Automatic Form Generation from Database based on Generative Programming

1Senthil J, 2Brijendra Singh, 1Gaurav Doshi, 1Ishan Gupta, 1Mohit Kumar

1School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu 632014, India.
2School of Information Technology and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu 632014, India.

ARTICLE INFO
Article history:
Received 19November 2013
Received in revised form 23 January 2014
Accepted 27 January 2014
Available online 25 February 2014

Keywords:
Generic Forms, Entity Class, Table Class.

ABSTRACT
Background: Most of the forms needed to fill a database on the internet are hard-coded and requires changes in the forms if any change is needed to be done to the database. So, we propose a generative algorithm for creating input forms on websites from the database tables given as a base. Objective: To develop an algorithm for form generation using generative approach. Results: This paper proposes an algorithm and simulation of effectiveness of using generative approach for form generation. Only database needs to be structured and the program shall generate the input form for it and populate the tables by itself, reducing designing incompatibility and overheads. Conclusion: The proposed method is generic for any type of database and doesn’t require knowledge of scripting for creating input forms. This method could reduce significant time lost in form creation, validation and testing.

INTRODUCTION
Forms are the basic input methods used for taking details of a user online or commit any transactions. However, most of the forms used are hard-coded and are intolerant to the modifications done in the database. This paper proposes an algorithm to create generic forms which gets defined at run time through the target database. Hence, the forms are consistent with the tables and need not be coded explicitly. Only the inclusion of proposed code in program will generate the forms automatically, based on the specified database. As any web service maintains a server side database to keep records of all its clients and for other management purposes, forms emerge as the basic interaction media for handling the user’s inputs. As the services provided increases, so is the need of generating and maintaining new databases and successively the need of creating new forms. The proposed algorithm automates the form creation process and is generic enough to be applied to any service, analysis shows the algorithm is less time consuming and more reliable than conventional methods of form creation. The results have been discussed in later sections.

Requisites:
In any kind of form we have the following types of input areas:
• Simple Text Field
• Radio Buttons
• Checkboxes
• File uploads
• Selection from a List
• Non-editable entries
• Fetching relevant data from other database table (Requires Joining)

For implementing the algorithm, the database scheme needs to be known, also the nature of all its Attributes, i.e. their names, types, ranges, etc. shall be known beforehand.

Model Structure:
The following models (BrankoMilosavljevic et al., 2002) needs to be implemented for application of proposed algorithm

Corresponding Author: Senthil J, VIT University, School of Information Technology and Engineering, Prof Brijendra Singh, Vellore, Tamil Nadu 632014. India.
Mob No-919791326500; E-mail: brijendrasingh@vit.ac.in, Postal Address: Room No-510, A-01, SJT, VIT University, Vellore, Tamil Nadu, India.
(i) Entity class is a specific class which contains array variables to store the attribute names that are of various types as illustrated above.

(ii) A generic Table Class which serves as a base class for all the derived entity classes contains:
   a) All the array variables present in its derived entity classes, with generic data which is applicable on all the tables of the database.
   b) Functions to retrieve the details about the attribute of a specific table, from the database.

(iii) Display class consisting of following:
   a) Functions to form JavaScript arrays for different type of attributes.
   b) Functions to display the form rows without customization (with simple text inputs).
   c) Function to invoke the JavaScript functions to convert the non-customized forms into customized ones as per the attribute types & the styling of the form fields as per the specification.

The hierarchy of the classes is shown in Fig 1.

---

**Fig. 1:** Hierarchy of Classes

**Methodology:**

For better understanding of the algorithm components used are explained (Seung C. Leem and Ashraf I. Shirani, 2004) an entity ‘course’ has been considered for which an input form is required, this is done in 5 steps. The schema of the course table is shown in Fig 1(a).

![Course Table Schema](image)

**Table 1:** Types of attributes and corresponding fields in the course entity

<table>
<thead>
<tr>
<th>Types</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchButtonType</td>
<td>Owner ID [from user], Dept ID [from Dept]</td>
</tr>
<tr>
<td>optionSelectType</td>
<td>Year, Sem, Type</td>
</tr>
<tr>
<td>nonShowType</td>
<td></td>
</tr>
<tr>
<td>uploaderType</td>
<td>Image</td>
</tr>
</tbody>
</table>

(ii) All table attributes from the database schema are retrieved along with their types by using the following query:

SHOW COLUMNS FROM `{tableName}`;

Each tuple vector contains the following information:

- **Fields:** The value that’s present in the specific column.
- **Type:** The type of the attribute, i.e. integer, real, string, char, etc.
- **Null:** Whether null values allowed or not in the column.
- Key: Is the column a primary, foreign or a unique key.
- Default: The default value of the column if no input is given.
- Extra: The extra information associated with the column like comments, etc.

Out of these six vector tuples, only ‘Fields’ and ‘Type’ for the basic format, which provide the following (Table 2).

