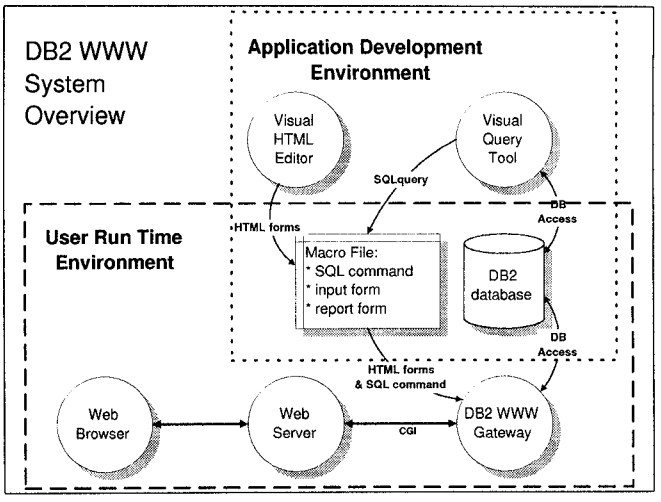

Figure 5: DB2 WWW System Overview

2. <FORM METHOD="post" ACTION =
   http:[{web-server}]/cgi-bin/db2www[.exe]/
   {macro-file}/{cmd}[?name=val&...]>

   {web-server} is the name of web server as defined by the WWW system administrator. This name is optional and the default is the name of the current web server.
   {macro-file} is the name of the file storing the macro defined by the DB2 WWW Connection application developer. As Figure 5 illustrates, existing HTML editors and SQL query tools can be used by an application developer to create new macros.
   {cmd} is either input or report. If cmd is input, the HTML input section of macro-file is processed. If cmd is report, the HTML report section of macro-file is processed.
   [?name=val&...] are optional HTML input variables that may be passed to DB2 WWW Connection from the Web client through the Web server. See Section 2.2 for a discussion of HTML input variables and see Section 2.3 for how these variables get passed to a CGI application like DB2WWW.

When a Web server receives an URL from a Web client like the one described above, it will start the db2www (or db2www.exe) program as a CGI application and pass to it two parameters, namely the values of the {cmd} and {macro-file} variables respectively. In addition, the Web server will pass the HTML input variables and their values to DB2WWW using either the QUERY_STRING interface (case 1. above) or the standard input (case 2. above). (Knowledge of any finer distinctions between these two types of interfaces is not necessary for the purpose of our discussion here and is therefore omitted.) Figure 6 illustrates two calls to DB2WWW once in the input mode ({cmd} = "input") and once in the report mode ({cmd} = "report").

4.1 Macro Processing in Input Mode

When DB2WWW is invoked on a macro in the input mode, it processes only the variable definition sections (DEFINE sections) and HTML input section of the macro (described earlier in Sections 3.1 and 3.3 respectively). The HTML report section and any SQL sections (including SQL message and SQL report sections) are completely ignored (skipped over) by DB2WWW in the input mode.

The variable sections are sequences of define statenments and these are processed and stored in a transient data structure. Note that the right hand side value strings of a variable definition are not evaluated until a variable is (recursively) dereferenced for printing in a HTML input section.

The text in the HTML input section of the macro is output in the same order that it occurs in the macro, i.e., DB2WWW processes macros from beginning to end. Any HTML text that occurs in an HTML input section without any referenced variables is output as is, except for the fact that output patterns which are of the form `$$(varname)` will have their leading $ stripped and appear as `$(varname)` in the output. Any variable referenced (using the `$(varname)` syntax) is substituted using its run-time value; the values of a referenced variable `varname` occurs in the output at the exact position where the string `$(varname)` occurs in the text of the HTML input section. Since macros are processed from the top to the bottom, only variables that were defined in earlier DEFINE sections (or provided through HTML input variables definitions) are recognized for dereferencing in a HTML input section. Note that an undefined variable is not an error, it merely evaluates to the null string. This property is used heavily in formatting reports. See the SQL report section of the example application in Appendix A for an illustration of this type of use.

