WebSiteGen2: Web–Based Database Application Generator 2

By
Seikyung Jung

Major Professor: Toshimi Minoura
Minor Professor: Bella Bose
Committee Member: Martin Erwig

Department of Computer Science
Oregon State University
Corvallis, OR 97331

June 19, 2001

Acknowledgements

I would like to thank Dr. Tohsimi Minoura, who gives me an opportunity to work on this project and for his advice and encouragement.

I would also like to thank Dr. Bose and Dr. Erwig for their support in the completion of my master degree.

I thank my family who support and encourage during graduate program. My thanks go out to all the friends for their help.

WebSiteGen2 : Web-Based Database Application Generator 2

Seikyung Jung
Dept. of Computer Science
Oregon State University
Corvallis, OR 97331
jung@cs.orst.edu

Abstract

In this project we implemented WebSiteGen2, which is a software tool that automatically generates HTML pages and server-side scripts for a Web-based database application. A user of WebSiteGen2 can select the tables and columns for which HTML pages and server-side scripts are generated. The menus for this selection process are created from the information stored in the system catalog of a database. Our software tool thus simplifies the implementation of a Web-based database application.

Contents

1 Introduction

With the rapid development of the Internet, it has become important to be able to develop Web-based applications quickly. The database for a large application may have more than over hundred tables, and several forms are needed to access and update each table. In a conventional development method, programmers write these forms manually, spending many hours.

WebSiteGen2 is a software tool addressing this issue. It generates server-side scripts from the information stored in the system catalog of a database. WebSiteGen2 displays the table names and the column names retrieved from the system catalog. Then the user can select the tables and columns that he wants to be included in each form.

WebSiteGen2 generates a set of four forms, which we call a *form set.* for each group of tables. Each form set consists of an *insert-search* form, a *select* form, an *update* form, and a *result* form. By using these four forms, we can perform `insert`, `search`, `select`, `update`, and `delete` operations.

Each form contains a *main area* and a *nested area.* We call the table whose information becomes the focal point of each form an *anchor table.* The main area displays the information on an *anchor record,* which is a record in the anchor table, and that on the records *appended* to the anchor record. A *nested area* displays the information on the *expanded records* and on the recordes appended to the expanded records.

Tables associated with the anchor table by many-to-one relationship types can be *appended tables,* and tables associated with the anchor table by one-to-many relationship types can be *expanded tables.* In this report we give the rules for determining which tables can be *appended* or *expanded.* The graphical user interface of WebSiteGen2 allows the user to select with a tree menu the tables that are appended or expanded.

The information needed to generate forms and server-side scripts are retrieved from tables `sysobjects` and `sysforeignkeys` in the system catalog. Table `sysobjects` contains information on the tables in the database, and table `sysforeignkeys` contains that on the relationship types between the tables as *foreign key constraints.*

Forms have been used to display and update data stored in a database. *Oracle Developer/2000 Forms* [9] can be used to create a client-server application that allows end-users to interact with an Oracle database through forms. It also generates the SQL and PL/SQL statements needed by such forms. Each form generated by *Oracle Developer/2000 Forms,* however, can include columns of only one table, and hence appending

and expansion of tables are not supported. Also, the system does not support automatic creation of a Web application.

Microsoft Access 2000 [5], on the other hand, supports automatic generation of a *data access page*, which is a Web page that can be connected to a Microsoft Access database. With a data access page, the user can retrieve and update data across the Web. A data access page may contain ActiveX controls in addition to statements in Cascading Style Sheets (CSS), Extended Markup Language (XML), and Vector Markup Language (VML). An ActiveX control is used to access a database. Each data access page can include columns of multiple tables that are connected with relationship types between the tables. A data access page, however, does not support a nested area with multiple records from an expanded tables are displayed.

With WebSiteGen2, we can generate Web pages that can access Microsoft SQL database with Active Server Pages (ASP). The Web pages generated can cover multiple tables, some in nested form.

In Section 2, we provide an overview of the WebSiteGen2. Section 3 presents the design of WebSiteGen2. Conclusions and some future research topics are provided in section 4.

2 Overview of WebSiteGen2

The architecture of WebSiteGen2 is shown in Figure 1. By using the information stored in the system catalog of a database, WebSiteGen2 generates HTML forms and server-side scripts as ASP pages for a web-based database application. The HTML forms can be used to insert data into or retrieve data from the database. The ASP pages, which are executed by the IIS Web Server, interact with an MSSQL database and dynamically generate HTML pages for displaying or updating the application data stored in the database.

2.1 SISL Database

In explaining the functionalities of WebSiteGen2, we use as an example the *Surface Irrigation Soil Loss* (SISL) database, whose schema is shown in Figure 2. The SISL

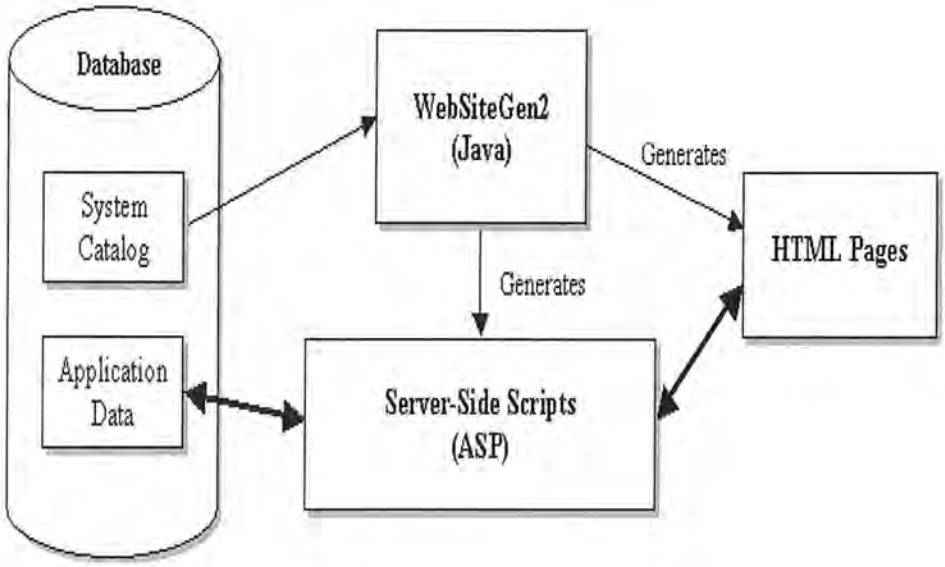

Figure 1: Architecture of WebSiteGen2.

application is used to predict the soil loss for surface irrigated croplands.

The SISL database is designed to store information related to fields. Besides storing primitive attributes of each field, such as the length and the width, the database also stores information about the soil composition of the field and the crop rotations. The information about the crops in each rotation and the conservation practice deployed for each crop is also maintained. All this information is necessary to calculate the soil loss.

