WebGen 5: A PHP Web Script Generator with Templates

By Song Yang

Submitted to Oregon State University

In partial fulfillment of the requirement for the degree of

MASTER OF SCIENCE

in Computer Science

School of Electrical Engineering and Computer Science

Oregon State University

Corvallis, OR

September 2004

Acknowledgements

I would like to express my gratitude to Dr. Toshimi Minoura for his support and guidance through my study. I am also very grateful to Dr. Jimmy Yang and Dr. Michael Quinn for serving on my committee and for their valuable comments and suggestions to my work. On a personal note, I would like to acknowledge the support and love from my parents and friends.

Abstract

WebGen 5 is a software tool for automatically generating *Web scripts* that display *Web forms* and operate on data in a database. WebGen 5 is implemented as a collection of *templates*. Each template, combined with a corresponding *configuration file*, generates one of the following six types of Web scripts: *search, select, edit, information, action,* and *treeview* scripts. Templates and configuration files are written in PHP. The Web scripts generated by them are also in PHP. We are using WebGen to generate web scripts for actual applications. One of the applications contains more than 700 tables in the database and hence more than 3500 scripts are generated.

Table of Contents:

1. Introduction

WebGen 5 is a software tool for automatically generating *Web scripts* that display *Web forms* and operate on data in a database. WebGen 5 is implemented as a collection of *templates*. Each template, combined with a corresponding *configuration file*, generates one of the following six types of Web scripts: *search, select, edit, information, action,* and *treeview* scripts. Templates and configuration files are written in PHP. The Web scripts generated by them are also in PHP. Now, we are using WebGen 5 in real Web site projects. WebGen 5 significantly reduces the time used for these projects. In addition, the generated code is easy to be maintained.

We had worked on four Web script generators before we began developing WebGen 5. WebSiteGen 1 was our first attempt. It generated Web scripts from an *ER diagram*. However, this approach was not effective, because an ER diagram may not accurately reflect the real structure of a database. Starting with WebSiteGen 2, we used relational database schemas to generate Web scripts. WebSiteGen 2 was a Windows application written in Java that generated ASP Web scripts. About one year later, we developed WebSiteGen 3 which generated more complex ASP.NET Web scripts supporting *one-to-many* and *many-to-one* relationships between tables. WebSiteGen 3 was written in C#, and it was actually used to generate Web scripts for real projects.

When WebSiteGen 3 was partially completed, we needed to generate PHP Web scripts, and started to develop WebSiteGen 4. However, WebSiteGen 4 was not very successful. The code became long and hard to understand. A change in one part often caused ripple effects throughout the entire Web script generator, and hence the generator was difficult to maintain.

While trying to overcome the problems in the previous versions, we came across a new idea of using templates for generating Web scripts. Because a template resembles the generated Web scripts, creating a set of templates is easier than writing a generator in a conventional programming language. Moreover, as one template

only generates one type of web scripts, changes in one template do not affect other templates, unless the changes are related to parameters passed between scripts.

Section 2 presents an overview of the organization of web scripts and forms. In Section 3, we explain to how to make a WebGen configuration file from a database schema to generate desired Web scripts. Section 4 explains a TreeGen configuration files and a treeview Web form. A summary and some suggestions for future developers are given in Section 5.

2. Organization of Web Scripts and Forms

WebGen 5 can generate six types of Web scripts: *search*, *select*, *edit*, *information*, *action*, and *treeview* scripts. The generated scripts are executed on a Web sever by a PHP interpreter. Each script, except for an action script, creates a Web form that is displayed on a client computer by a Web browser. Figure 2.1 illustrates the interactions among the Web scripts and forms.

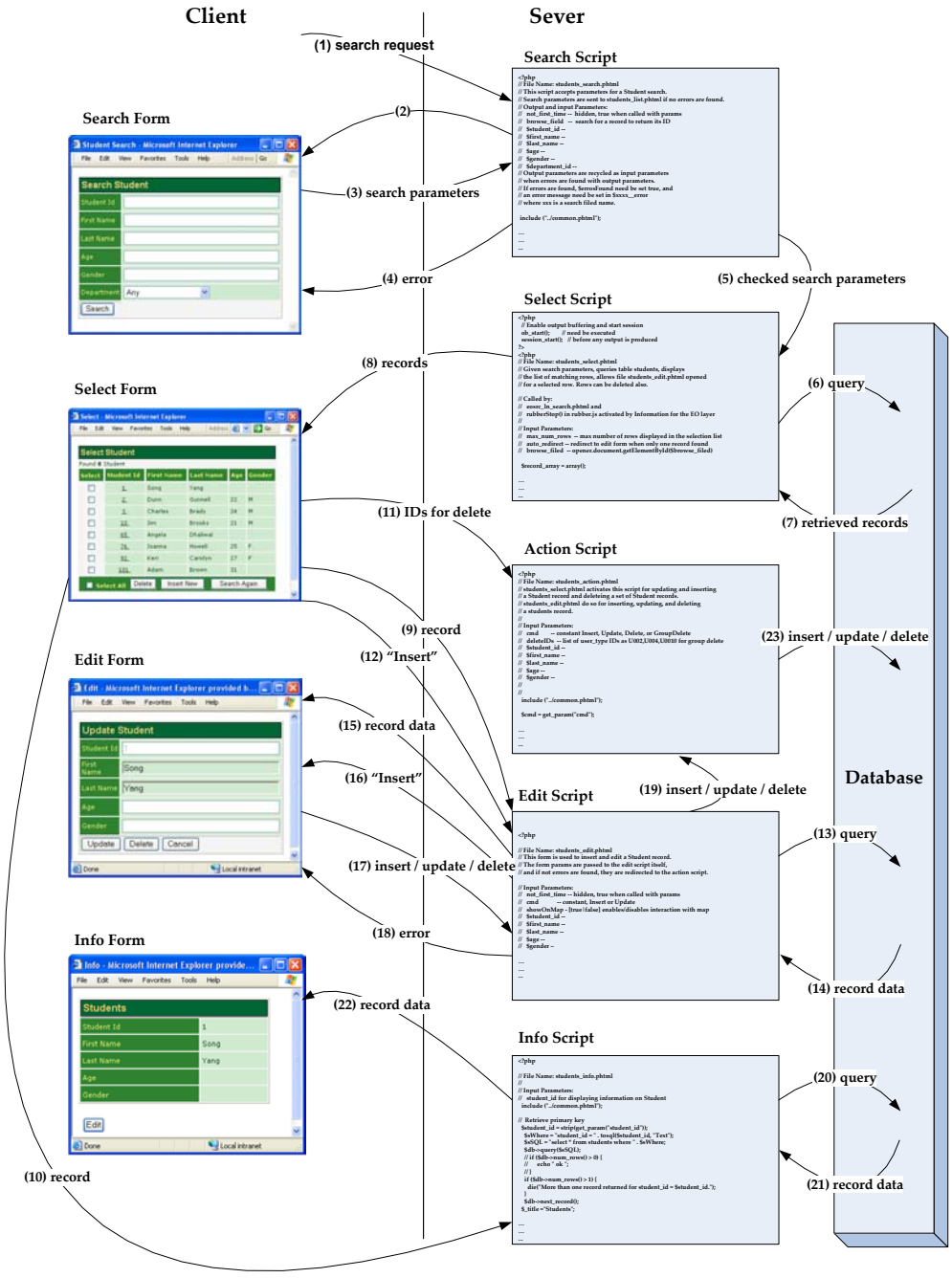

Figure 2.1: Interactions among Web Scripts and Forms.