4.2 Macro Processing in Report Mode

When DB2WWW is invoked in report mode, part of the processing is similar to the processing in the input mode (described earlier) except the HTML report section gets processed here rather than the HTML input section. In fact all of the things discussed in the preceding section are applicable here too except that the output produced in the report mode is based on the text present in the HTML report section. In addition to this, processing a HTML report section involves processing execute SQL statements (`%EXEC_SQL` directives). Each execute SQL statement is processed by processing one or more SQL sections and placing the output of processing the SQL sections at the place in the output report corresponding to the position where the `%EXEC_SQL` directive occurs in the text of the HTML report section in the macro. Exactly which SQL section or sections are processed by an execute SQL statement is determined by the type of `%EXEC_SQL` directive (the three types of execute SQL statements are described in Section 3.2). Executing a SQL command involves the following:

1. Constructing the SQL string to be executed (by dereferencing any variables referenced in the command string) and preparing and executing the SQL command.
2. Create a report for the result of the SQL command. If no SQL report section is available for the SQL section being processed, a default format of the result is printed. If however, a SQL report section exists, then appropriate system supplied variables are instantiated (`Ni`, `N_column-name`, `Vi`, `V_column-name`, etc., described in Section 3.2.1) and the string inside the ROW block is evaluated once for each row of the data retrieved and its output printed.
3. Any error or warning in executing a SQL command is handled by evaluating and printing a warning or error message string defined in a SQL message section, if one exists, or by printing the DBMS error message.

4.3 Runtime Variable Substitution

In a DB2WWW Application, variables can be defined in one of three ways:

1. Variable assignments in a DEFINE section as described in Section 3.1.
2. The NAME parameter of HTML form's `SELECT` and `INPUT` tags. These variables (described in Section 2.2) are set by user inputs or preset by hidden fields in the HTML forms, e.g.,
   ```
   <INPUT NAME="varname" TYPE="hidden"
          VALUE="value-string">
   ```
3. System-defined variables that are automatically set at run-time with the values from the SQL query results (Section 3.2.1).

The key features of the DB2WWW run-time variable substitution mechanism are *lazy evaluation* of variables (the right hand side value strings of variable definitions are not evaluated until the latest possible moment), unifying the name space of the HTML input variables with the variables defined in the macros while giving the HTML input variable values from the Web client (i.e., the user) higher priority than the variable values defined in the macro itself using DEFINE sections.

4.3.1 Lazy Substitution

As previously described, a variable may contain other variables (e.g., `%DEFINE varx ="... $(var2)..."`). Variables are dereferenced (substituted with their values) when they are referenced directly or indirectly in an HTML input or report section, where the values of these variables need to be printed out either for the HTML input form or the HTML report. Variables are not dereferenced at the time of their use in `%DEFINE` or `%SQL` sections. Consider the example below:

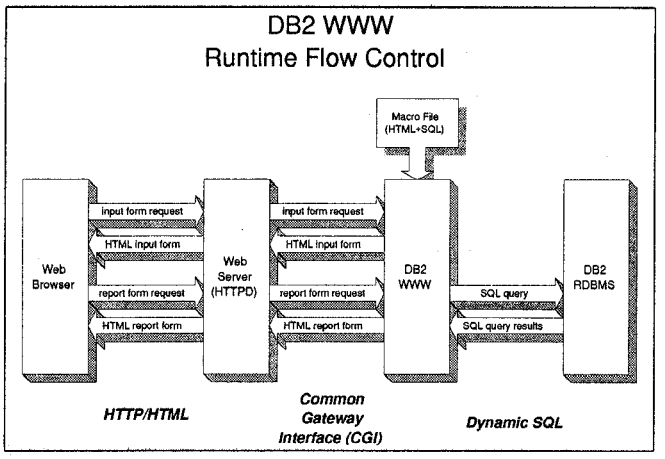

Figure 6: DB2 WWW Runtime

```
%define X = "One$(Y)$(Z)"
%define Y = " Two"
%HTML_INPUT{
    $(X)
%}
%define Z = " Three"
```

Variable X contains references to variables Y and Z. When the HTML input section is processed, Y is already defined, but Z is still undefined and is equivalent to null. Thus, $(X) will be substituted with One Two and not One Two Three. Note that there is a leading blank character in Y and Z.

4.3.2 HTML input variable processing

When DB2WWW is invoked, a number of HTML input variables are passed to DB2WWW using the QUERY_STRING environment variable. DB2WWW treats every var="value-string" that is passed through the CGI interface (see Figure 4 for the format of how variables are passed) as a simple assignment statement (Section 3.1.1) and processes it as such. Since, the value-string of a simple assignment statement can have references to variables, the HTML input variable value can contain references to other variables and hence needs to be parsed before the values can be correctly computed. In addition to simple variables that are passed using the CGI interface, it is also possible to have list variables as HTML input variables (See the end of Section 2.2 for the list variable example.) The default delimiters for list variables is the comma (,), and this can be overridden using the list variable declaration (Section 3.1.3).

