ELEKTROTEHNIŠKI VESTNIK 84(3): 76–84, 2017
ORIGINAL SCIENTIFIC PAPER

Development of a framework for dynamic creation of web-interfaces to support data acquisition in clinical settings

Aleš Smrdel[†]

Faculty of computer and information science, University of Ljubljana,
Večna pot 113 , 1000 Ljubljana, Slovenia

[†] *E-mail: ales.smrdel@fri.uni-lj.si*

Abstract. We present a new framework for dynamic creation of web-interfaces to acquire data and to manage the acquired data. The requirements for the framework are such that it is suitable for acquiring the person-related data and requires minimal programming skills to set it up and use. The developed framework is connected to a relational database management system and reads the structure of the records from the database. According to the structure of the records, the framework generates a web-interface consisting of several web-pages. The structure of the records is also used to manage the data in the database. Positioning of the web-page elements is achieved by using cascading style sheets. These features enable changes to the existing structure of the records which are automatically reflected in the generated web-interface without the need to change the underlying code. All the above features, in combination with the person-related data acquisition design, make this framework unique, and enable changes to the structure of the records and use of the framework for different purposes without the need for altering the programming code. We also present a case-study of using the framework.

Keywords: relational database, web-interface framework, dynamic creation of web-interface

Razvoj ogrodja za dinamično kreiranje spletnih vmesnikov za podporo zbiranju podatkov v kliničnem okolju

Predstavljamo novo ogrodje za dinamično kreiranje spletnih vmesnikov za zbiranje in upravljanje s podatki. Zahteve pri razvoju ogrodja so postavljene tako, da je ogrodje primerno za zbiranje podatkov, povezanih z osebami, obenem pa zahteva minimalno znanje programiranja za postavitev in uporabo. Razvito ogrodje se poveže z relacijsko podatkovno bazo in prebere strukturo zapisov iz podatkovne baze. Glede na strukturo zapisov ogrodje kreira spletni vmesnik, ki sestoji iz nekaj spletnih strani. Struktura zapisov je uporabljena tudi pri upravljanju podatkov v podatkovni bazi. Postavitev elementov spletnih strani je dosežena z uporabo kaskadnih slogovnih predlog. Te lastnosti omogočajo spremembe v obstoječi strukturi zapisov, ki se avtomatsko odražajo v kreiranem spletnem vmesniku, brez potrebe po spremembi osnovne kode. Zgornje lastnosti v kombinaciji z zbiranjem podatkov, povezanih z osebami, naredijo ta vmesnik tudi edinstven, obenem pa omogočajo spremembe strukture zapisov in uporabo ogrodja za različne namene, brez potrebe po spreminjanju programske kode. Predstavljamo pa tudi študijo primera uporabe ogrodja.

Received 20 January 2017
Accepted 12 April 2017

1 INTRODUCTION

Recent years have witnessed an immense growth of the Internet. The popularity of the Internet has also encouraged the trend of providing documents in electronic form that can be accessed over a network [1]. This has also made the data acquisition in different fields much easier, although the structure of the data can be very versatile for different fields. Quite often the data to be acquired are connected to a person. One such field, where the central entity is a person, is the clinical environment, where large amounts of data are acquired daily. The majority of the data are acquired as a part of an every-day regular clinical practice either during regular check-ups or during emergencies. Sometimes the data acquisition is performed also in a scope of different studies where the data can be much more specific to satisfy the needs of a given study.

The data acquisition as a part of a regular clinical practice is ordinarily performed with applications which can perform different tasks, such as acquiring data, connecting to centralized Relational Database Management System (RDBMS), storing data in a relational database, and retrieving the stored data. These applications are usually developed for a large number of users, e.g. several hospitals or medical institutions, making these applications hard and impractical to use for acquiring