The search script and form allow a user to provide search parameters.

- When a search request is received by the search script (1), the script creates a search form and sends the form to the browser (2).

- When the `Search` button on the search form is clicked, the form sends the search parameters provided by the user to the search script (3).

- When the search parameters are received (3), the search script checks the received parameters. If there are errors, the search script sends back the search parameters with error flags to the browser (4). Otherwise, the search script sends the search parameters to the select script (5).

The select script and form retrieve the records satisfying the given search parameters and display them.

- When the search parameters are received from the search script (5), the select script generates a SELECT statement based on these parameters to retrieve records (6).

- When the records are retrieved (7), the select script creates a select form for the records retrieved and sends the form to the browser (8).

- If the system mode is `Update`, when the ID of a record in the select form is clicked, the form sends the primary key of the selected record, along with parameter `cmd=Update`, to the edit script (9). If the system mode is `Info`, the select form sends the value of the primary key of the selected record to the information script (10).

- When the `Delete` button on the select form is clicked, the form sends the values of the primary keys of all the selected records, along with parameter `cmd=GroupDelete`, to the action script (11).

- When the `Insert` button on the select form is clicked, the form sends parameter `cmd=Insert` to the edit script (12).

The edit script and form display the detailed information about one record and allow the user to perform data operations.

- When parameter `cmd=Update` and the value of the primary key of the selected record are received (9), the edit script generates a SELECT statement based on the value of the primary key to retrieve the complete record (13).

- When the complete record is retrieved (14), the edit script creates the edit form for the complete record and sends the form to the browser (15).

- When parameter `cmd=Insert` is received from a select form (12), the edit script creates the edit form for insertion and sends the form the browser (16).

- When the `Update` button on the edit form is clicked, the form sends the modified data, along with parameter `cmd=Update`, to the edit script (17).

- When parameter `cmd=Update` and the modified data are received from the edit from (17), the edit script checks the data. If there are errors, the edit script sends back the data with error flags to the browser (18). Otherwise, the edit script sends the parameter and the data to the action script (19).

- When the `Delete` button on the edit form is clicked, the form sends the value of the primary key of the record, along with parameter `cmd=Delete`, to the edit script (17).

- When parameter `cmd=Delete` and the value of the primary key are received from the edit form (17), the edit script sends the parameters and the value of the primary key to the action script (19).

- When the `Insert` button on the edit form is clicked, the form sends the new data, along with parameter `cmd=Insert`, to the edit script (17).

- When parameter `cmd=Insert` and the new data are received from the edit from (17), the edit script checks the data. If there are errors, the search script sends back the data with error flags to the browser (18). Otherwise, the search script sends the parameters and the data to the action script (19).

The information script and form display the detailed information about one record.

- When the value of the primary key of the selected record are received from the select form (10), the information script generates a SELECT statement based on the value of the primary key to retrieve the complete record (13).

- When the complete record is retrieved (14), the information script creates the information form for the complete record and sends the form to the browser (15).

The action script actually inserts, deletes, and updates data stored in the database.

- When parameter `cmd=Insert` and the new data are received (19), the action script generates an SQL statement for inserting a new record (23).

- When parameter `cmd=Update` and the modified data are received (19), the action script generates an SQL statement for updating the record (23).

- When parameter `cmd=Delete` and the value of the primary key of a record are received (19), the action script generates an SQL statement for deleting the record (23).

- When parameter `cmd=GroupDelete` and the values of the primary keys of the records are received (19), the action script generates SQL statements for deleting those records (23).

- When the data insertion, modification, or deleting is completed, the action script refreshes the Web forms that activates the script.

3. WebGen Configuration File

A *schema* of a database determines the structure of the data stored in the database. A *configuration file* specifies how data fields are grouped in Web forms for searching, inserting, updating, and deleting data stored in the database. By using the information provided in the configuration file, a *template* generates a Web script. In this section, we explain how a configuration file and Web scripts are generated from the schema given in Figure 3.1.

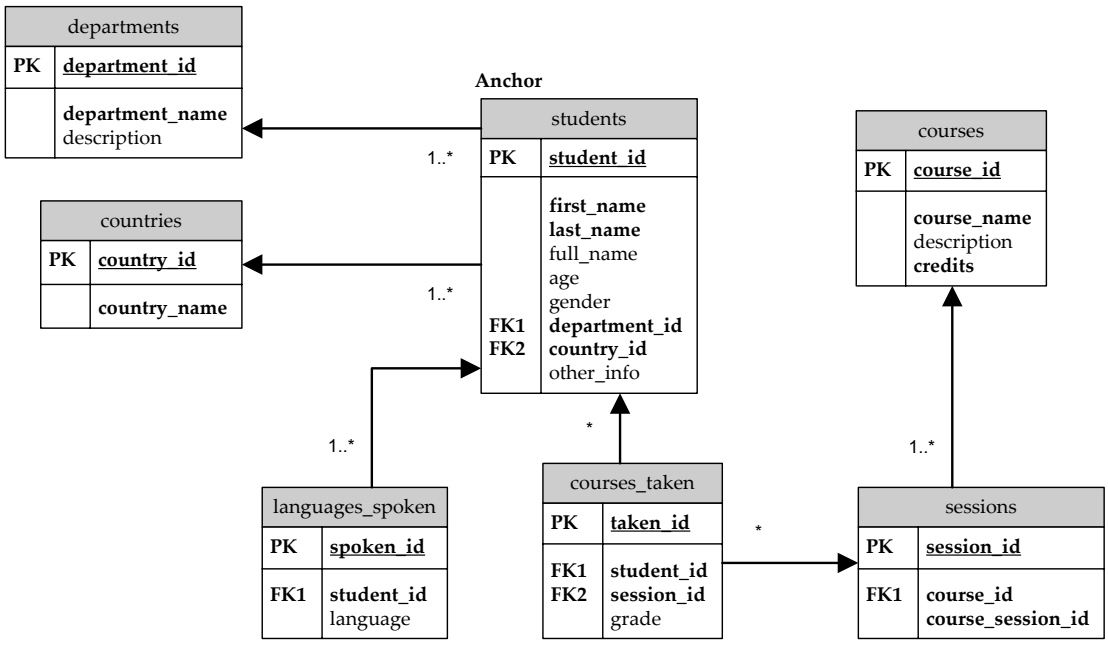

Figure 3.1: A Database Schema.