The lazy substitution mechanism and the HTML input variable processing features can also be used as a basis for implementing useful application features like hiding variables from the end user, scrollable cursors, and relating multiple client-server interactions on the web as part of the same application [D2W95].

4.4 An Example Application

The DB2 WWW Connection product has been released since November 1995 on multiple platforms and several applications have already been built. We provide the macro file of one such application in Appendix A. The resulting Web page on invoking this macro using DB2WWW in the input mode is shown in Figure 7. The user selections are also shown. When the user clicks on the button named Submit Query on the form in Figure 7, DB2WWW gets invoked in the report mode, the HTML input variables corresponding to the user's selections get passed to it through the CGI interface. DB2WWW executes by reading the macro and the HTML input variables, and processes the HTML report section, performing any SQL queries necessary to generate the report. The resulting report form is shown in Figure 8. The report contains further data specific hyperlinks that the user can click on to proceed further (these hyperlinks could result in further calls to DB2WWW or be any other URL).

5 Some Practical Issues

Since DB2WWW is a fully supported IBM product, we had to tackle various practical issues during the development of the system that are important in developing applications for the Web. These issues include support for large objects, multi-byte character support for international languages, transaction support, and security considerations.

DB2WWW currently supports two transaction modes on a single client-server interaction, one mode in which every SQL statement in a macro is a separate transaction (auto-commit) and another mode in which all SQL statements in a macro are executed as a single transaction (i.e., a rollback will occur if any SQL statement fails). For executing more complex types of transactions, the current variable substitution scheme of DB2WWW enables implementation of a rudimentary scheme for linking multiple client-server interactions. We are working on supporting more complex transaction modes in the future.

While DB2WWW does not provide any new security measure, it works with the DB2 database, the Web server, and the firewall products to provide secure data access over the internet. For additional details on this and other practical considerations please see the DB2WWW Application Developer's Guide [D2W95].

6 Related Work

There have been various efforts, mostly from universities and governmental agencies, to develop tools for creating Web applications that access databases. These efforts look to automate or simplify the application development process.

GSQL [GSQL] uses an intermediate declarative language which is a hybrid of SQL and HTML. The GSQL language is simpler than pure HTML and SQL, and