The database also maintains the information on the cooperators who own fields, the districts to which cooperators belong, and the conservationists who advice cooperators. Each cooperator may own multiple fields, and hence there exists a one-to-many relationship type between table `Cooperators` and table `Fields`. A district may have miltiple cooperators. Therefore, a one-to-many relationship type between table `Districts` and table `Cooperators` is defined.

Table `Fields` stores the attributes of the fields. Each field may have several rotations. A field belongs to one or more soil map units, whose information is stored in table `SoilMapUnits`. Also table `Fields` has a many-to-one relationship type with table `SlopeCodes`, in which the slope codes used by fields are defined.

As stated above, a field may belong to one or more soil-map units. More than one field may belong to one soil-map unit. The table `FieldToSoilMapUnits` represents from this many-to-many relationship type between fields and soil-map units. Table `SoilErodibility` holds the adjustment value for each K (soil erodibility factor) value.

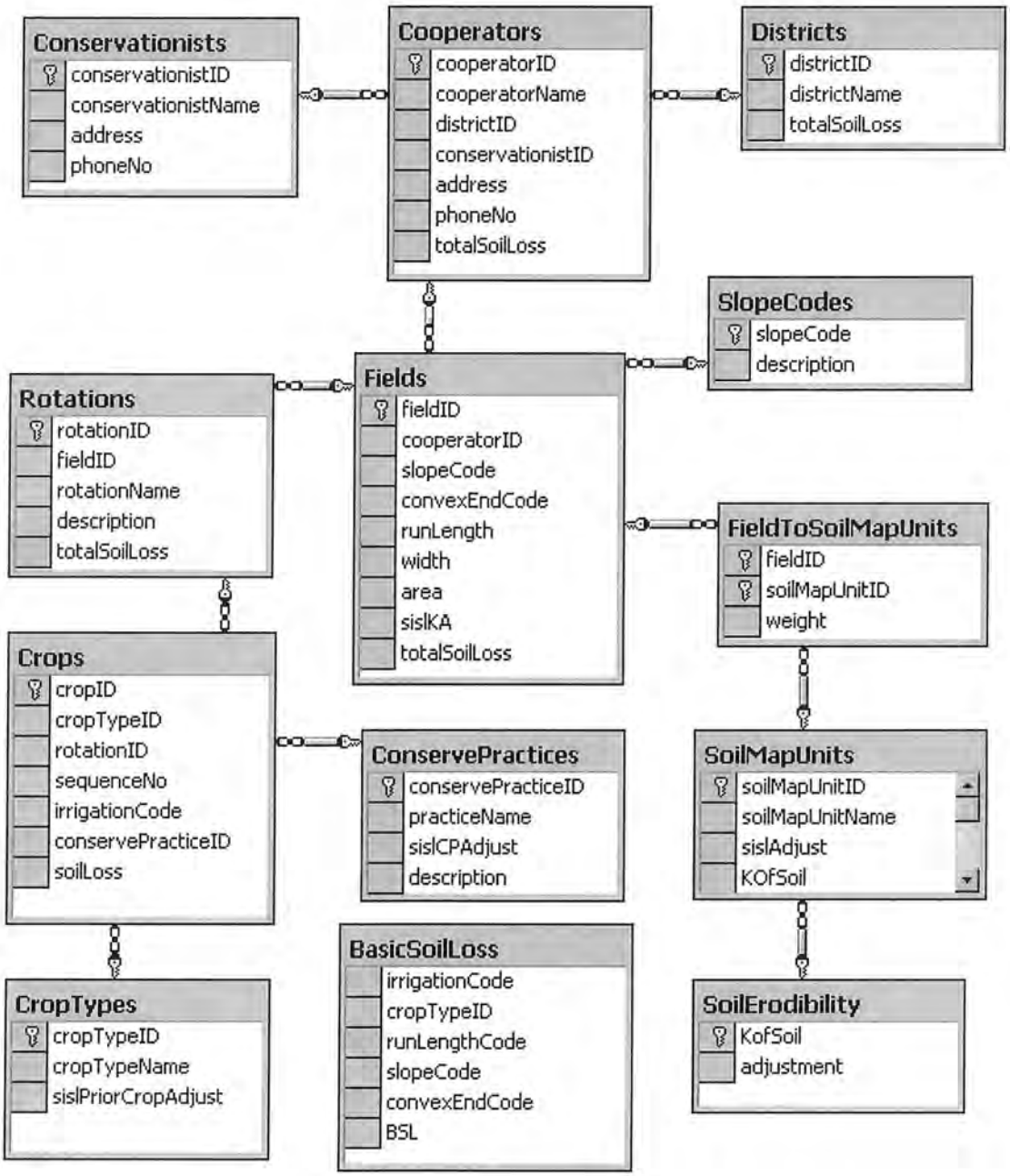

Figure 2: SISL database schema.

Table `Rotations` stores information about rotations employed on a field. Thus there exists a many-to-one relationship type from table `Fields` to table `Rotations`. The information about a particular crop cultivated in a rotations is stored in table`Crops`. A rotation usually includes several crops, and hence there is a one-to-many relationship type between table `Rotations` and table `Crops`. Table `CropTypes` stores data about the crop types, such as wheat and corn. A many-to-one relationship type is defined between table `Crops` and table `CropTypes`.

Table `ConservePractices` stores information on the conservation practices employed by crops. Several crops may use the same conservation practice. Thus, there exists a many-to-one relationship type from table `Crops` to table `ConservePractices`. Table `BasicSoilLoss` stores the information for the soil loss value for each combination of properties of a field.

2.2 Form Set

To access and update a database, we must perform `insert, search, select, update`, and `delete` operations on the entries of each table. WebSiteGen2 performs these operations with a set of four forms, namely, an *insert-search* form, a *select* form, an *update-delete* form, and a *result* form, for each group of tables.

The form `searchCooperators` given in Figure 3 is the insert-search form for the cooperator information. This form is used to insert a cooperator record into the database and to search for cooperator records. To insert the information about a cooperator, the user needs to fill in the fields of the form and press the `Insert` button. To search for cooperators, the user enters partial information on cooperators as the search condition and presses the `Search` button. Then all the cooperator records that satisfy the search condition are retrieved. The result of this search is displayed as a select form `selectCooperators` as given in Figure 4.

The form `selectCooperator` can be used to select a particular cooperator record, which the user wants to inspect, modify, or delete. The user can select a particular cooperator from the list of cooperators by clicking on the radio button associated with that cooperator and pressing the `Select` button. Then the detailed information on the selected cooperator will be displayed with the update-delete form `updateCooperators` as shown in Figure 5.