We use table `students` as an anchor table. The anchor table `students` and table `departments` are associated by a many-to-one relationship type, as are the anchor table and table `countries`. The relationship type between the anchor table and table `languages_spoken` is one-to-many. The relationship type between the anchor table and table `courses_taken` is also one-to-many. Table `courses_taken` and table `sessions` are related by a many-to-one relationship type. Table `sessions` and table `courses` have a many-to-one relationship type.

Table `countries` is used as a *domain table*. A domain table is a table containing a set of possible values that can be displayed in a dropdown list for a user to choose. In our example, table `countries` stores the names of countries in column `country_name`.

In a configuration file, *main variables* specify the information applicable to all the scripts generated from that configuration file. In order to generate Web scripts from the database schema in Figure 3.1, we must define at least the following four main variables:

```
$anchor_table = 'students';
$anchor_label = 'Student';
$anchor_label_plural = 'Students';
$primary_key = 'student_id';
```

`$anchor_table`: The name of the anchor table. In our example, the anchor table is table `students`.

`$anchor_label`: The label representing the anchor table. This label is used as part of the title of a form. In our example, we use `Student` as the label.

`$anchor_label_plural`: The plural form of the anchor label. In our example, we use `Students`.

`$primary_key`: The name of the primary key of the anchor table. In our example, column `student_id` is the primary key of the anchor table `students`.

3.1 Search Fields

Each element in array `$search_fields` designates a field in a search form. By specifying various *options* for the elements in `$search_fields`, we can make a desired search form. In order to generate the search form shown in Figure 3.2, we must define `$search_fields` as follows:

```
$search_fields=array(
  array("column"=>"student_id", "label"=>"Student Id",
    "type"=>"numeric", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"first_name", "label"=>"First Name",
    "type"=>"text", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"last_name", "label"=>"Last Name",
    "type"=>"text", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"age", "label"=>"Age",
    "type"=>"text", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"gender", "label"=>"Gender",
    "type"=>"text", "maxlen"=>"40", "size"=>"40"),
  array("table"=>"departments",
    "column"=>"department_name",
    "label"=>"Department Name", "type"=>"text",
    "maxlen"=>"40", "size"=>"40", "match"=>"exact"),
  array("table"=>"courses", "column"=>"course_name",
    "label"=>"Courses Name", "type"=>"text",
    "maxlen"=>"40", "size"=>"40"),
  array("column"=>"country_id", "label"=>"Country Name",
    "type"=>"domain","domain_table"=>"countries",
    "domain_key"=>"country_id",
    "display_column"=>"country_name")
);
```

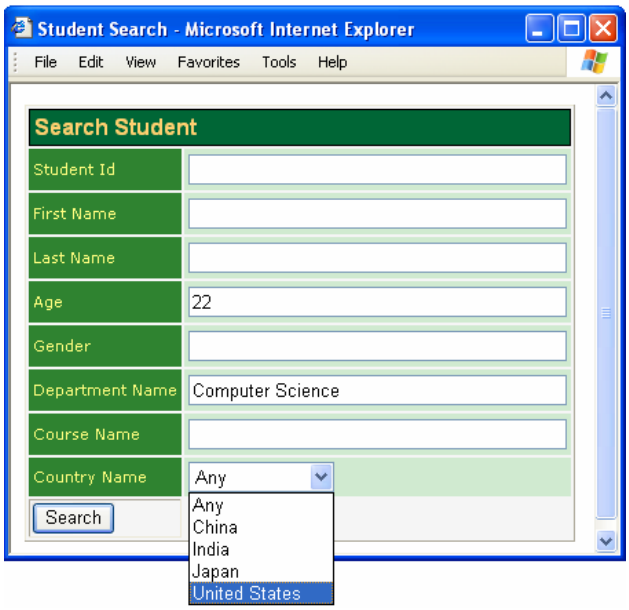

Figure 3.2: Student Search Form.

`column`: The name of a column in a table. In the first element, for example, the column name is `student_id` in the anchor table `students`.

`label`: The label for the field in the search form. In the first element, we use `Student ID` as the label.

`type`: The type of the field. For a field in a search form, the value of option `type` can be one of the followings:

- `numeric`: The search parameter is a number. The input control is a textfield.

- `text`: The search parameter is a string. The input control is a textfield. In searching, if the search parameter is a substring of a column value in a record, the record satisfies the search parameter. However, if the option `match` is set as `match=>exact`, the column data must exactly match the search parameter.

- `date`: The search parameter is a date. The input control is a textfield.

- `timestamp`: The search parameter is a time. The input control is a textfield.

- `email`: The search parameter is an email address. The input control is a textfield.

- `phone`: The search parameter is a phone number. The input control is a textfield.

- `domain`: The search parameter is selected from a dropdown list. This type requires other three options: `domain_table`, `domain_key`, and `display_column`. Option `domain_table` designates the name of the domain table, option `domain_key` the primary key of the domain table, and option `display_column` the name of the column in which the option values are stored. In our example, the field for column `country_id` uses a dropdown list and the values are retrieved from column `country_name` in table `countries`.

`table`: The name of the table to which the column belongs. If the column is in the anchor table, the option can be omitted. In the first element, we omit this option, because column `student_id` is in the anchor table `students`. In the

element for column `department_name`, we set this option, because column `department_name` is in table `departments`.

`maxlen`: The maximum length of a search parameter allowed to be entered in the textfield.

`size`: The size of the textfield, or the maximum length of a search parameter allowed to be entered in the textfield without scrolling.

3.2 Select Fields

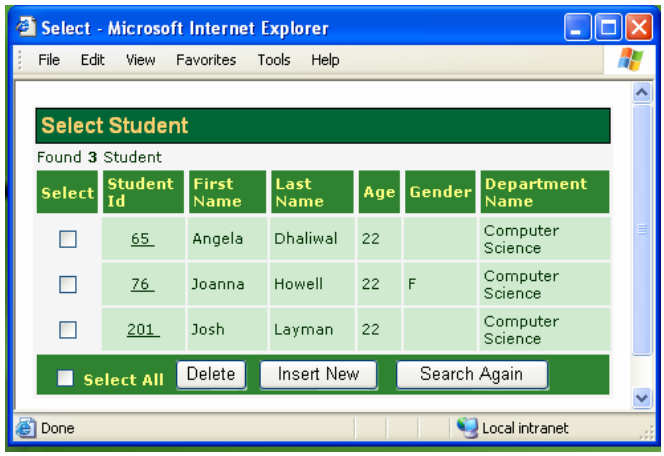

Figure 3.3: Student Select Form.