the data in a scope of a single specific study, which thus has to be implemented in other ways. Quite often the data are initially acquired in a paper form. These data have to be transformed later into an electronic form in order to perform data analysis. As an alternative, special web-applications or web-interfaces capable of acquiring and storing the data into the relational database can be used. To produce different web-interfaces, general frameworks such as *TurboGears* [2] or *Ruby on Rails* [3] can be used. Such frameworks enable rapid development of web-interfaces. Since these frameworks cover a wide range of possible applications, they have to allow for full configurability. The existing frameworks also require changes to the programming code (for the interface and for the controller) when changing the structure of the records or when data validation is needed. As a consequence, they require extensive programming skills, which the person performing a specific study might lack. Sometimes changes in the structure of the records trigger extensive changes in the web-interface to which the user has to adapt, thus minimal alterations to the web-interface are desired. When using such general frameworks to set up a web-interface, the person implementing such web-interface usually has to define the structure of the data and also has to write at least some programming code to enable interaction with the user and the database, and for data validation. On the other hand, the web-based tools for the data acquisition, which allow for a relatively easy set-up, might be used. However, these web-based tools might, especially in the case of a clinical environment, raise security issues due to the sensitive nature of the data being acquired, because the acquired data is stored in public remote locations. Since the studies are often contained within a given department, the storage and access to a public remote location is not required. These considerations suggest the development of a new framework, which would allow for an easy set-up, would enable generating web-interfaces, and could be accessed through a local network. When developing the framework for a person-oriented data acquisition, it becomes obvious that such data have a similar structure: a person represents a database entry with a unique identification (ID), for whom data in several subcategories can be acquired. In case of a clinical environment, the subcategories can be different examinations, which can again have a similar structure. The similarity of the person-oriented data structures can then be used to develop a framework applicable to different fields. The adjustment enables an easy set-up while the framework does not require any additional programming besides defining the structure of the records, which might be quite similar even for different fields, thus enabling reuse of an existing structure.

In this paper we present a new framework for generating dynamic web-interfaces. The framework is developed as a tool for facilitating a move from acquiring the data in a paper form to an electronic data acquisition. The framework exploits the similarity of the data structures, which is usually present when acquiring data pertaining to a person. This framework can be used for different purposes, requires minimal alterations to an existing structure of the records, while completely avoiding the need for any changes in the underlying code-base of a particular web-interface. This is possible due to the ability of dynamically creating web-pages and managing the acquired data using only the structure of the records in the database. The web-page elements (widgets) used for data acquisition and representation are created according to the structure of the records and are positioned using the cascading style sheet (CSS) files. We also present a case study of using the developed framework to dynamically create web-pages for acquiring the data about pregnant women. The dynamically created web-pages were developed in collaboration with the University Medical Center Ljubljana, Department of Obstetrics and Gynecology.

2 METHODS

At the beginning of the development of the framework we laid down the following requirements:

1) capability of producing a web-interface comprised of several web-pages;
2) easy to set-up and use also for the non-programmers;
3) capability of adding data columns to (or removing them) the structure of the records without the need for altering the programming code, since the need for additional data might arise during the use of the web-interface.

The third requirement also imposed a restriction that the framework should allow for a partial data entry, i.e., some of the data could be missing. The framework also should fully support create, read, update and delete operations in the database. The above requirements also imposed several constraints on the structure of the records. The main table (relation) of the structure of the records (the topmost table in the database) is given a predefined name and an identification column (attribute), which also has a predefined name. This constraint is necessary in order for the framework to be able to access the structure of the records without the need to change the programming code. Several *supporting tables* with the predefined names are also required. The *supporting tables* and their structure are shown in Fig. 1. These *supporting tables* are used to describe, among other things, the data types and relations between the data types and web-interface widgets. Due to the predefined names and the *supporting tables*, a separate database in the given RDBMS is created for each structure of the

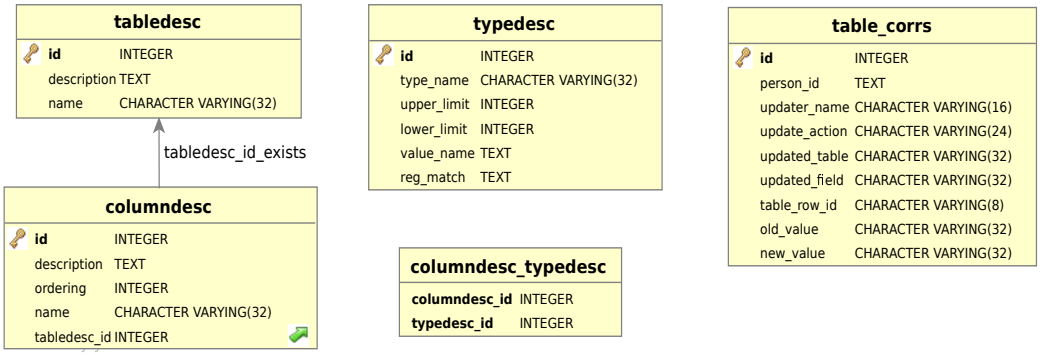

Figure 1. Structure of the *supporting tables* required by the framework.

records and thus for each web-interface.

