Towards a Form Based Dynamic Database Schema Creation and Modification System

Kunal Malhotra[1], Shibani Medhekar[1], Shamkant B. Navathe[1], and M.D. David Laborde[2]

[1] College of Computing, Georgia Institute of Technology, Atlanta, Georgia, USA
{kmalhotra7,smedhekar3}@gatech.edu, sham@cc.gatech.edu
[2] Iconic Data Inc, Atlanta, Georgia, USA
dlaborde@aya.yale.edu

Abstract. The traditional approach to relational database design starts with the conceptual design of an application based schema in a model like the Entity-relationship model, then mapping that to a logical design and eventually representing it as a set of related normalized tables. The project we present has been motivated by needs of healthcare-IT where small group practices are currently in need of systems that will cater to their dynamic requirements without depending on EMR (Electronic Medical Record) systems. It is also relevant for researchers for mining huge repositories of data such as social networks, etc. and create extracts of data on the fly for data analytics. Based on user characteristics and needs, the data is likely to vary and hence, a dynamic back-end database must be created. This paper addresses a form-based approach to schema creation and modification.

Keywords: Dynamic user interface, Schema modification, Dynamic form generator, Schema evolution.

1 Introduction

It has been observed that a database schema frequently experiences a lot of changes with time [13]. In most of the domains, there is a lack of a set of common data elements, that users would like to store in the database thus requiring different database schemas. In such cases, a static database schema would pose a problem. One approach to modifying the schema would be to have a database administrator track the data elements and periodically modify the schema. This would require a lot of manual labor and periodic updates would eventually delay data entry. In most large organizations, the DBA staff has been entrusted with dealing with schema management however the costs associated with database management systems and other large-scale application systems such as EMR (Electronic Medical Record) systems tend to be prohibitive for most "small-business", or start-up operations. For most small outfits dealing with a specialty practice with a handful of physicians, adopting large generic systems is prohibitive in adoption, training and maintenance costs. We propose an approach in this paper to:

M. Jarke et al. (Eds.): CAiSE 2014, LNCS 8484, pp. 595–609, 2014.

1. Create schemas based on predefined forms
2. Update and customize schemas as per changing user needs
3. Align the back-end storage of data as the schemas evolve

Our primary goal is to reduce user intervention and to let the database "evolve" consistently as time progresses. For developing a generically applicable system, we base it on the relational model. Our approach to dynamic schema creation and management has been described here in the context of healthcare just for illustrative purposes. This project was motivated by our interaction with a local neurosurgery practice through Dr.Laborde, a neurosurgeon, who convinced us that there is merit to developing approached to "ad-hoc" database creation and management for applications where elaborate and costly solutions like EMR and EHR (Electronic Health Record) systems are an overkill.

We made some assumptions while developing the prototype. Our current implementation has been designed to cater to clinical researchers and physicians who want to use existing data for clinical trials and studies and want to add some parameters of their own. Very little knowledge about database modeling and query languages is assumed. The forms could then be made available to the end users such as patients. The users (physicians, nurses, etc. but not patients) are provided with predefined forms developed based on the underlying database schema. These forms are automatically generated based on the metadata, which is stored in a separate database. This process is discussed later in the paper.

The users can customize these existing forms by adding and deleting data elements of their choice. As a result of this the tables in underlying database will undergo appropriate schema modifications as discussed later in the paper. In our test implementation, we have used a neurosurgery application database from a local clinic called the ALIF (Anterior Lumbar Inter-body fusion) database [10]. We dealt with the ALIF data with Dr.Laborde's expertise as a domain expert in the specific specialty which deals with surgical procedures of the spine. The term "user" will apply to physicians , nurses, researchers etc. who are knowledgeable about the application domain, who can evaluate the suitability of existing forms and who can be guided in their choices when they undertake to modify the forms. Totally naive end users will not be a target audience for our approach.

2 Technical Challenges and Claims

One of the major challenges we faced while developing this system was to avoid anomalies or inconsistencies at the back-end when the user makes changes to the forms provided for entering the data. The traditional approach to constructing a relational database application involves building a conceptual schema of the relational database using a model like the extended entity-relationship model and then constructing a relational database schema [12]. For most advanced applications the schema of the relational database changes during development. We are interested in providing a solution to environments where the database schema needs to be adjusted in real time keeping all constraints and rules of a relational database intact while the user makes changes to the existing forms or creates

new forms. This involves maintaining a metadata database to store metadata about the forms (FORMS DATABASE) and a domain specific database (DSD) to store the actual data entered by the user. The system also provides an easy to use Form Field Selection feature, which can be used by novice users to build their own forms.