Each element in array `$select_fields` designates a column used in a select form. In order to generate the select form shown in Figure 3.3, we need to define `$select_fields` as follows:

```
$select_fields = array(
  array("column"=>"student_id", "label"=>"Student Id",
    "type"=>"integer", "size"=>"40"),
  array("column"=>"first_name", "label"=>"First Name",
    "size"=>"40"),
  array("column"=>"last_name", "label"=>"Last Name",
    "size"=>"40"),
  array("column"=>"age", "label"=>"Age",
    "size"=>"40"),
  array("column"=>"gender", "label"=>"Gender",
    "size"=>"40"),
  array("table"=>"departments",
```

```
    "column"=>"department_name",
    "label"=>"Department Name", "size"=>"40")
);
```

An element in `$select_fields` has options `column`, `label`, `table`, `size`, and `type`. These options are similar to those for an element in `$search_fields`. However, the value of option `type` can not be `domain`.

In addition to `$select_fields`, there are two main variables `$default_sort_column` and `$select_join_tables`. These variables are used by a select script in creating a SELECT statement to retrieve the data satisfying search parameters.

`$deafult_sort_column` designates the name of the column used for sorting. The column must be one of the columns used in the select form. The statement below states that column `student_id` is used to sort the retrieved records.

```
$default_sort_column = "student_id";
```

Each element in array `$search_join_tables` indicates how a table associated to the anchor table is linked to the anchor table in the SELECT statement generated by a select script. Each associated table that has columns used in a search form or a select form must be an element of `$search_join_tables`. `$search_join_tables` for retrieving the records displayed in the select form shown in Figure 3.3 need be defined as follows:

```
<%
  $search_join_tables = array(
    "departments"=>array(
      "joined_table"=>"departments ON departments.department_id=
        students.department_id",
      "used"=>"false", "select"=>"true"),

    "courses"=>array(
      "joined_table"=>"courses_taken ON
          students.student_id=courses_taken.student_id
        LEFT JOIN sessions ON
          courses_taken.session_id=sessions.session_id
        LEFT JOIN courses ON
          courses.course_id=sessions.course_id",
    "used"=>"false", "select"=>"false" ),
  );
%>
```

Column `department_name` in table `departments` is used in the search form and the select form, and hence table `departments` need be an element of this array. Column `course_name` in table `courses` is used in the search form, and hence table `course` is also an element of `$search_join_tables`. For each element in the array, there are three options:

- `used`: This option specifies whether the associated table is used for searching. In a configuration file, this option is always set to `false`. However, if a user gives a search parameter for a column in the associated table, the select script sets the value of this option to `true`. In other words, the value of this option is based on the search parameters passed from the search form.

- `select`: This option specifies whether or not columns of the associated table are used in the select form. In our example, the column `department_name` in table `departments` is used in the select form, but no columns in table `courses` are used.

- `joined_table`: This option designates how the associated table is linked to the anchor table in the FROM clause of the SELECT statement generated. In generating the FROM clause, only the associated tables whose option `used` or option `select` is `true` are linked to the anchor table. In the search form shown in Figure 3.2, the user gives the search parameter `Computer Science` for column `department_name` in table `departments`, but he does not give a search parameter for column `course_name` in table `courses`. Therefore, only table `departments` is linked to the anchor table. The generated FROM clause is:

```
FROM students LEFT JOIN departments ON
  students.department_id = departments.department_id
```

If the user also gives a search parameter `Database` to column `course_names` in table `courses`, then both table `departments` and table `courses` are linked to the anchor table.

```
FROM students
  LEFT JOIN departments
    ON students.department_id = departments.department_id
  LEFT JOIN courses_taken
    ON students.student_id = courses_taken.student_id
  LEFT JOIN sessions
    ON courses_taken.session_id = sessions.session_id
  LEFT JOIN courses
    ON sessions.course_id = courses.course_id
```