We developed a framework using the Python programming language [4]. This language was selected because of its versatility and numerous modules, which can be used to implement some of the required functionalities. We used the *CherryPy* module [5] as the basis for the framework. The *CherryPy* is a Python-based framework offering basic HTTP functionality, and allowing web-applications to be developed in the same manner as the desktop applications. We also used an *SQLObject* object relational mapping (ORM) library [2] as an underlying basis for the connectivity to the RDBMS and interaction with the databases.

2.1 The supporting tables and the structure of the records

The *supporting tables* are created (or updated) automatically at the time of creation (or modification) of the structure of the records. The *supporting tables* allow interpretation of the structure of the records, selection of widgets according to the column type, display of the data from the database, and also logging of the data modifications. The required *supporting tables* are (refer also to Fig. 1):

- **tabledesc**: containing a list and the description of all tables which constitute the structure of the records (the *supporting tables* do not constitute the structure of the records and are not included in this table);
- **columndesc**: containing a description for each column in every table (the columns in the *supporting tables* do not constitute the structure of the records and are not included in this table) and information to which table this column belongs;
- **typedesc**: containing the type of each column in every table (the columns in the *supporting tables* do not constitute the structure of the records and are not included in this table);
- **columndesc_typedesc**: containing the data about the relationships between the table columns and data types described in the tables **columndesc** and **typedesc**; and
- **table_corrs**: containing the data about all changes made to the entries in the database.

The step of defining the structure of the data cannot be avoided. This step can be performed in different ways and requires some programming skills. We tried to simplify this step as much as possible. Instead of defining the structure of the records using the Structured Query Language (SQL) queries, we used an additional layer. This layer helps in masking the SQL query syntax and also enables an additional data validation with optional constraints. The layer is constituted from classes written in Python. Each table in the database is represented using a Python class, which describes the table structure. Each class is composed of variables referencing objects of different types. Each object represents a table column of a given type. To define a column type, we used extended *SQLObject* classes. The classes were extended with additional properties enabling an automatic data validation. We implemented only classes for those types which are necessary for the framework: 1) *Int*; 2) *Float*; 3) *String*; 4) *Date*; 5) *DateTime*; 6) *Enum*; 7) *Bool*; 8) *ForeignKey*; and 9) *MultipleJoin*. The first seven classes represent column types used for storing the data, while the last two classes specify the relationships between the tables. Using these extended classes, additional constraints (properties) can be specified for each column of a given data type if desired, e.g., the upper limit for a value. The use of the constraints for a given column enables an automatic validation of the entered data, without the need to write the validation code. Using the defined classes, the framework creates the structure of the records in the database. When creating the structure of the data, the *supporting tables* are also created. The structure of the records should also be normalized in order to avoid a possible update, deletion and insertion anomalies, which might arise from the user input [6], but this step is left to the user.

2.2 *The structure of the framework*

The developed framework consists of several classes.

- The most important is the class responsible for manipulating the structure of the records. This class provides an access to the structure of the records and the *supporting tables* in the database. The information obtained through this class is used to construct the web-interface. It also enables creating and deleting the tables constituting the structure of the records. This class is also responsible for modifying the existing tables of the structure of the records by adding or deleting columns in a given table and also for modifying the *supporting tables*.
- The classes defining the data types, which are derived by extending the *SQLObject* classes, enable data validation with added constraints. These classes are also responsible for generating the necessary meta data and are required for creating tables and table columns.
- The connectivity class is responsible for communication between the user and the database. This class enables accepting and processing the user requests, retrieving and updating the data, and generating new entries in the database.

The flowchart depicting the interaction of the framework with the user and the database is shown in Fig. 2, while Table 1 shows transitions from a given web-page to a resulting web-page according to the user input, together with a possible operation performed on the database. Upon the user request, the framework presents the user with a web-page, which is initially the page for entering the person ID. The user then performs an action which can vary according to the given web-page. The action can be either the request for a particular page (either for the web-page with the buttons to substructures or for a web-page containing the selected substructure), submission of the data, or cancellation of editing a selected record. The action is sent together with the relevant data to the framework, where it is processed. The framework then performs an interaction either with the user (if a new data entry has to be created, or if the users has the choice of updating either an existing record or a new one, or deleting an existing record), or the database (creating a record, retrieving the data, writing the data, or deleting a record). To interact with the database, the framework forms an SQL query and sends it to the database. The database performs the SQL query (which can be one of the create, read, update or delete operations) and returns the result. The framework processes the result, generates a web-page and sends it to the user. The web-pages showing the existing subsections and for editing the selected subsection are generated dynamically according to the structure of the records, thus all changes made to the structure of the records are automatically reflected in the generated web-pages.