<table>
<thead>
<tr>
<th>Field</th>
<th>Id</th>
<th>Name</th>
<th>Desc</th>
<th>Code</th>
<th>OwnerID</th>
<th>DeptID</th>
<th>Year</th>
<th>Sem</th>
<th>Type</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Int(10)</td>
<td>Text</td>
<td>Text</td>
<td>Varchar(24)</td>
<td>Int(10)</td>
<td>Int(10)</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>Var</td>
</tr>
</tbody>
</table>

Table 2: Tables attributes and their corresponding types

(iii) The form is now displayed in a non-customized way i.e. in a plain HTML format (Quan Liang Chen et al., 2007) consisting of the entire column entries in the table, with each associated to the input type text. Suitable names and ids are given to the inputs to have a better control on them through various client and server based scripts as well as with style-sheets.

Example:-

```html
<div class='table-row-label'>"Column Name."</div>
<div class='table-row-field' id='{COLUMN_NAME}_input'>
<input type='text' name='{COLUMN_NAME}' id='{COLUMN_NAME}' value='{$value}'>
</div>
```

The value of COLUMN_NAME will be stored in the ‘field’ column extracted in step 2 and the ‘type’ column can be used to do the validations.

The output of this step is as follows:

![](image)

Fig. 2: Generated unformatted plain form

(iv) Every option SelectTypes, nonEditTypes, nonShowTypes, radioSelectTypes, checkboxSelectTypes, uploaderTypes, searchColumnType array variables are stored in form of JavaScript arrays, which are then sent to the respective functions for form manipulation.

- showSearchBtn(searchColumns)

This function is used for displaying the search button which will get the required data. Eg. If in a form field an ID is required which is an integer, as remembering all the ID’s is not possible. The class entity is joined with the table which contains the association of Names with IDs. Join is used to maintain the integrity of the database.

Thus, for every value in the array searchColumns, the corresponding row from the non-customized form is obtained and converted to search input type with the help of client-side scripts, onclick functions are assigned to them, so as to pass the following:

- Name of the table that needs to be joined.
- Attribute to be taken from table containing Name-ID association, which are later fetched through the database queries.
showSelectBox(optionSelectKeys)
This function stores the attribute columns which require a ‘select’ dropdown menu. For every value in the array optionSelectKeys, the corresponding rows from the non-customized form are obtained and converted into the Select/Option type, providing options as per the column name.

readOnlyNonEditKeys(nonEditKeys)
This function stores the attribute columns which are non-editable. For every value in the array nonEditKeys, the corresponding rows from the non-customized form are obtained and converted into non-editable types.

hideNonShowKeys(nonShowKeys)
This function stores the attribute columns which are hidden from the form. These may be the values that will be automatically filled. For every value in the array nonShowKeys, corresponding rows from the non-customized form are obtained and hidden.

showUploaderKeys(uploaderKeys)
This function stores the attribute columns which are FILE type and needs to be uploaded.

After the application of all the above functions, we render the final form (Jian Hu et al., 1996) with all appropriate types of inputs. The form obtained in this step is used to get inputs from the user, for the table. All the input fields are consistent with what is required. The output is shown in Fig 3.

![Fig. 3: Final formatted output form](image)

Analysis:
Repetitive analysis was done on a set of 20 forms of varying fields and complexity (FilippoRicca and Paolo Tonella., 2001) the findings were averaged and summarized comparing various aspects of form generation, validation and errors incurred using plain HTML and the proposed method. The findings are shown in Table 3.

<table>
<thead>
<tr>
<th>SNo</th>
<th>Estimation Criteria</th>
<th>HTML Coding</th>
<th>Proposed Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Time taken in generating effective lines of code for form.</td>
<td>5-8 minutes</td>
<td>10-20 seconds</td>
</tr>
<tr>
<td>2.</td>
<td>Risk of incompatibility of form with the database</td>
<td>Moderate</td>
<td>No such risk</td>
</tr>
<tr>
<td>3.</td>
<td>Skills requirement for scripting to create a form consistent with tables.</td>
<td>Low</td>
<td>No skills required</td>
</tr>
<tr>
<td>4.</td>
<td>Testing time to ensure successful input forms generation.</td>
<td>Moderate</td>
<td>No testing required</td>
</tr>
<tr>
<td>5.</td>
<td>Delay in connecting to database before form generation.</td>
<td>No delay</td>
<td>Small delay</td>
</tr>
</tbody>
</table>

Advantages:
The results from the analysis clearly show that, the delay caused in generation of forms by connecting to the server’s database and processing its attributes, has been traded off with the reliability and speedier implementation of the overall system. As observed through the quantitative models used for analysis, overall the algorithm proves to be more efficient than the conventional methods of form generation. It is much more
reliable, reusable, generic and suitable for web services generating various forms and maintaining a large amount of client based database. However, for web services which doesn’t require varied databases and have little interactions conventional methods are rather more appropriate.

**Conclusion:**

The proposed algorithm presented a generic and generative solution for form generation using the database attributes. This method is generic for any form of database and doesn’t require knowledge of scripting for creating input forms. As interactive web pages are on the rise, the proposed method could reduce significant time lost in input form creation, validation and testing.

**REFERENCES**