Once the information about a cooperator is displayed with the `updateCooperator`

form, the user can update or delete the information of that cooperator.
The `updateCooperator` form has a *main* area and a *nested area*. In the main area, the information about the current cooperator, the district to which the current cooperator belongs, and the conservationist who advises the current cooperator is displayed. We call the record that is used as the starting record, the current cooperator record in this example, an *anchor* record. Note that one district and one conservationist are associated with a cooperator. The information about the fields that belong to the current cooperator is displayed in the nested area. Since multiple fields are associated with one cooperator, the field information must be displayed as a nested table.

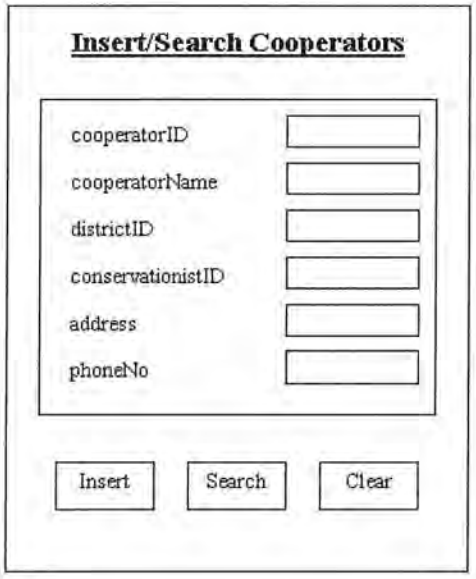

Figure 3: Form `searchCooperators`.

2.3 Grouping Rules

We now describe the rules for determining the set of tables whose entries can be displayed in one form. For example, the `updateCooperators` form can display the information contained in table `Cooperators`, `Districts`, `Conservationists`, `Fields`, and `SlopeCodes`.

Assume that a form displays information on a record of table A and that there is a relationship type between tables A and B.

Rule 1: When the relationship type between tables A and B is one-to-one or many-to-one, the record in table B associated with the record in table A can be *appended* to

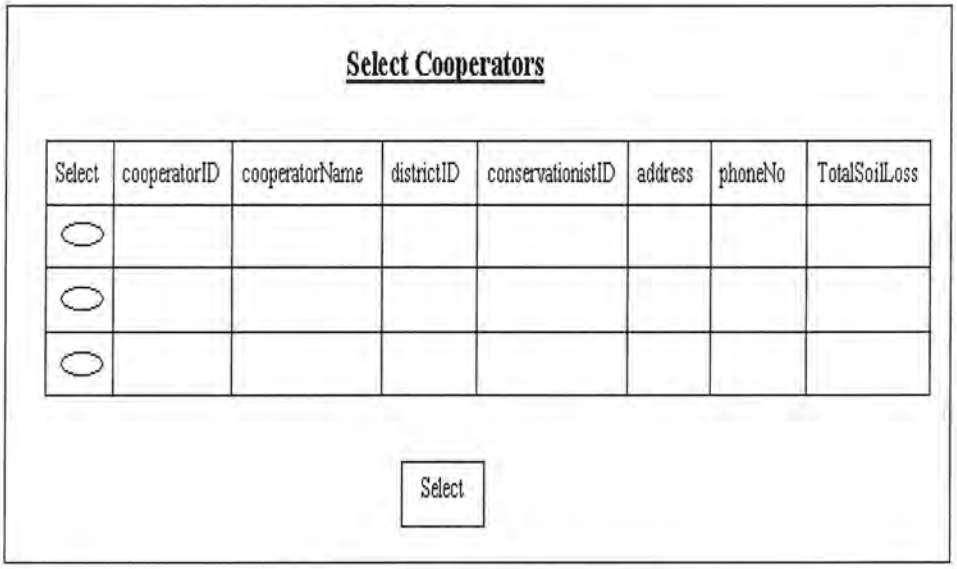

Figure 4: Form `selectCooperators`.

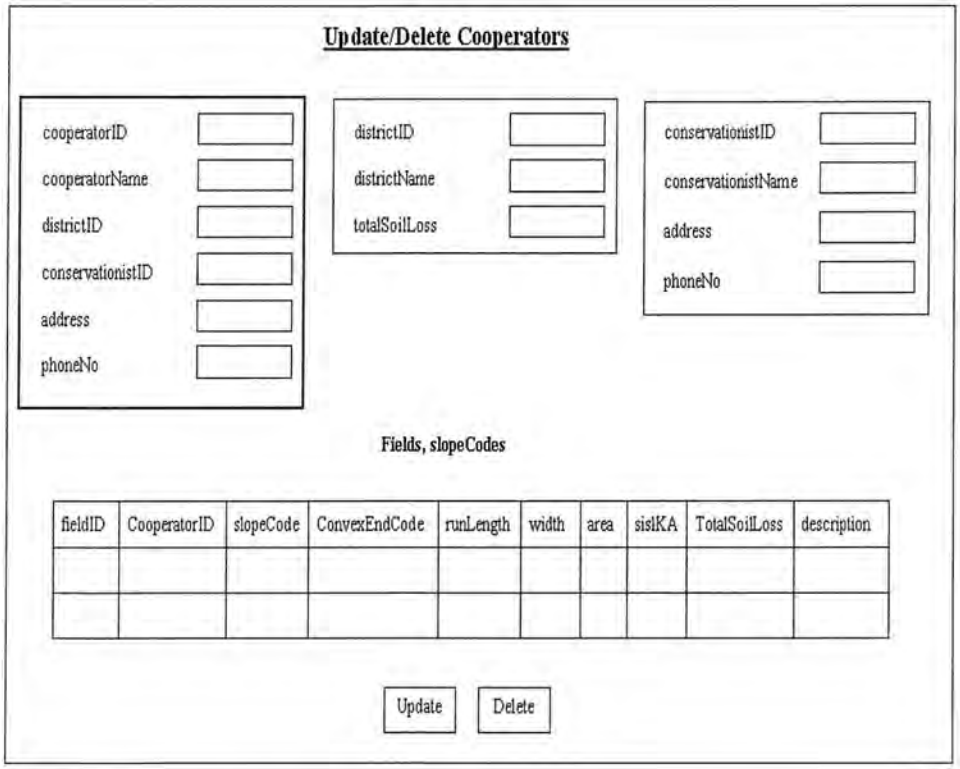

Figure 5: Form `updateCooperators`.

the record in table A. This is called *appending*, and the record in table B is called an *appended record.*

Rule 2: When the relationship type between tables A and B is one-to-many, multiple records in table B can be reached from one record in table A. Such records in table B are called *expanded records.*

Rule 3: An appending operation is possible for any record, i.e., the anchor record, an appended record, or an expanded record. Furthermore, any number of appending operations are allowed.