The type of each displayed widget in the web-page is determined by mapping the relations between the column types of the structure of the records and the HTML elements. The framework uses the text field widgets (“input” element of the type “text”) for the *Int*, *Float* and *String* column types. To present the *Date* and *DateTime* column types, it uses compound widgets constructed from drop-down menus (“select” element containing several “option” elements) to present years, months and days; and also for hours and minutes for the *DateTime* column type. To present the *Enum* column type, it uses the radio button group widget (“input” element of the type “radio” for each possibility) while for the *Bool* column type the framework uses the check box widget (“input” element of the type “check”). The *ForeignKey* column type is used to cross-reference the tables and is not displayed. The *MultipleJoin* column type is handled in two ways, depending on the level in which it appears in the structure of the records. If this column type appears in the topmost table, the description is displayed in a form of a button, which a user can click to access the appropriate part of the structure of the records, otherwise the substructure to which this column points is displayed as a part of the web-page where the description is displayed as a section heading.

The desired placement of the displayed widgets is achieved using the CSS files. For the appropriate positioning, each widget has to be uniquely identified and has to be identified in the same way for each record, given the structure of the records. From the available possibilities we selected the attribute *class*. The value of this attribute for the widget representing a given column is constructed from the names of the tables leading from the topmost table to the given column and its name. In this way we ensure that the value of the attribute *class* for a given widget is unique, consistent and the same for different data entries, since the names of the tables and the columns do not change for different records.

3 CASE STUDY

The following case study shows the use of the framework to create the web-interface for acquisition of data about pregnant women. The web-interface was developed in the scope of the effort to establish a database containing data about pregnant women. The database is intended as a support to the medical staff during pregnancy, with an additional aim of helping to predict an impending premature labor and to help manage pregnancies which could result in a premature labor. Early prediction of a premature labor is of utmost importance since it is one of the leading causes of morbidity and mortality in infants [7]–[9]. Previous studies [10]–[12] identified numerous risk factors which elevate the possibility of a premature labor: e.g. diabetes,

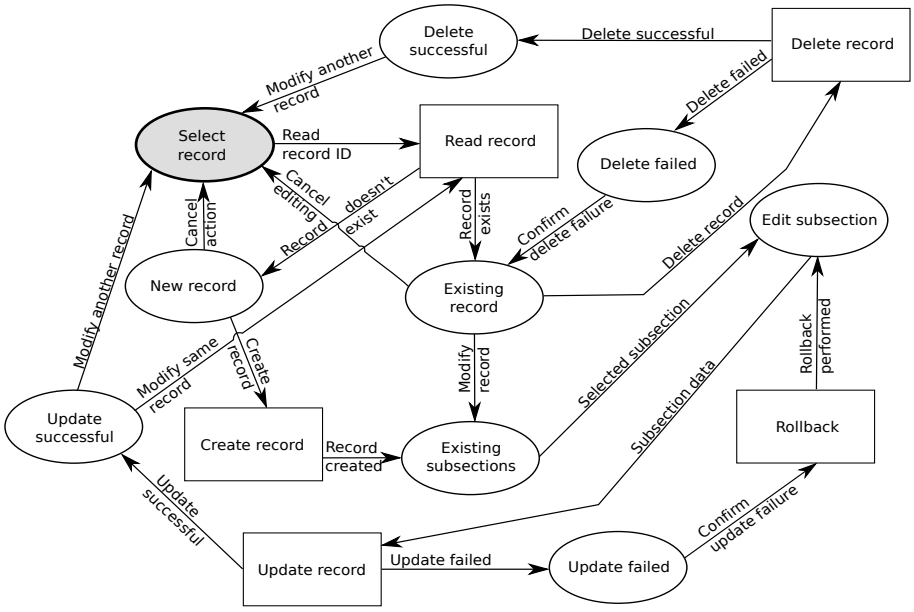

Figure 2. Flowchart depicting the interaction of the framework with the user and with the RDBMS. The ovals represent web-pages presented to the user, the rectangles represent requests (SQL queries) performed by the RDBMS, while the arrows represent the user input processed by the framework (from the ovals), or the result of the interaction with the RDBMS (from the rectangles). The shaded oval indicates the starting point of the interaction or the entry page.