3.3 Edit Fields

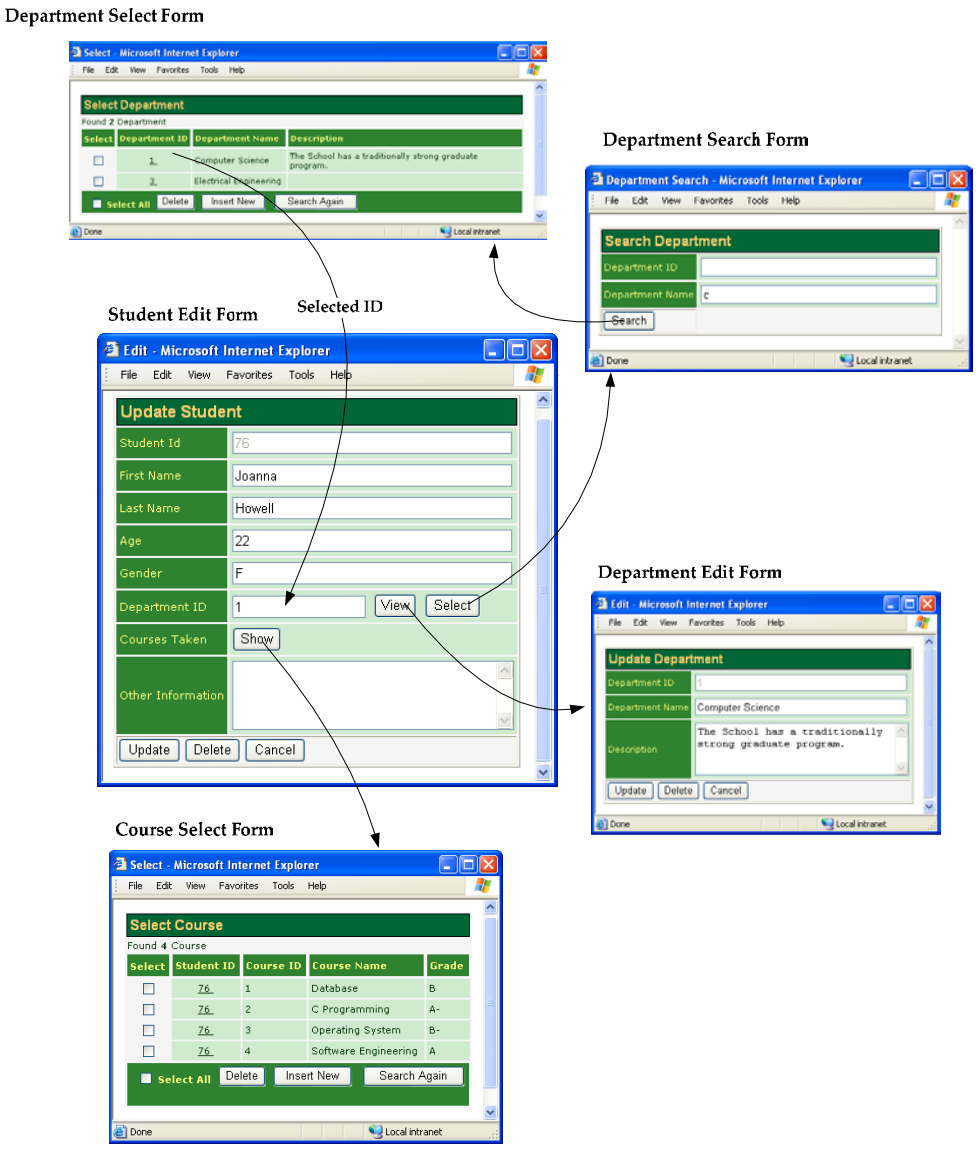

Figure 3.4: Student Edit Form.

Each element in array `$edit_fields` represents a field in an edit form. In order to generate the `Student` edit form shown in Figure 3.4, we must define `$edit_fields` as follows:

```
$edit_fields=array(
  array("column"=>"student_id", "label"=>"Student Id",
    "type"=>"numeric", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"first_name", "label"=>"First Name",
    "type"=>"text", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"last_name", "label"=>"Last Name",
    "type"=>"text", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"age", "label"=>"Age",
    "type"=>"text", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"gender", "label"=>"Gender",
    "type"=>"text", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"department_id", "label"=>"Department ID",
    "type"=>"to_one", "linked_table"=>"departments",
    "maxlen"=>"40", "size"=>"40"),
  array("label"=>"Courses Taken", "type"=>"to_many",
    "linked_table"=>"courses", "maxlen"=>"40", "size"=>"40"),
  array("column"=>"other_info", "label"=>"Other Information",
    "type"=>"textarea", "rows"=>"4", "cols"=>"32"),
);
```

Each element in `$edit_fields` has options `column`, `label`, `table`, `maxlen`, `size`, and `type`. Option `type` can be `numeric`, `text`, `time`, `date`, `email`, `phone`, `textarea`, `to_one`, or `to_many`:

- `textarea`: The input is a string displayed in a textarea. This type requires two other options `rows` and `cols` to specify the numbers of the rows and the columns of the textarea. In our example, the field for column `other_info` uses a textarea.

- `to_one`: First, using type `to_one` allows a user to view or modify the record related to the anchor record via a *one-to-one* or *many-to-one* relationship type. In our example, column `department_id` in table `students` is specified for `to_one`. When the `View` buttion is clicked, the edit form showing the record of the student's department is displayed. Second, a user can search and select a value from the associated table. Assume that the user does not know the ID of the department of interest. By clicking on the `Select` button, he can search and select a department with the search and select forms for departments. This type

needs two other options: `linked_table` and `child_column`. Option `linked_table` designates the name of the table storing the associated record. Option `child_column` indicates the other linking column in the associated table. If this option is omitted, the name of the linking column is the same as the name of the foreign key column in the anchor table.

- `to_many`: The purpose of this type is to exhibit the records related to the anchor record via a *one-to-many* or *many-to-many* relationship type. In our example, the field labeled `Courses Taken` is specified to be `to_many`. When the `Show` button is clicked, all the courses taken by the student are displayed in the select form for courses. This type needs options `linked_table`, `parent_column`, and `child_column`. Option `linked_table` and option `child_column` are similar to those for type `to_one`. However, the child_column can be omitted, when the foreign key of the associated table has the same name of the primary key of the anchor table. Option `parent_column` indicates the *foreign key* column in the anchor table. If this option is omitted, the foreign key column is the primary key of the anchor.

3.4 Info Blocks

An information form can have one *main block* and many *nested blocks*. The main block can display the fields in the anchor record and the records related to the anchor record directly or indeirectly via a one-to-one or many-to-one relationship type. In the form shown in Figure 3.5, the main block contains the fields for columns `student_id`, `first_name`, `last_name`, `age` and `gender` in the anchor table `students`, column `department_name` in table `departments`, and column `country_name` in table `countries`.

Each nested block contains the records expanded from the anchor record directly or indirectly via a one-to-many or many-to-many relationship type. In the `Student` information form, there are two nested blocks. Block 1 shows the languages the student can speak. The records in this block are from table

`languages_spoken`. Block 2 shows all the courses taken by the student and the grades. The records are retrieved from table `courses_taken`, table `sessions`, and table `courses`.