Rule 4: An expansion is possible only from the anchor record or from a record directly or indirectly appended to the anchor record. We can display within one form only those records that can be reached via at most one expansion.

Rule 5: The attribute values of the anchor record and those of the records appended to it directly or indirectly can be displayed in the main area of the form.

Rule 6: The attribute values of an expanded record and those records directly or indirectly appended to the expanded record can be displayed in one row of a nested table within the form.

We now explain how the above rules can be used to determine the set of tables covered by form sets.

The anchor record is a `Cooperators` record. Since table `Cooperators` is associated with tables `Conservationists` by a many-to-one relationship type, according to rule 1, a `Conservationists` record can be appended. Similarly, a `Districts` record can be appended to the anchor record. The `Fields` records associated with the anchor `Cooperators` record can be included as expanded records according to rule 2. Table `SlopeCodes` is associated with the table `Fields` record by many-to-one relationship type, `SlopeCodes` record can be appended. The one-to-many relationship type between `Fields` and `Rotaitons`, according rule 4, cannot be included because this will require doubly-nested tables.

2.4 System Catalog

The definition of the structure of a database is called a *schema.* The schema information is stored in the *system catalog*, which is also known as the *data dictionary.* Microsoft

SQL server organizes the system catalog as a set of system tables such as `sysobjects` and `sysforeignkeys`.

The information needed to generate forms and server-side scripts is stored in tables `sysobjects` and `sysforeignkeys` in the system catalog. Table `sysobjects`, whose format is shown in Figure 6, contains one row for each object in the database. An object may be a table, a check constraint, a default constraint, a foreignkey constraint, a primary key constraint, a unique constraint, a log, a scalar function, a inlined table-function, a table function, a system table, a trigger, a view, an extended stored-procedure, or a replication-filter stored-procedure.

Table `sysforeignkeys`, whose format is shown in Figure 7, contains the information on the *foreign-key constraints*. Each foreign key contraint indicates a *many-to-one* relationship type between tables.

Column Name	Data Type	Description
name	sysname	Name of the object
ID	int	Identification number of the object
xtype	char(2)	Type of the object
uid	smallint	User ID of the object

Figure 6: Format of table `sysobjects`.

Column Name	Data Type	Description
constid	int	ID of the foreign key constraint
fkeyid	int	Object ID of the table with the foreign key constraint
rkeyid	int	Object ID of the table referenced in the foreign key constraint
fkey	smallint	ID of the referencing column
rkey	smallint	ID of the referenced column
keyno	smallint	Position of the column in the reference column list

Figure 7: Format of table `sysforeignkeys`.

Consider now the partial contents of tables `sysobjects` and `sysforeignkeys` given in Figure 8 and Figure 9, respectively.

The unique identifier of each table is stored in column `id` of table `sysobjects`. For example, the unique identifier of table `Cooperators` is defined to be 645577338. Similarly,

unique identifier 613577224 and 693577509 are assigned to table `Conservationists` and `Districts`, respectively. If table A is related to table B by a many-to-one relationship type, there is a row in table `sysforeighkeys` such that its `fkeyid` value is the `id` of table A and its `rkeyid` value the `id` of table B. For example, the `fkeyid` value 645577338 is associated with the `rkeyid` values 613577224 and 693577509 in table `sysforeignkeys`, and hence there are many-to-one relationship types from table `Cooperators` to tables `Conservationists` and `Districts`.

The row with the `fkeyid` value 725577623 and the `rkeyid` value 645577338 in table `sysforeignkeys` implies a one-to-many relationship type from table `Cooperators` to table `Fields`, whose unique identifier is 725577623.

The tables that can be appended to, e.g., table `Cooperators`, can be obtained with the following SQL statement:

```
select s2.name
from sysobjects s1, sysobjects s2, sysforeignkeys
where ( s1.name = 'Cooperators') and
      ( sysforeignkeys.fkeyid = s1.id ) and
      ( s2.id = sysforeignkeys.rkeyid )
```

The names of the tables whose records can be expanded from a cooperator record can be obtained with the following SQL statement:

```
select s2.name
from sysobjects s1, sysobjects s2, sysforeignkeys
where ( s1.name = 'Cooperators') and
      ( sysforeignkeys.rkeyid = s1.id ) and
      ( s2.id = sysforeignkeys.fkeyid )
```

3 Implementation of WebSiteGen2

We now describe the major capabilities provided by WebSiteGen2, which is implemented in Java. We first present the user interface of WebSiteGen2 and describe its classes. We then explain how WebSiteGen2 uses a tree menu to let a user select the tables to be appended and expanded to and how server-side scripts in ASP are generated. We also explain the form sets generated for the SISL application and how JDBC is used by WebSiteGen2.

name	id
BasicSoilLoss	597577167
Conservationists	613577224
ConservePractices	629577281
Cooperators	**645577338**
CropTypes	661577395
Crops	677577452
Districts	693577509
FieldToSoilMapUnits	709577566
Fields	725577623
Rotations	741577680
SlopeCodes	757577737
SoilErodibility	773577794
SoilMapUnits	789577851

Figure 8: Partial content of table `sysobjects`.

fkeyid	rkeyid
645577338	613577224
645577338	693577509
677577452	629577281
677577452	661577395
677577452	741577680
709577566	725577623
709577566	789577851
725577623	**645577338**
725577623	757577737
741577680	725577623
789577851	773577794

Figure 9: Partial content of table `sysforeignkeys`.

3.1 Graphical user Interface

Figure 10 shows the user interface of WebSiteGen2. The main frame contains a menu bar, a tool bar, and a status bar.