Table 1. Transitions between the consecutive web-pages of the interface together with the actual user input and the operations on the records in the database (DB).

Initial web-page	User input	DB operation	Resulting web-page
Select record	Enter record ID	Read record	Existing record
			New record
Existing record	Modify record	/	Existing subsection
	Delete record	Delete record	Delete successful Delete failed
New record	Cancel action	/	Select record
	Create record	Create record	Edit subsection
Delete successful	Select record	/	Select record
Delete failed	Confirm failure	/	Existing record
Existing subsection	Selected subsection	/	Edit subsection
Edit subsection	Subsection data	Update record	Update successful
			Update failed
Update successful	Modify same record	Read record	Existing subsection
	Modify another record	/	Select record
Update failed	Confirm update failure	Rollback	Edit subsection

conization, hypertension, smoking, uterine abnormalities. A database containing the data of pregnant women, containing relevant risk factors for a premature labor, can help in predicting and postponing an impending premature labor, since it can be used to regularly update the information about pregnancies. Each pregnant woman can thus be monitored continuously, the data can be entered into the database promptly, and screening for pregnancies with an elevated risk of a premature labor can be carried out more thoroughly.

At the beginning we determined the structure of the records which, is defined as a collection of the data to be acquired. The physicians specified the relevant data to be acquired as a collection of the previously identified risk factors [10], and other data which they felt could also help in predicting a premature labor. When setting up the database and the web-interface, it was ensured that the personal data-protection law is respected and only an authorized personnel can access the data stored in the database. The data to be acquired were divided into several sections according to an established protocol for monitoring pregnancy, which assumes several visits of a pregnant woman to the clinic, including birth giving. The first section contains the general data about examinations, e.g., examination dates, the second, third and fourth sections contain the specific data for each of

the three planned examinations, while the fifth section contains the data about the birth. Each of these sections is represented with a table in the structure of the records and can contain additional subtables.

The structure of the records could be generated in the database in different ways. This is the only step in setting-up a web-interface that requires some programming knowledge. We simplified it by using an abstraction layer. In order to specify a table in the database, a class in Python is used. The topmost class for our case study web-interface written in Python is shown below:

```
class Person(MetaDB.Table):
  personID=MetaDB.String(desc='Person ID',
                unique=True,order=1)
  basicInfo=MetaDB.MultipleJoin('BasicInfo',
                desc='Exam data',order=2)
  firstExam=MetaDB.MultipleJoin('FirstExam',
                 desc='1st exam',order=3)
  secondExam=MetaDB.MultipleJoin('SecondExam',
                desc='2nd exam',order=4)
  thirdExam=MetaDB.MultipleJoin('ThirdExam',
                desc='3rdexam', order=5)
  birth=MetaDB.MultipleJoin('Birth',desc='Birth', order=6)
```

The name of the class represents the name of the table in the database. The objects for which the variables (properties) represent either the table columns or the tables in the database are of a given type (extended *SQLObject* type). In the definition of the relational type, e.g. *MultipleJoin*, the first parameter represents the name of the table in the database to which this column relates and is equal to the name of the Python class which describes the table in the database. The names of the tables are obtained from the Python class names by transforming them into a lower case and appending an underscore before each new non-leading word, e.g., BasicInfo is converted into **basic_info**. Parameter *desc* contains a description of the column and is used to display either the column value or a substructure of the structure of the records. Parameter *unique* identifies whether the column values should be unique for different records, while *order* determines the order of the data processing. The following excerpt from the case-study class defining the **birth** table (the last line of the above example defines the reference to this table) shows the data column definition for the Apgar score after the first minute:

```
  scoreApgar1=MetaDB.Int(desc='after 1. minute',order=2,
                default=None, upperLimit=10,lowerLimit=0)
```

The above example also illustrates the possibility of defining the default value of a column, which can be used to automatically fill the record when desired, and the constraints for the upper and lower limits for the column value. According to the specified constraints, an automatic data validation is performed when submitting the data. Fig. 3 shows the structure of the records created for the task of acquiring the information about pregnant women and is comprised of 36 tables. For the above case study defining class Person, table person exists in the database to which several subtables are related (**basic_info**, **first_exam**, **second_exam**, **third_exam** and **birth**).