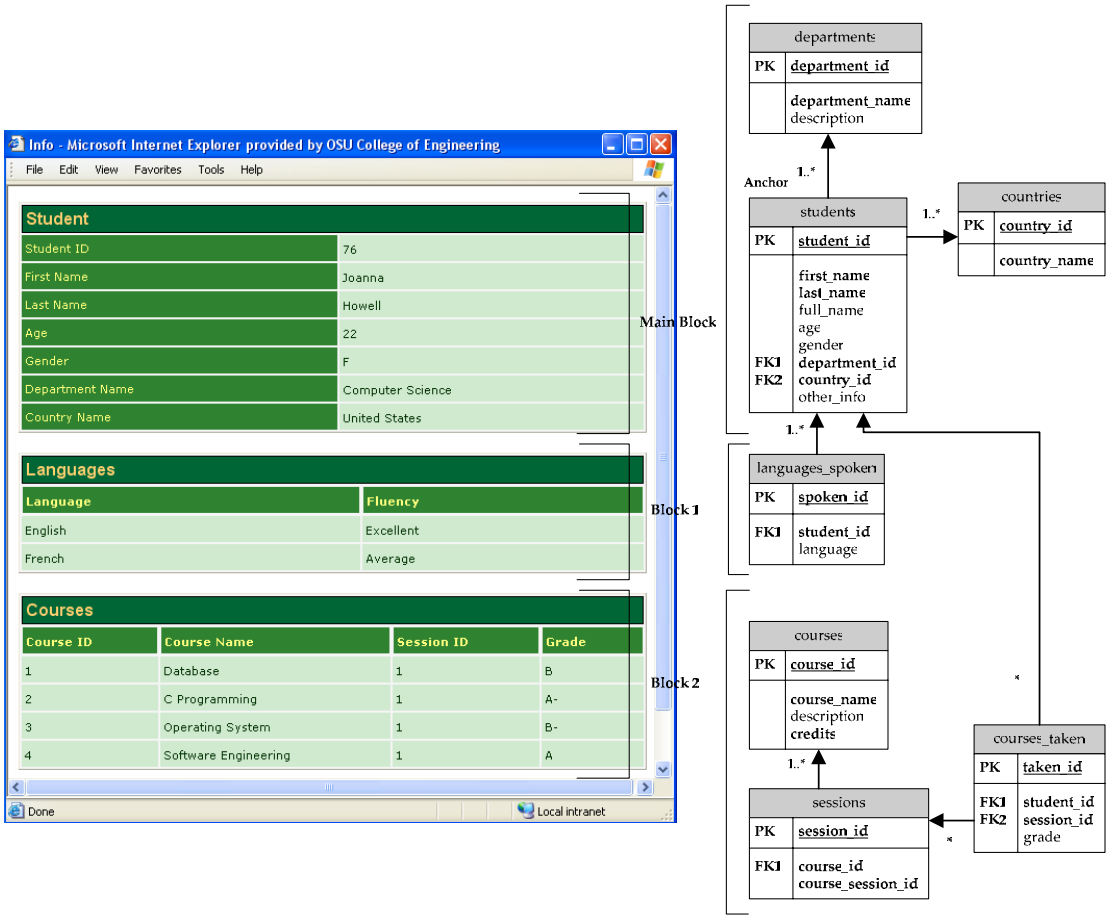

Figure 3.5: Student Information Form.

In the configure file, each element in `$info_blocks` designates a block. To generate the information form shown in Figure 3.5, we must define `$info_blocks` as follows:

```
$info_blocks = array(
  array("title"=>"Student", "type"=>"main",
    "joined_tables"=>"students LEFT JOIN departments
        ON students.department_id=departments.department_id
      LEFT JOIN countries
        ON students.country_id=countries.country_id",
    "fields"=>array(
      array("column"=>"student_id", "label"=>"Student ID",
        "type"=>"numeric", "size"=>"40"),
      array("column"=>"first_name", "label"=>"First Name",
        "type"=>"text", "size"=>"40"),
      array("column"=>"last_name", "label"=>"Last Name",
        "type"=>"text", "size"=>"40"),
```

```
        array("column"=>"age", "label"=>"Age", "type"=>"text",
          "maxlen"=>"40", "size"=>"40"),
        array("column"=>"gender", "label"=>"Gender",
          "type"=>"text", "size"=>"40"),
        array("table"=>"departments",
          "column"=>"department_name",
          "label"=>"Department Name", "type"=>"text",
          "size"=>"40"),
        array("table"=>"countries",
          "column"=>"country_name",
          "label"=>"Country Name", "type"=>"text", "size"=>"40"),
    )),

    array("title"=>"Languages", "type"=>"nested",
      "joined_tables"=>"students LEFT JOIN languages_spoken
       ON students.student_id=languages_spoken.student_id",
      "fields"=>array(
       array("column"=>"language", "label"=>"Language",
         "type"=>"text", "size"=>"40"),
       array("column"=>"fluency", "label"=>"Fluency",
         "type"=>"text", "size"=>"40"),
    )),

    array("title"=>"Courses", "type"=>"nested",
      "joined_tables"=>"students LEFT JOIN courses_taken
          ON students.student_id = courses_taken.student_id
        LEFT JOIN sessions
          ON courses_taken.session_id=sessions.session_id
        LEFT JOIN courses
          ON courses.course_id=sessions.course_id",
      "fields"=>array(
        array("column"=>"course_id", "label"=>"Course ID",
          "type"=>"integer", "size"=>"40"),
        array("column"=>"course_name", "label"=>"Course Name",
          "type"=>"text", "size"=>"40"),
        array("table"=>"sessions","column"=>"course_session_id",
          "label"=>"Session ID", "type"=>"text", "size"=>"40"),
        array("table"=>"courses_taken","column"=>"grade",
          "label"=>"Grade", "type"=>"text", "size"=>"40"),
    )),

  );
```

There are four options in each element of `$info_block`:

- `title`: The title of the block.

- `type`: This option designates whether the block is a `main` block or a `nested` block.

- `joined_tables`: This option designates how the associated tables in the block are linked to the anchor table in the SELECT statement generated. This option is similar to option `joined_table` of `$search_join_table`.

- `fields`: The value of this option is an array, and each element in this array designates a field of the block. If the block is a main block, this option is similar to `$edit_fields` for the edit script. If the block is a nested block. This option is similar to `$select_fields` for the select script.