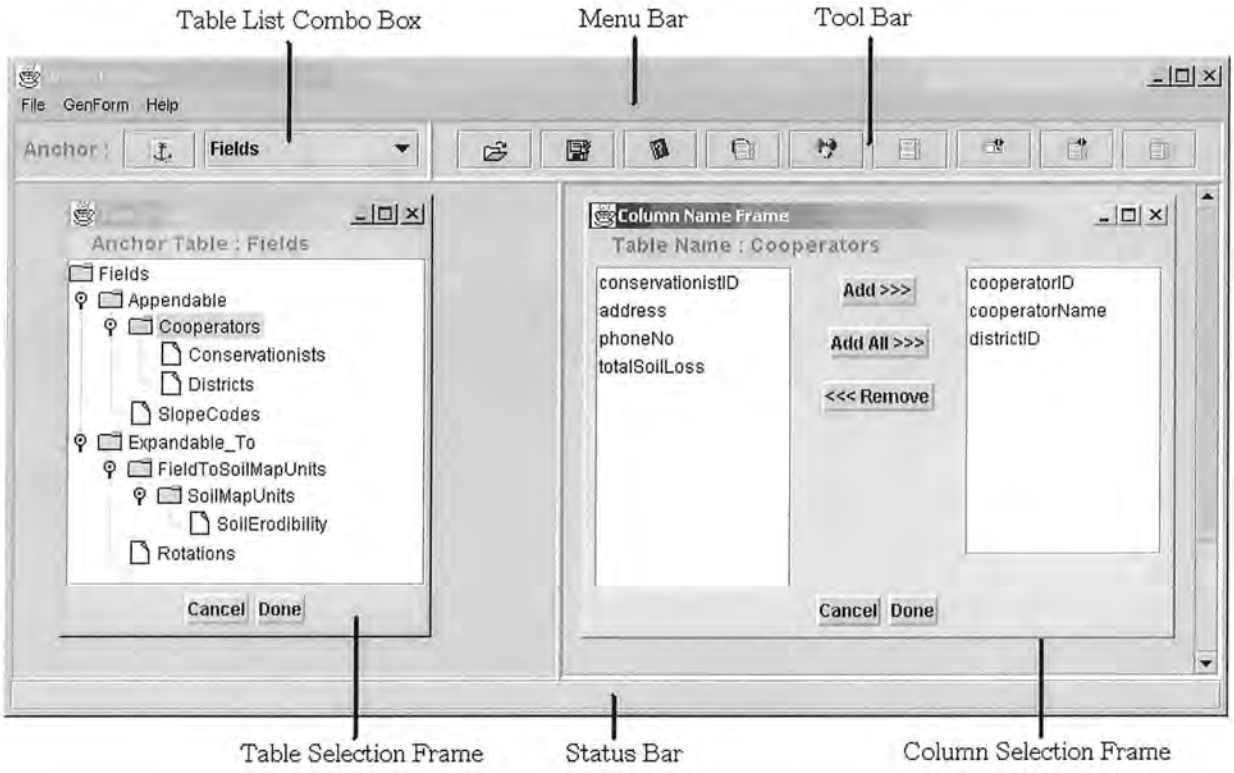

Figure 10: Graphical interface of WebSiteGen2.

When a user conection to a database is established, the anchor table can be selected with the list of the table names displayed in the *table-list combo-box*. The names of the tables are retrieved from the table `sysobjects`. This combo-box is located at the top-left coner of the graphical interface of WebSiteGen2.

When the anchor table is selected by a user, the *table selection frame* will show up. This frame contains a tree menu showing the names of the anchor table and the names of the tables that can be added to the current form by appending and expansion. The root node in the tree menu is the name of the anchor table. The names of the table that can be appended to the anchor table are listed under the node *Appendable*. The names of the table that can be expanded to the anchor table are listed under the node *Expandable-to*.

When a table to be appended or expanded to is selected, the *column selection frame* is displayed. This frame contains the column names of the selected table from which the

user can choose the column names that he wants to be included in the form.

Once all the tables and their columns to be covered by a form set are selected, press the button `GenFrom`, then four forms are generated by WebSiteGen2 automatically.

3.2 WebSiteGen2 Classes

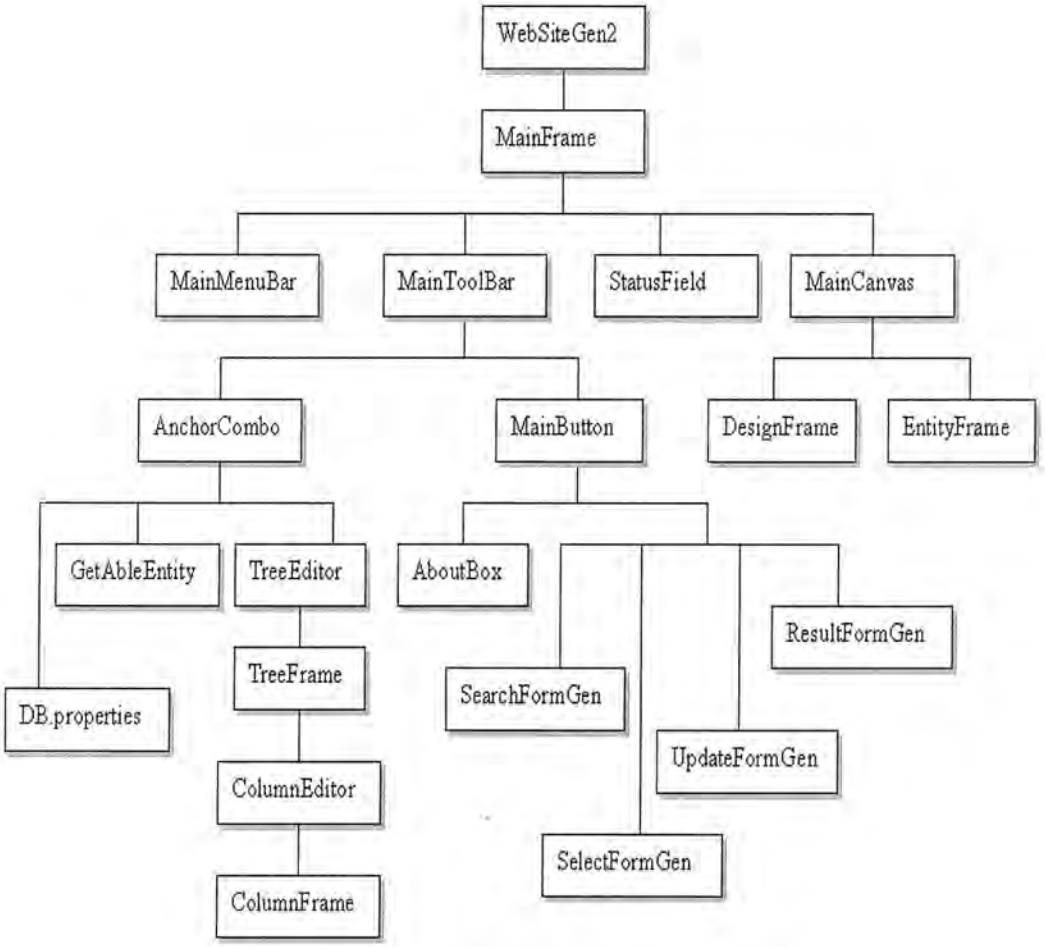

Figure 11: WebSiteGen2 component hierarchy.

The component hierarchy of WebSiteGen2 is shown in Figure 11. Instance of the classes shown under each class are created by an instance of that class. For example, an instance of class `MainFrame` creates instances of classes `MainMenuBar`, `MainToolBar`, `MainCanvas`, and `StatusField`. We explain the classes shown in Figure 11.

`WebSiteGen2`: This class has the main method. It creates the main frame of WebSiteGen2.