Figs. 4 and 5 show two web-pages generated with this framework as a part of the web-interface. Initially, the user logs in to access the web-interface. Then, the user enters the ID of a person. If a record does not exist, the framework generates a web-page offering the user a possibility of creating a new record or aborting the action. If the record either already exists or was just created, the user can select a substructure of the structure of the records as shown in Fig. 4 (refer also to the example of defining class Person and to Fig. 3, tables related to table **person**). Each substructure is represented with a button containing a description obtained from the database (created using the example class Person). When a substructure is selected, the framework displays all the data in the substructure as another web-page.

Fig. 5 shows the upper part of the web-page representing the section containing the birth data, accessed using the bottom button in Fig. 4. The user enters the missing data and updates the existing data. The framework allows part of the data to be missing and can be entered during subsequent visits. Upon the submission, the data are validated. If the validation fails, the web-page describing the error and the cause of the error is shown to the user. The user returns to the previous page to correct the data and repeats the submission. If the validation succeeds, all the changes are stored into the database, and the user either continues updating the current record or returns to the first page.

During the development we tested the usability of the framework and the web-interface generated by the framework using this case-study. The testing was performed by the users, for whom the framework was developed. The users quickly learned to use the web-interface. The testing of the framework revealed a low efficiency, as there was too much information presented on a single web-page. The users commented that the web-interface was over cluttered, therefore the information was hard to find. This made the generated web-pages difficult to use. To address this problem, we imposed additional restriction to the structure of the records. The topmost table contains only an identification column and references to tables, where each table contains the related data, e.g., the data about a given examination. The architecture of the framework is then altered in such a way that instead of displaying the entire structure of the records, the framework displays these references to tables as buttons (refer to Fig. 4). These buttons then allow an access to the desired part of the structure of the

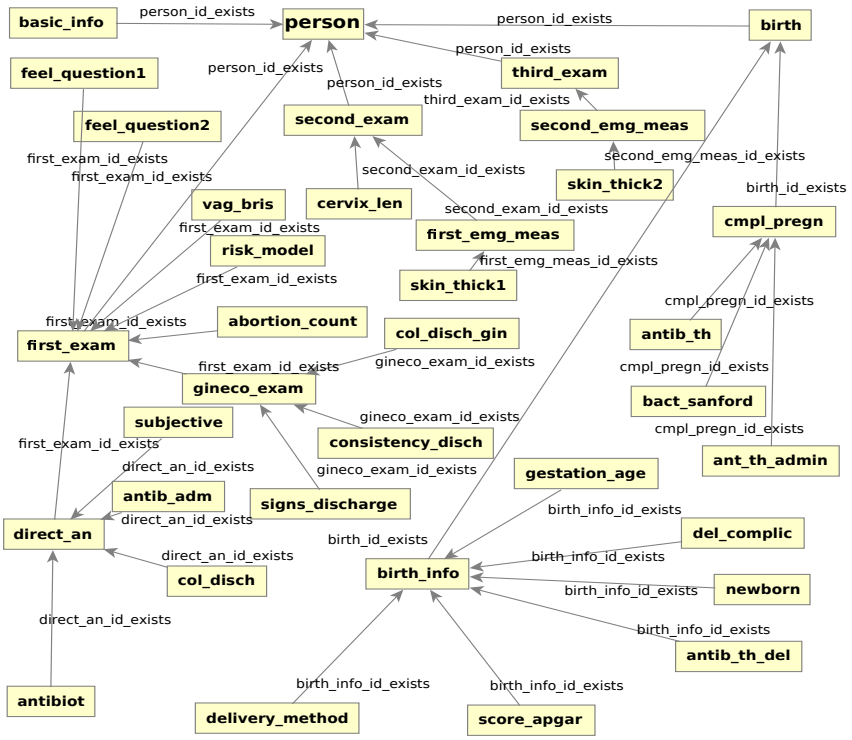

Figure 3. Structure of the records for the web-interface developed for data acquisition about pregnant women. Shown are the tables (names of the tables are included within the boxes) of the structure of the records. The arrows between the boxes represent relations between the two connected tables (the text over the line indicates the foreign key) in the structure of the records where the direction indicates the path from a subtable to the table