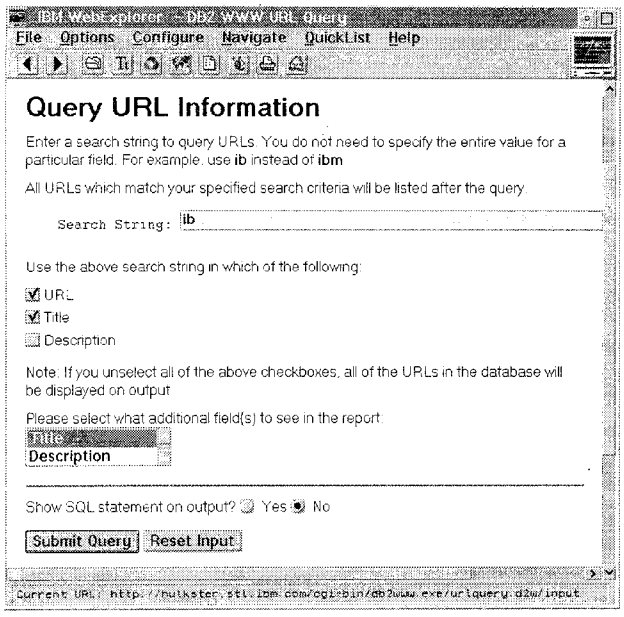

Figure 7: Application Input Form

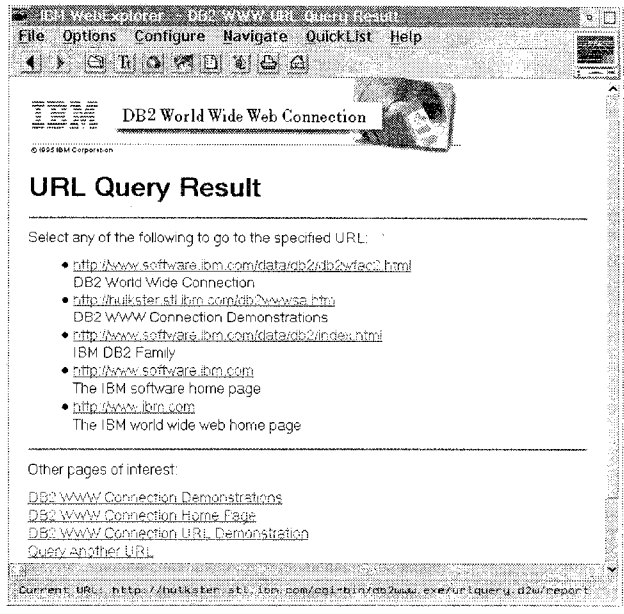

Figure 8: Application Report Form

the new language blurs the line between these two languages. This language, however, is quite restrictive and its method of variable substitution does not allow full use of SQL and HTML capabilities. Furthermore, there is no mechanism defined for custom layout of query reports.

WDB [WDB] contains two components: a form definition file (FDF) generator and the WDB run time engine. The FDF generator extracts table and field definitions from a database to build a skeleton form definition file that contains attributes about the fields. The WDB run time engine automatic generates the HTML query forms, the SQL query, and the report forms based on the FDFs. While the FDF generator provides a quick and easy way to build simple query and report forms to navigate the database, the FDF files contain no information about the input/output form layout. Besides, WDB has very limited limited query and report form building capabilities.

General purpose interpreted scripting languages, such as Perl [WALL91] and Rexx [GERM94], can be extended to support calls to the databases. Perl or Rexx provides the full power of a programming language but Web application development using these languages requires extensive programming and also knowledge of the procedural interfaces.

In Oracle's PL/SQL [PL/SQL], a new mechanism is provided to send the HTML output from the PL/SQL stored procedure back to the Web CGI's output stream. For the programmer who is already familiar with PL/SQL, the new library routines provides a simple way to output results into HTML pages for building Web applications. However, building applications require extensive programming (as in the scripting languages described above), and the PL/SQL language is primarily limited to Oracle databases.

7 Conclusion

We have described in this paper a new, easy to use method of developing applications on the World Wide Web that access data stored in commercial relational DBMSs. The basis of our solution is a novel cross language variable substitution scheme between HTML and SQL. Based on this scheme, we have designed and implemented a system called DB2WWW that has already been released on the Web (in beta versions). The power and ease of use of the general purpose cross language substitution scheme described in this paper can be attested to by the fact that applications are already being built using DB2WWW by scores of application developers on the Web.

The most interesting feature of the cross language variable substitution scheme is its full support for current (and future) versions of HTML and SQL. This feature makes our scheme extremely attractive for application developers well versed in SQL and HTML, since we support these languages in their native form in our system. The incremental work needed for application developers to learn the macro substitution mechanism is rather small and requires no coding at all. The advantage of our solution stems from the fact that the full power of HTML is available for designing input and report forms and the full power of SQL is available for accessing and manipulating data in relational DBMS. In the future, we plan to use DB2WWW's page layout and variable substitution approach for building Web applications for databases

and processing engines other than DB2.

References

[**ADOB90**] Adobe Systems, "Postscript Language Reference Manual", *Addison-Wesley Publishers,* **ISBN 0-201-18127-4**, 1990.

[**CGI95**] "The Common Gateway Interface", *University of Illinois, Urbana-Champaign,* **http://hoohoo.ncsa.uiuc.edu/cgi/overview.html**, 1995.

[**D2W95**] "DB2 WWW Connection Home Page", *IBM Corporation,* **http://service.software.ibm.com/pbin-usa-demos/getobj.pl?/demos-pdocs/wwwdb2dnld.html**, 1995.

[**GERM94**] German, H., "OS/2 2.1 Rexx Handbook", *Van Nostrand Reinhold,* **ISBN 0-442-01734-0**, 1994.

[**GOSL95**] Gosling, J., and McGilton, H., "The Java Language Environment: A White Paper", *SUN Microsystems,* **http://www.javasoft.com/whitePaper/javawhitepaper_1.html**, 1995.

[**GSQL**] Eng, J., "GSQL Database Gateway", *NCSA,* **http://www.ncsa.uiuc.edu/SDG/People/jason/pub/gsql/starthere.html**, 1994.

[**KERN88**] Kernighan, B., and Ritchie, D , "The C Programming Language", *Prentice-Hall Publishers,* **ISBN 0-131-10163-3**, 1988.