`MainFrame`: This class is a subclass of `JFrame` and contains four components, namely, a `MainMenuBar`, a `MainToolBar`, a `MainCanvas`, and a `StatusField`. These components are added to the `JPanel` obtained from the `MainFrame` by the `getContentPane()` method of class `Container`. Method `windowEvent()` for closing the `MainFrame` is also defined.

`MainMenuBar`: This class is a subclass of `JFrame`, and it contains menu items `File`, `GenForm`, and `Help`. The `File` menu allows commands `Open`, `Save`, and `Exit`. The `GenForm` menu allows form sets to be generated. Menu events are processed by method `windowEvent()`.

`MainToolBar`: This class is a subclass of `JFrame`, and it contains two components, namely, an `AnchorCombo` and a `Mainbuttons`. These components are added to the `JPanel` of the `MainToolBar`.

`AnchorCombo`: An instance of this class is created when the program is started. After a connection to a database is established, the names of the tables are retrieved from table `sysobjects`. The anchor table can be selected by the user from the list of the table names displayed in the `JComboBox` object.

`MainButtons`: This class is a subclass of `JPanel`, and it contains the buttons for all the commands supported with the menu bar. Button events are proccessed by method `windowEvent()`.

`MainCanvas`: This class is a subclass of `JFrame` and it contains two components, namely, an `EntityFrame` and a `DesignFrame`. These components are added to the `JPanel`.

`GetAbleEntity`: This class creates a tree structure by using class `JTree` in Swing library. It selects the tables that can be appended and expaneded to.

`TreeFrame`: This class is a subclass of `JFrame`. The `TreeFrame` contains a tree menu showing the names of the anchor table and the names of the tables that can be added to the current form by appending or expansion. A tree menu is added to the `JPanel` obtained from the `TreeFrame` by the `getContentPane()` method of class `Container`. Method `TreeSelectionEvent()` handles a request for table selection.

`ColumnFrame`: This class is a subclass of `JFrame`. The `ColumnFrame` contains the names of the column of the selected table. The column list is added to the `Panel` obtained from the `ColumnFrame` by the `getContentPane` method of class `Container`. The user can choose the columns that he wants to be included in the form. Method `ListSelectionEvent()` handles a request for column selection.

`SearchFormGen`: An instance of this class is created when the user clicks the `GenForm` menu. This class is used to generate a `insert-search` form from the tree structure constructed by the `TreeFrame`. The output file is written with a `PrintWriter` object.

`SelectFormGen`: This class is used to generate a `select` form.

`UpdateFormGen`: This class is used to generate an `update-delete` form.

`ResultFormGen`: This class is used to generate a `result` form

3.3 Selecting Tables to be Appended and Expanded

A tree menu is effective in representing a hierarchical structure, such as an organization of directories and files and an inheritance relationships among classes. By using the java class `JTree`, we implemented a tree menu as shown in Figure 12 for selecting the tables to be appended and expanded. Nodes in the tree menu can be expanded or collapsed.

In our example shown in Figure 12, the anchor table is `Fields`. Table `Fields` is associated with tables `Cooperators` and `SlopeCodes` by many-to-one relationship types, and hence these tables are included in the tree menu as `Appendable` nodes. Furthermore, table `Cooperators` is associated with tables `Conservationists` and `Districts` by many-to-one relationship types, and hence they can be appended to table `Cooperators`.

On the other hand, table `Fields` is associated with tables `Rotations` and `FieldToSoilMapUnits` by one-to-many relationship types, and hence they are included in the tree menu as `Expandable-To` nodes. Furthermore, table `FieldToSoilMapUnits` is associated with table `SoilMapUnits` by a many-to-one relationship type, and hence it can be appended to table `FieldToSoilMapUnits`. Similary, table `SoilErodibility` can be appended to table `SoilMapUnits`.

Since the relationship types between tables can be retrieved from table `sysforeignkeys` in the system catalog, the tree menu is automatically generated by WebSiteGen2 for any anchor table selected.

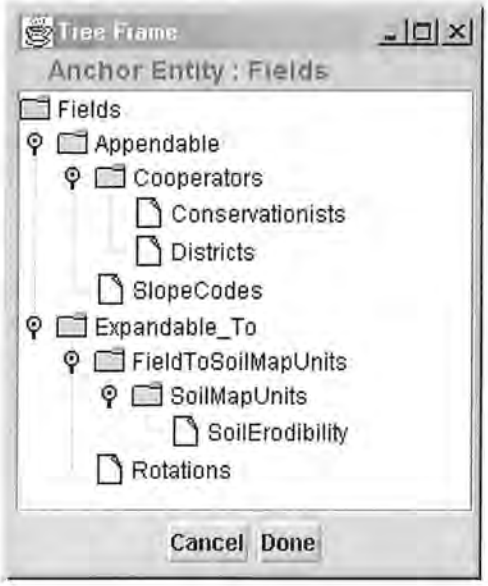

Figure 12: Tree menu for table selection.

3.4 Grouping of Tables

We now explain how our rules are used to determine the set of tables covered by each form set. The SISL database application is used for four form sets, namely, `Cooperators` form set, `Rotations` form set, `SoilMapUnits` form set, and `Fields` form set, as shown in Figure 13.

In the `Cooperators` form set, the anchor table is `Cooperators`. Table `Cooperators` is associated with table `Fields` by a one-to-many relationship type. Therefore, table `Fields` is made an expanded table. Table `Cooperators` is associated with tables `Conservationists` and `Districts` by many-to-one relationship types. They are appended to table `Cooperators`. The relationship type between tables `Fields` and `Rotations` is one-to-many. We cannot include table `Rotations` in `Cooperators` form set since doing so will require a doubly-nested table.

In `Rotations` form set, the anchor table is `Rotations`. Table `Rotations` is associated with table `Fields` by a many-to-one relationship type, and hence table `Fields` is appended to table `Rotations`. Table `Rotations` is associated with table `Crops` by a one-to-many relationship type. Therefore, table `Crops` is expanded from table `Rotations`. Furthermore, table `Crops` is associated with tables `CropTypes` and `ConservePractices` by many-to-one relationship types, and hence tables `CropTypes` and `ConservePractices` are appended to table `Crops`. The relationship type between tables `Fields` and `Cooperators` is many-to-one, and hence table `Cooperators` is appended to

table Fields. Similary, tables Conservationists and Districts are appended to table Cooperators.

In SoilMapUnits form set, the anchor table is SoilMapUnits. Table SoilMapUnits is associated with table SoilErodibility by a many-to-one relationship type, and hence table SoilErodibility is appended to table SoilMapUnits. Table SoilMapUnits is associated with table FieldToSoilMapUnis by a one-to-many relationship type. Therefore, table FieldToSoilMapUnits is expanded from table SoilMapUnits. Furthermore, table FieldToSoilMapUnits is associated with table Fields by a many-to-one relationship type, and hence, table Fields is appended to table FieldToSoilMapUnits. Similary, tables Cooperators and Slopecode are appended to table Fields. Also, tables Conservationists and Districts are appended to table Cooperators.