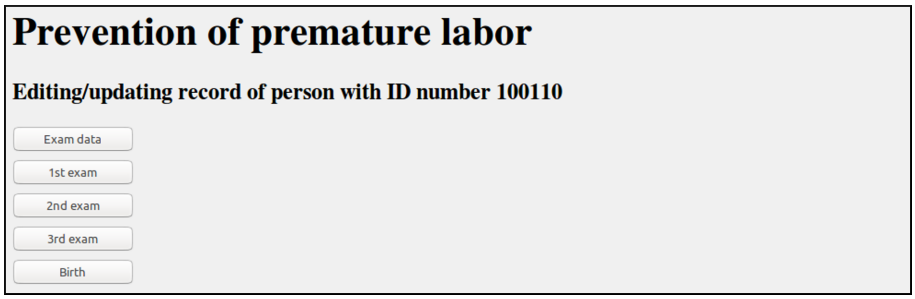

Figure 4. Page for displaying and selecting part of the structure of the records for updating.

records. Each part of the structure of the records is then presented as a web-page. The complete web-interface can then be displayed as a set of several logically separated web-pages, where the data are easier to access. The users also did not report making many errors, which can easily be undone by reediting the affected record.

4 Discussion

The developed framework allows creating web-interfaces without the need for writing or altering the programming code. This feature is possible due to the unique architecture of the framework, which enables an automatic creation of the web-interface according to the structure of the records in the database, and not through the hard-coding of the interface. The framework can access RDBMS and dynamically create the web-pages constituting a web-interface using the structure of the records in the database. The changes in the structure of the records (and the *supporting tables*) automatically cause the changes in the generated web-interface.

The initial aim of the project was to develop a web-interface capable of adding, modifying and deleting data in the database. Later we decided to develop a general framework capable of manipulating the person-oriented data stored in the RDBMS, which would require a minimal or no programming intervention when setting up a new web-interface or altering the existing one.

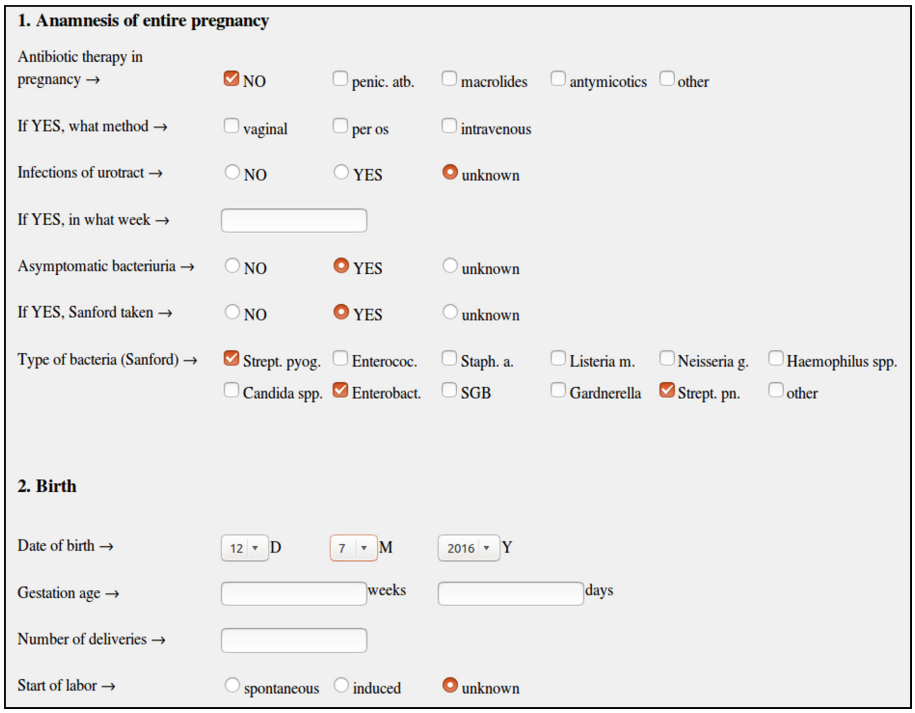

Figure 5. The upper part of the page for displaying and updating the section containing the birth data.

By dynamically creating the web-interfaces according to the structure of the records, we managed to eliminate almost all of the programming required by other frameworks. Defining the structure of the records still requires defining the tables and relations. This step cannot be avoided since the structure of the data to be acquired has to be specified. We minimized the complexity of defining the data structures using SQL through an abstraction layer. This layer is implemented with classes written in Python and hides most of the underlying complexity. It also allows, using the extended data types, an additional data validation, for which no additional programming code is required. Due to the similarities in the data collected for different studies, the classes used for creating the structure of the records can often be reused with only minimal adjustments. The framework dynamically creates a web-interface and uses the CSS files for positioning, which can be defined using one of the available visual editors. Thus, there is no need for any changes to the framework even if the structure of the records changes. Only the appropriate CSS files need to be changed or rebuilt. The benefit of this approach is that it transfers the task of creating a new web-interface or modifying an existing web-page when creating or changing the structure of the records to a much easier task of creating or modifying the CSS file. This makes the developed framework suitable for the use also for users with no extensive programming knowledge.