4. TreeGen Configuration file

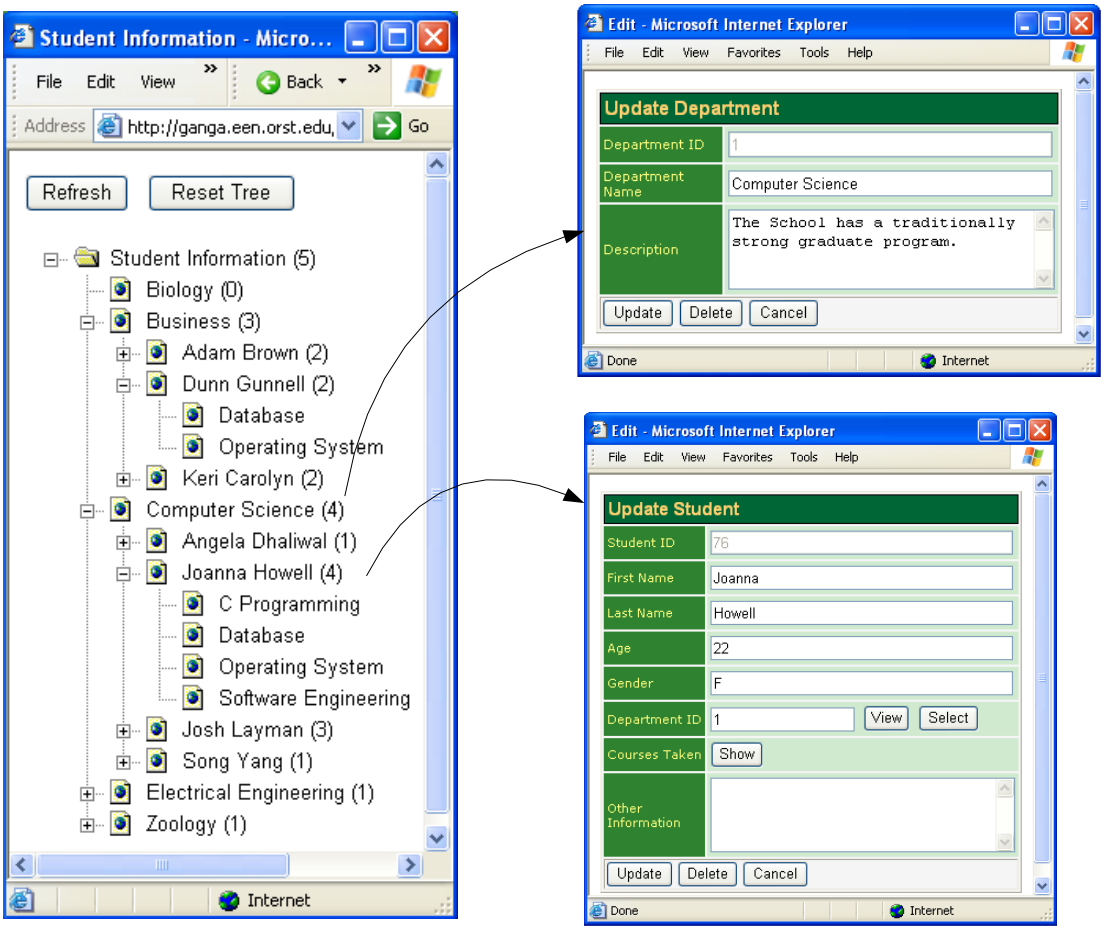

Figure 4.1: TreeView Web Form.

A *treeview form* displays a hierarchy among the records in different tables. Each node in the treeview represents either a data record or a category. A node may have many *child nodes* which represent the records directly or indirectly associated to the record for the node or the records belonging to the category designated by the node.

The treeview shown in Figure 4.1 has three layers. Each node in the first layer represents a department and may have many child nodes representing the students in the department. Those nodes for students are in the second layer. Each student takes many courses. In the figure, the nodes for courses are in the third layer.

When a node in a tree is clicked, an edit form appears to display the complete information about the record of the node. In the treeview shown in Figure 4.1, when the node for a department is clicked, the detailed information about this department is displayed in the `Department` edit form. When the node for a student is clicked, this student's information is displayed in the `Student` edit form.

To generate a treeview form, we need to prepare a *TreeGen configuration file*. In this file, we have main variables and variable `$tree_layers`. To generate the treeview shown in Figure 4.1, the main variables need to be defined as follows:

```
$app_directory = 'forms';
$root_label = 'Student Information';
$tree_prefix = 'student';
$auto_populate = false;
```

`$app_directory`: The path of the folder related to folder ".." where the Web scripts for displaying the detailed information about the record of a node are stored.

`$root_label`: The label of the *root*. A root is the parent node of all the nodes in the first layer. In our example, we use `Student Information` as the label of the root.

`$tree_prefix`: The prefix of the name of a treeview script. In our example, we name the treeview script `student_tree.phtml`. Then the prefix is `student`.

`$auto_populate`: If this main variable is `false`, then when a node is created, its child nodes are not created immediately. The child nodes are created when a user clicks on "?" sign. If the variable is `true`, then when a node is created, all its child nodes are also created.

Each element in `$tree_layers` designates a layer in the tree. Each element can have tweleve options: `this_node_type`, `parent_node_type`, `table`, `primary_key`, `label_column`, `parent_key`, `no_parent_id`, `whereAdd`,

`sqlFrom`, `order_by`, `navigate_url`, and `dir`. To generate the treeview, we need to define `$tree_layers` as follows:

```
$tree_layers = array(
  array(
    "this_node_type"   => "DEPARTMENTS",
    "parent_node_type" => "ROOT",
    "table"            => "departments",
    "primary_key"      => "department_id",
    "label_column"     => "department_name",
  ),

  array(
    "this_node_type"   => "STUDENTS",
    "parent_node_type" => "DEPARTMENT",
    "table"            => "students",
    "primary_key"      => "student_id",
    "label_column"     => "full_name",
    "parent_key"       => "department_id",
  ),

  array(
    "this_node_type"   => "COURSES_TAKEN",
    "parent_node_type" => "STUDENT",
    "table"            => "courses_taken",
    "primary_key"      => "taken_id",
    "label_column"     => "course_name",
    "parent_key"       => "students.student_id",
    "sqlFrom"          => "students INNER JOIN courses_taken
        ON students.student_id=courses_taken.student_id
      INNER JOIN sessions
        ON sessions.session_id=courses_taken.session_id
      INNER JOIN courses
        ON sessions.course_id=courses.course_id",
  ),

);
```

- `this_node_type`: This option designates the node type of the nodes in the layer.

- `parent_node_type`: This option designates the node type of the parent nodes of the nodes in this layer. If the layer is the first layer, then we must set this option as `parent_node_type=>ROOT`.

- `table`: This option designates the name of the table from which the records for the nodes in the layer are retrieved.

- `primary_key`: This option designates the name of the primary key of the table designated by option `table`.

- `lable_column`: This option designates the name of the column whose values are used as the labels for the nodes in the layer.

- `parent_key`: This option designates the name of the primary key of the records for the parent nodes of the nodes in the layer.

- `no_parent_id`: If the value of this option is `true`, then no value is passed from the parent node for retrieving records and creating the nodes in this layer. If the option is omitted, the primary key value of the record for the parent node is used as a condition for retrieving the records and creating the nodes in this layer.

- `sqlFrom`: By using this option, we speicfy the FROM clause of the SELECT statement for retrieving records and creating the nodes in this layer. If this option is omitted, the FROM clause is generated by using the value of option `table`. In our example, the nodes for departments in the first layer do not set this option, so the FROM clause generated is

  ```
  FROM departments
  ```

  The nodes for the courses in the third layer have the option `sqlFrom`. Therefore, the FROM clause is

  ```
  FROM students INNER JOIN courses_taken
     ON students.student_id=courses_taken.student_id
   INNER JOIN sessions
     ON sessions.session_id=courses_taken.session_id
   INNER JOIN courses
     ON sessions.course_id=courses.course_id
  ```

- `whereAdd`: This option designates the extra conditions in the WHERE clause of the SELECT statement for retrieving records and creating the nodes in this layer.

- `order_by`: This option designates the column for sorting the retrieved records. If this option is omitted, the column indicated by option `label_column` is used for sorting.

- `navigate_url` and `dir`: When a node is clicked, a Web form appears to display the detailed information about the record or the category of the node. Option `navigate_url` designates the URL of the Web script. If this option is omitted, the value set in option `dir` is used. Option `dir` designates the path and the folder under which the corresponding edit script is stored. If option `dir` is also omitted, the value of main variable `$app_directory` is used for locating the edit forms or the information forms.