In Fields form set, the anchor table is Fields. Table Fields is associated with tables Cooperators and SlopeCodes by many-to-one relationship types, and hence tables Cooperators and SlopeCodes are appended to table Fields. Table Cooperators is associated with tables Conservationists and Districts by many-to-one relationship types, and hence tables Conservationist and Districts are appended to table Cooperators. Table Fields is associated with tables Rotations and FieldToSoilMapUnits by one-to-many relationshipt types. Therefore, tables Rotations and FieldToSoilMapUnits are expanded from table Fields. Furthermore, table FieldToSoilMapUnits is associated with table SoilMapUnits by a many-to-one relationshipt type, and hence table SoilMapUnits is appended to table FieldToSoilMapUnits. Similary, table SoilErodibility is appended to table SoilMapUnits.

3.5 Active Server Pages (ASP)

ASP (Active Server Pages) is a server-side scripting language designed by Microsoft. An ASP file can contain HTML statements and scripts that are processed by the Internet Information Services (IIS) server. When a browser requests an HTML file, the server returns the file. When a browser requests an ASP file, the server passes the request to the ASP extension. The ASP extension then reads the ASP file and executes the scripts in the file, generating a plain HTML file to be returned to the browser.

An ASP page can formulate a database query by using the parameters passed from the brower with an HTML form. It then can create an HTML page displaying the result of the query. Since a plain HTML file is returned to the web browser, the ASP scripts

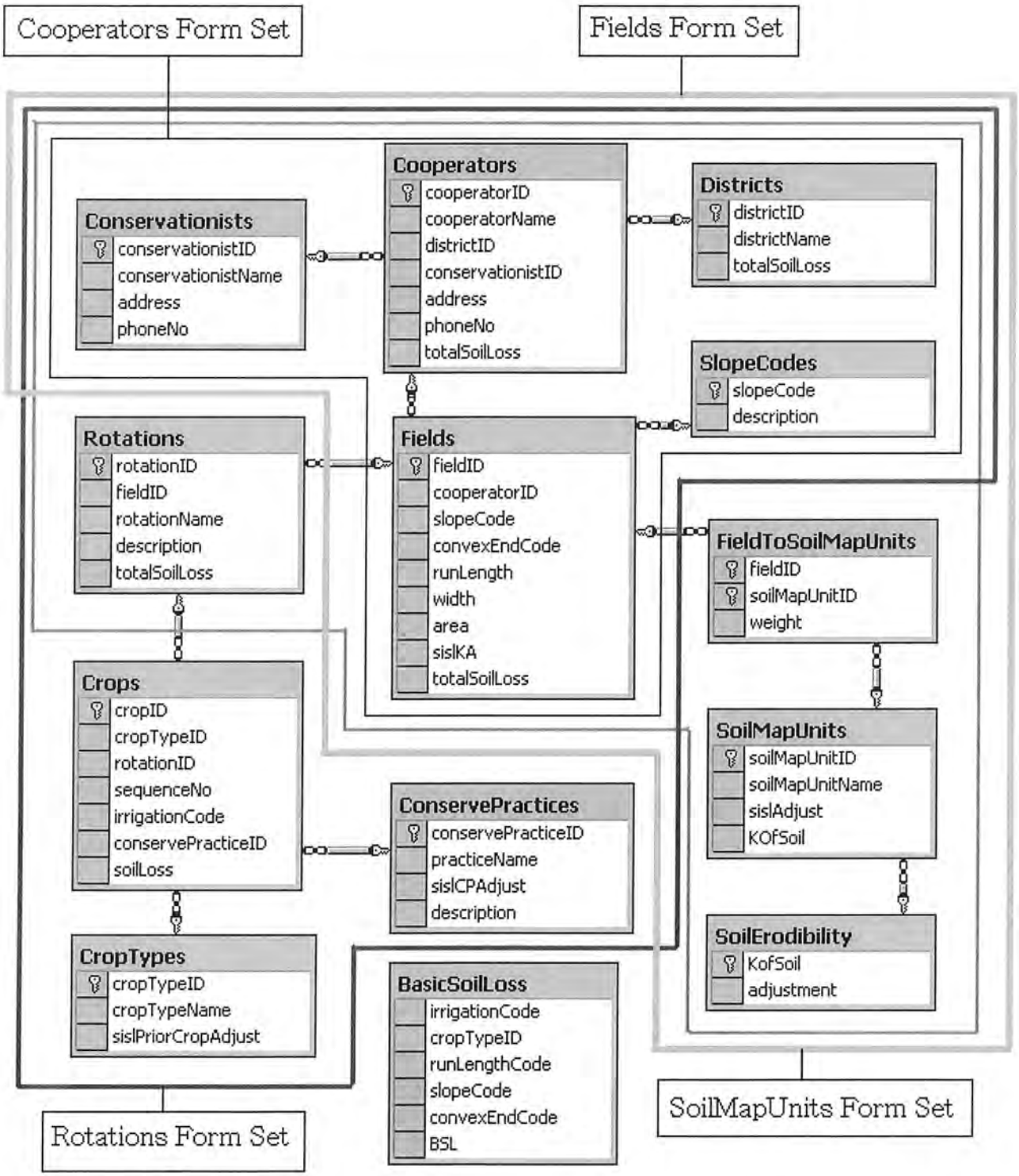

Figure 13: Coverage by form sets.

```
<%@ Language = VBScript %>

<HTML>
  <BODY>
  <%
    Dim connect
    Dim recordSet
    Dim VARSQLStatement
    Dim VARcooperatorName
    Dim VARcooperatorID
    Dim VARdistrictID
    Dim VARconservationistID
    Dim VARaddress
    Dim VARphoneNO
    Dim VARtotalSoilLoss

    VARcooperatorID = Request.Form("cooperatorID")
    VARcooperatorName = Request.Form("cooperatorName")
    VARaddress = Request.Form("address")
    VARphoneNO = Request.Form("phoneNo")

    Set connect = Server.CreateObject("ADODB.Connection")
    connect.Open "Provider = SQLOLEDB;User ID = webAccess; PWD=ari3rang;" & _
                "Initial Catalog = SISL4;Data Source = 128.193.248.245,1072"

    if (CStr(Request.Form("Delete")) = "Delete" ) then
      strSQLStatement = "delete from Cooperators where cooperatorID = '"_
                        & VARcooperatorID & "'"
      set recordSet = connect.Execute(strSQLStatement)
      Response.Write "cooperators has been deleted"
    else
      strSQLStatement = "update cooperators set "_
                       & " cooperatorName = '" & VARcooperatorName & "',"_
                       & " PhoneNo = '" & VARphoneNo & "'"_
                       & " WHERE cooperatorID = '" & VARcooperatorID & " ' "
      set recordSet = connect.Execute(strSQLStatement)
      Response.Write "cooperators has been Updated"
    End if

  %>
</BODY>
</HTML>
```

Figure 14: ASP file `resultCooperators.asp` (part).

within an ASP file are not visible to the browser.