[**LEMA95**] Lemay, L., "Teach Yourself Web Publishing with HTML in a week", *Sams Publishing,* **ISBN 0-672-30667-0**, 1995.

[**MOSA95**] "Mosaic for X version 2.0 Fill-Out Form Support", *University of Illinois, Urbana-Champaign,* **http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html**, 1995.

[**PERO95**] Pero, C., "HTML FORMS Tutorial", *University of Illinois, Urbana-Champaign* , **http://robot0.ge.uiuc.edu/ carlosp/cs317/cft.html**, 1995.

[**PL/SQL**] "PL/SQL Web Extensions", *Oracle Inc.,* **http://www.oracle.com**, 1995

[**SQL92**] "Database Language SQL", *ISO-ANSI,* **ISO/IEC 9075**, 1992.

[**STEI95a**] Stein, L. D., "How to Set up and maintain a World Wide Web Site: The Guide for Information Developers", *Addison-Wesley Publishers,* **ISBN 0-201-63389-2**, 1995.

[**STEI95b**] Stein, L. D., "The World Wide Web Security Frequently Asked Questions", *Massachusetts Institute of Technology,* **http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq**, 1995.

[**STRO93**] Stroustrup, B., "The C++ Programming Language", *Addison-Wesley Publishers,* **ISBN 0-201-12078-X**, 1993.

[**WALL91**] Wall, L., "Programming PERL", *O'Reilly & Associates,* **ISBN 0-937-17564-1**, 1991.

[**WDB**] Rasmussen, B., "WDB - A Web Interface to SQL Databases", *European Southern Observatory,* **http://arch-http.hq.eso.org/bfrasmus/wdb/wdb.html**, 1994.

A An Example Macro File

```
%define{
DATABASE="CELDIAL"
dbtbl = "urldb"
%LIST " OR " L_INFO
L_INFO = USE_URL ?
  "$(dbtbl).url LIKE '%$(SEARCH)%'" : ""
L_INFO = USE_TITLE ?
  "$(dbtbl).title LIKE '%$(SEARCH)%'" : ""
L_INFO = USE_DESC ?
  "$(dbtbl).description LIKE '%$(SEARCH)%'" : ""
WHERELIST = ? "WHERE $(L_INFO)"
%LIST ", " DBFIELDS
D2 = ? "<br>$(V2)"
D3 = ? "<br>$(V3)"
%}

%SQL{
SELECT url, $(DBFIELDS)
FROM   $(dbtbl) $(WHERELIST) ORDER BY title
%SQL_REPORT{
Select any of the ... to the specified URL:
<UL>
%ROW{ <LI> <A HREF="$(V1)">$(V1)</a> $(D2) $(D3) %}
</UL>
%}
%}

%HTML_INPUT{
<TITLE>DB2 WWW URL Query</TITLE>
<IMG SRC="/icons/head1.gif">
<H1>Query URL Information</H1>
<P> Enter a search URLs ...  listed after the query.
<P>
<FORM METHOD="post"
 ACTION="/cgi-bin/db2www.exe/urlquery.d2w/report">
Search String: <INPUT NAME="SEARCH" VALUE="ib">
Use the above search string in which of the following:
<INPUT TYPE="checkbox" ...> URL<BR>
<INPUT TYPE="checkbox" ...> Title<br>
<INPUT TYPE="checkbox" ...> Description
<P> Note: If ...  in the report:<BR>
<SELECT NAME="DBFIELDS" SIZE=2 MULTIPLE>
<OPTION VALUE="$$(hidden_a)" SELECTED>Title
<OPTION VALUE="$$(hidden_b)">Description
</SELECT> <P> <HR>
Show SQL statement on output?
 <INPUT TYPE="radio" NAME="SHOWSQL" VALUE="YES"> Yes
 <INPUT TYPE="radio" NAME="SHOWSQL" VALUE=""> No
<INPUT TYPE="submit" VALUE="Submit Query">
<INPUT TYPE="reset" VALUE="Reset Input">
</FORM> <HR>
Other pages of interest:
...
%}

%DEFINE{
hidden_a = "title"
hidden_b = "description"
%}
%HTML_REPORT{
<TITLE>DB2 WWW URL Query Result</TITLE>
<IMG SRC="/icons/head1.gif">
<H1>URL Query Result</H1>
<HR>
%EXEC_SQL
<HR>
Other pages of interest:
...
%}
```