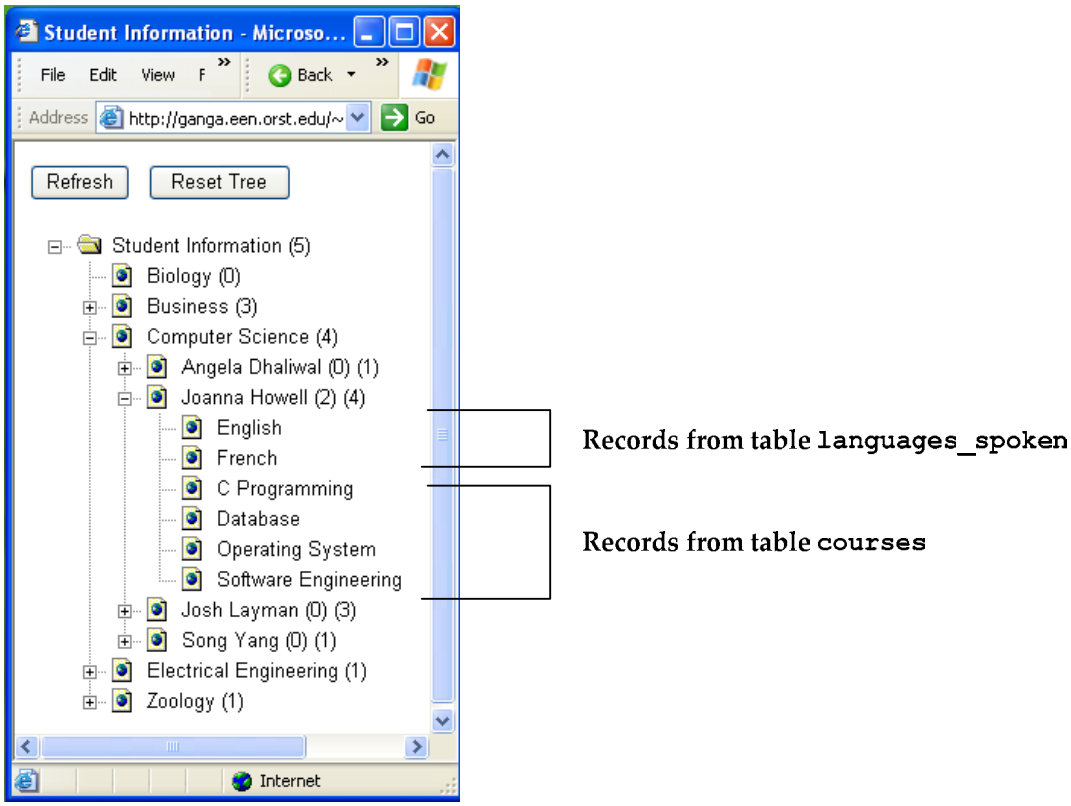

Figure 4.2 Treeview using Option `tables`.

- `tables`: By using option `tables`, we can display the records from different tables. In the third layer of the treeview shown in Figure 4.2, there are the records from table `courses` and the records from table `languages_spoken`. The value of option `tables` is an array. Each element in the array designates a table. Most options explained previously are also used for the elements in the array, except for option `parent_node_type`. To generate the treeview shown in Figure 4.2, we need to define the third layer as follows with option `tables`:

```
array(
  "parent_node_type" => "STUDNETS",
  "tables"           => array(
    array(
      "table"          => "languages_spoken",
      "primary_key"    => "spoken_id",
      "label_column"   => "language",
      "parent_key"     => "student_id",
      "this_node_type" => "LANGUAGES",
    ),
    array(
      "table"          => "courses_taken",
      "primary_key"    => "taken_id",
      "label_column"   => "course_name",
      "parent_key"     => "students.student_id",
      "sqlFrom"        => "students INNER JOIN courses_taken
          ON students.student_id=courses_taken.student_id
        INNER JOIN sessions
          ON sessions.session_id=courses_taken.session_id
        INNER JOIN courses
          ON sessions.course_id=courses.course_id",
      "this_node_type" => "COURSES_TAKEN"
    ),
  ),
),
```

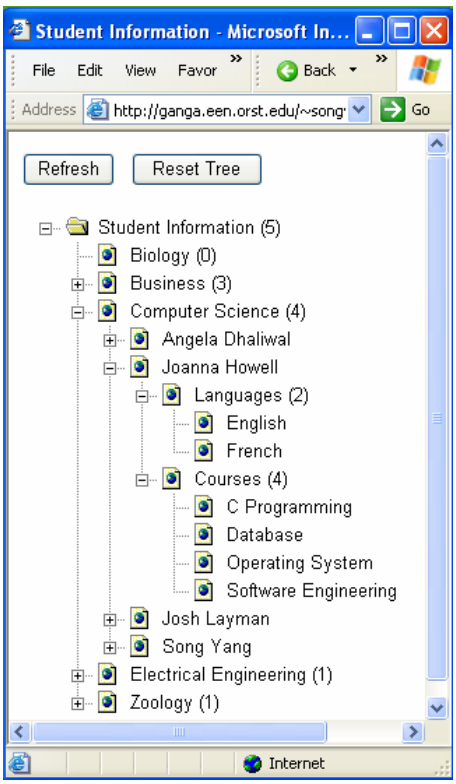

Figure 4.3: Treeview using Option `nodes`.

- `nodes`: By using option `nodes`, we can display category names in a treeview. In the third layer of the treeview shown in Figure 4.3, two categories `languages` and `courses` are displayed. The child nodes of these two categories are the

records from table `languages_spoken` and table `courses`. The value of option `nodes` is an array. Each element in the array designates a category. Those elements can have options `this_node_type`, `text`, and `navigate_url`. To generate the treeview shown in Figure 4.3, we must define the third layer and as follows with option `nodes`:

```
array(
    "parent_node_type" => "STUDENTS",
    "nodes"            => array(
      array(
        "text"           => "Languages",
        "this_node_type" => "LANGUAGE_CATEGORY",
      ),
      array(
        "text"           => "Courses",
        "this_node_type" => "COURSE_CATEGORY",
      ),
    ),
),
```

5. Conclusions

WebGen 5 is a collection of *templates*, which automatically generates Web scripts from *configuration files*. So far, we have developed six templates for creating *search*, *select*, *edit*, *information*, *action*, and *treeview* scripts. Those templates are written in PHP, and the generated Web scripts are also in PHP.

Now, WebGen 5 is being used in real projects. We have observed the following benefits of this Web script generator:

1. Time for developing Web site is significantly reduced.
2. Maintenance becomes easier. Most Web scripts of a Web site project are generated by WebGen 5, rather than written by different programmers. Therefore, they all have the same style.
3. Before WebGen 5, changes on a database structure often caused a large amount of code to be re-written. Now, we only need to modify the configuration file and the program can be easily regenerated.

6. Reference

[1] Parijat Naik; *WebSiteGen3: A Tool For Generating ASP.NET Forms*

[2] George Schlossnagle; *Advanced PHP Programming*; Sams; 2003

[3] Danny Goodman; *JavaScript Bible*; John Wiley & Sons; 2004

[4] Jeff Perkins; *PostgreSQL*; Learning Express; 2001