Figure 14 shows an example of an ASP file generated automatically by WebSiteGen2. This ASP page creates a connection to the SISL database and executes a `delete` or `update` SQL statement based on the parameters submitted with an HTML form.

Consider now the part of the ASP file `resultCooperators.asp` given in Figure 14. The first line of the file `<%@ Language = VBScript %>` indicates that the scripting lanage is VBScript. In addition to the standard HTML statements, an ASP file can contain server-side scripts, which are surrounded by delimiters `<%` and `%>`.

Within the script, variable names are declared with `Dim` statements:

```
Dim connect
Dim recordSet
Dim VARcooperatorName
```

By using a `Request` object, an ASP script can obtain such informations as the values of the parameters passed from a HTML form, querystring data, URL, cookies, client cerificate data, authentication data, and content length. In our example, the value of paramenter `cooperatorName` passed from form `updateCooperators.asp` is retrieved with the following ASP statement:

```
VARcooperatorName = Request.Form("cooperatorName")
```

Then a connection to a database is opened with a `Connection` object. The `Connection` object created is an ADO (ActiveX Data Objects) object, and it uses the OLEDB provider, which is a software module like a driver. The other arguments provided for the method `Open()` are the database login name, the password for the database access, the name of the database, and the IP address of the database server.

```
Set connect = Server.CreateObject("ADODB.Connection")
connect.Open "Provider = SQLOLEDB;User ID = webAccess; PWD=ari3rang;" & _
             "Initial Catalog = SISL4;Data Source = 128.193.248.245,1072"
```

Once the connection to the databse is established, an SQL statement is used to retrieve data from the database. The data retrieved are stored in a `RecordSet` object `recordSet`.

```
set recordSet = connect.Execute(strSQLStatement)
```

3.6 Retrieving Metadata

WebSiteGen2 uses an ODBC-JDBC bridge driver to access the system catalog of a database. The parameters specified for the driver are shown Figure 15. The statement `jdbc.drivers = sun.jdbc.odbc.JdbcOdbcDriver` indicates the driver to be used. `SISL` in the statment `jdbc.url = jdbc:odbc:SISL` indicates the ODBC data source, `jdbc.username = seikyung` indicates for the database login name, and `jdbc.password = ari3rang` indicates the password for the database access.

```
jdbc.drivers = sun.jdbc.odbc.JdbcOdbcDriver
jdbc.url = jdbc:odbc:SISL
jdbc.username = seikyung
jdbc.password = ari3rang
```

Figure 15: Specifying database connection parameters.

Figure 16 shows how the WebSiteGen2 program retrieves the table names and the column names from the system catalog. The method `getConnection()` returns `Connection` object `con`, which is used to create a `DatabaseMetadata` object `md`. Method call `md.getTables()` returns result set `mrs`, whose row contains the information on the tables in the database. The table name in each entry in `mrs` is retrieved by method call `mrs.getString(3)`, where `"3"` indicates the position of the table name field.

In order to retrieve the column names of the table whose name is stored in string `tableName`, we execute with a `Statement` object the SQL command `"select * from" + tableName`. Once the result set `rs` is properly set, we can retrieve the number of the columns and their names with methods call `getColumnCount()` and `getColumnLabel()`.

4 Conclusions and Future Work

We implemented a software tool that generates server-side scripts for a web-based database application from the system catalog of a database. This tool is called WebSiteGen2 and generates ASP pages for Microsoft SQL Server 2000. The graphical user interface of WebSiteGen2 allows the user to select the tables and the columns to be included in each form.

```
// Read tables names from database
Connection con  = getConnection();
DatabaseMetadata md = con.getMetadata();
ResultSet mrs = md.getTables(null, null, null, new String[] { "Table" });
while (mrs.next(0) {
  tableList.addItem(mrs.getString(3));
}
mrs.close();
// Read column names from database
Statement stmt = con.createStatement();
String query = "select * from" + tableName;
ResultSet rs = stmt.executeQuery(query);
DatabaseMetadata rsmd = rs.getMetadata();
for (int i = 0 ; i <= rsmd.getColumnCount() ; i++ ){
  columnNames.addElement(rsmd.getColumnLabel(i);
}
```

Figure 16: Connecting to a database and retrieving its metadata.

The main record included in a form is called an *anchor* record. Besides the anchor record, additional records can be included within the same form. Such records are called *appended* or *expanded* records, and we presented a set of rules for appending records and expanding a record.

WebSiteGen2 generates four forms for each group of tables, namely, an *insert-search* form, a *select* form, an *update* form, and a *result* form. By using these four forms, we can perform `insert`, `search`, `select`, `update`, and `delete` operations on tables.

The information needed to generate forms and server-side scripts is stored in table `sysobjects` and `sysforeignkeys` in the system catalog. Table `sysobjects` contains informaion on the tables in the database and table `sysforeignkeys` contains the information on the relationship type as *foreign key constraints* between the tables.

We can extend WebSiteGen2 to allow its user to adjust the layout of a form with drag-and-drop operations. It also should be extended to generate forms for other scripting languages such as Javascript and Java Server Pages (JSP).

References

[1] MicroSoft, *Microsoft Books Online*, Microsoft.

[2] Cay S.Horstmann, Gray Cornell, *CoreJava2 Volumn I-Fundamentals*, Sun Microsystems, 2000.

[3] Cay S.Horstmann, Gray Cornell, *CoreJava2 Volumn II-Advanced Features*, Sun Microsystems, 2000.

[4] David Buser, John Kauffman, Juan T.Libre, Brian Francis, David Sussman, Chris Ullman, Jon Duckett, *Active Server Page 3.0*, Wrox, 2000.

[5] Virginia Anderson, *Access 2000*, McGraw-Hill, 2000.

[6] C.J.Date, *An Introduction to Database Systems, 6 Edition*, Addison-Wesley, 1995.

[7] Lin Li, *Automatic Generation of Forms and Java Servlets for a Web-Based Database Application*, Master Project Report, Dept. of Computer Science, Oregon State University, 2000.

[8] Floyd G.Bailey, *Technical Note*, USDA-Soil Conservation Service.

[9] Albert Lulushi, *Oracle Deceloper/2000 Forms*, Prentice Hall, 1999.