At least some of the developed functionalities, e.g. ORM or automatically generated web-interfaces, provide also other frameworks. But the developed framework combines the functionalities in such a way that it offers several advantages over other frameworks. The unique feature of creating a web-interface on the basis of the structure of the records without the need to specify the desired relation to object mapping, makes this framework easy for setting up and allows for changes in the data structure without the need for adding or changing the code responsible for data management or web-page generation. The specific person-oriented structure of the records makes it suitable for the use in different fields, where such data structures are quite common. The structure of the framework and also the extended data types also allow for setting up of a new web-interface without writing any programming code. This approach also facilitates an easy integration of different relational databases. The drawbacks of this approach are restrictions in grouping the data, and the predefined names of the topmost table, *supporting tables*, and identification column. The grouping of the data into several sections presents an additional level, which the user has to access, but offers, as a trade-off, a more structured data display.

During the development of the framework, the users, for whom this framework was primarily intended, actively participated by providing the feedback when using the framework and the created web-interface. This led to the framework, and consecutively to the web-interfaces, that is created according to their needs and wishes, helping immensely in facilitating the acceptance of this

framework and positive user responses. The framework was developed and put in the use to acquire the data in the scope of the study at the University Medical Center Ljubljana, Department of Obstetrics and Gynecology. The main contribution of the developed framework is its ability to assist in electronic data acquisition in different fields. The changes in the user performance when acquiring the data using this framework stem mainly from avoiding the need to transform the data acquired in a paper form into an electronic format, and also from data validation, which provides an immediate correction of erroneous data.

ACKNOWLEDGMENTS

This work was financed by the Slovenian Research Agency (ARRS) which provided a research project P3-0124 - Metabolic and inborn factors of reproductive health, birth, II

REFERENCES

[1] L. C. Chivers. Electronic document supply: experience at the British Library. Interlending & Document Supply. vol. 28, pp. 27-37, 2000.
[2] M. Ramm, K. Dangoor, G. Sayfan. Rapid Web applications with TurboGears: using Python to create Ajax-powerd sites. Upper Saddle River, NJ: Prentice Hall cop, 2007.
[3] S. Ruby, D. Thomas, D. Heinemeier Hanson. Agile Web Development with Rails 4. Pragmatic Bookshelf, 2013.
[4] M. Lutz. Programming Python (3rd ed). Sebastopol, CA: O'Reilly Media, 2006.
[5] S. Hellegourach. CherryPy essentials: rapid Python web application development. Birmingham: Packt Publishing, 2007.
[6] E. Sciore. Database design and implementation. Hoboken, NJ: John Wiley & Sons, Inc, 2008.
[7] M. McLean,W. A. W. Walters, R. Smith. Prediction and early diagnosis of preterm labor: a critical review. Obstetrical & Gynecological Survey. vol. 48, pp. 209-225, 2003.
[8] W. M. Callaghan, M. F. MacDorman, S. A. Rasmussen, et al. The contribution of preterm birth to infant mortality rates in the United States. Pediatrics. vol. 118, pp. 1566-1573, 2006.
[9] D. J. Murphy. Epidemiology and environmental factors in preterm labour. Best Practice & Research Clinical Obstetrics & Gynaecology. vol. 21, pp. 773-789, 2007.
[10] I. Verdenik. Multilayer prediction model for preterm delivery.PhD Thesis. University of Ljubljana, Medical faculty, Ljubljana, 2002.
[11] J. D. Iams. Prediction and early detection of preterm labor. Obstetrics & Gynecolgy. vol. 101, pp. 402-412, 2003.
[12] S. T. Meekai, Z. Alfirovic,V. C. F. Heath, S. Cicero, et al. Cervical cerclage for prevention of preterm delivery in women with short cervix: randomised controlled trial. Lancet. vol 363, pp. 1849-1853, 2004.

Aleš Smrdel received his M.Sc. and Ph.D. degrees in computer and information science from the University of Ljubljana in 2000, and 2004, respectively. He is assistant professor at the Faculty of Computer and Information Science at the same university. His main interests are web technologies, web development, user interfaces, and human-computer interaction.