A dialog based UI is designed to create a new database from scratch or modify an existing one. It would gather information about the nature of the form field (label in the form) being added which would in turn lead to appropriate changes in the schema of the DSD. Addition of a form field leads to formation of a new attribute in the appropriate table in the DSD. For the data which already exists in the database before adding a new field, the system populates default values for that particular field. Deletion of the form field does not lead to deletion of the corresponding attribute from the schema. These deletions are tracked and recorded in a metadatabase, and a customized form is presented to the user. Modification of a form field does not modify the field name at the back-end. Mappings are created between the field at the back-end and the ones in the UI. A new user may choose from any of the existing customized versions of a form and select the most appropriate one or he (henceforth we will use the pronoun "he" for the user without any intended bias) may customize them further based on his requirements. In order to ensure a smooth working of the system after plugging in any DSD which stores the data entered via the forms, the FORMS DATABASE stores the metadata of all forms and also the information about what tables a form is connected to.

A user is typically shown all available forms, which guide the user to use one that comes close to their requirements and to modify it. They are also given the option to create a form from scratch. Our algorithms for storing metadata, for defining the schema for the back-end, for storing actual data as well as displaying user defined forms are generic. We propose the mechanism by which a metadata layer called the FORMS meta-database is created to accommodate the current form definitions and subsequent changes to it. In this approach, a user can choose to display data elements that he would like to view together without providing an SQL query. At the back-end the algorithm performs joins between tables based on primary key-foreign key relationships and displays data elements requested by the user. The user can also request the system to perform aggregation operations on data elements, add conditions as well as sort the results. This is particularly geared for researchers who would want to do studies or clinical trials using patient treatment data related to a specific drug as the DSD. They would then create a new database after appropriate aggregation to define a cohort of patients.

3 Approach to Dynamic User Interface Management

The user interface of the system is a form-based UI since its functionality is guided by a set of forms. The user is provided with two modes; one of them deals with 'Data Entry' and the other with 'Data Retrieval'. Data entry can be

done by either using the existing template forms provided by default based on the current state of the DSD, or the existing forms may be customized to suit new requirements before being used to insert data. The users can also create new forms if the requirements for the data they want to enter are not met. These changes are automatically translated into appropriate modifications at the schema level.

A large set of options is provided to the user as to what type of form field he wants to create for his form. The options are Radio buttons, Checkboxes, textboxes, dropdowns and buttons. Any of these options apply while modifying existing forms or creating new forms. On creating a new form field, the user is required to provide details about its data type, default value, ability to have multiple values, etc. helping the system to make appropriate modifications in the database. Similarly, creation of a new form requires some information from the user which helps in creating a new table at the back-end. E.g. information regarding its relationship with the existing forms, cardinality of the relationship, etc. The UI has help buttons to describe the semantics of these questions in simple language with examples.There may be a class of users such as researchers that may use an existing large database, create a form using our tool and then modify the form slightly to add some fields/attributes of their own using the data entry function.

The other important feature of this system is 'Data Retrieval' which is used for retrieving data by generating queries based on user input. The operations currently handled by the system are 'Select', 'Aggregation', 'Group By', and 'Having'. Appropriate selections may be made by the user and on doing so the system builds the query and displays the results after validation. This feature would be explained later in section 5. Figure 1 illustrates the process flow adopted by the system.

3.1 Data Entry

The user is presented with template forms based on the underlying DSD. He has a choice of either using the available forms to enter data or customize the forms appropriately. In the latter case he uses the Form Field Panel which displays the potential types of form fields available. A dialog is initiated with the user, which requires him to provide details about the form field chosen. For example, if a radio button is selected for a field 'Gender', then he is required to enter the number of options he wants to keep for this field and their corresponding labels. The new form field added to a form would become a new attribute in the corresponding table having the rest of the form fields of that form after the user provides information about this new attribute using an 'Add New Form Field' screen. This information consists of the label of the form field, it's data type, e.g Integer, String, etc. The user would also need to specify a default value of the field which would be stored if the field is left blank while entering data. This new field may have multiple values e.g If the new field which is getting added is 'Symptoms' then it may contain multiple values since a patient may have multiple symptoms. This can be specified in the 'Add New Form Field' screen.

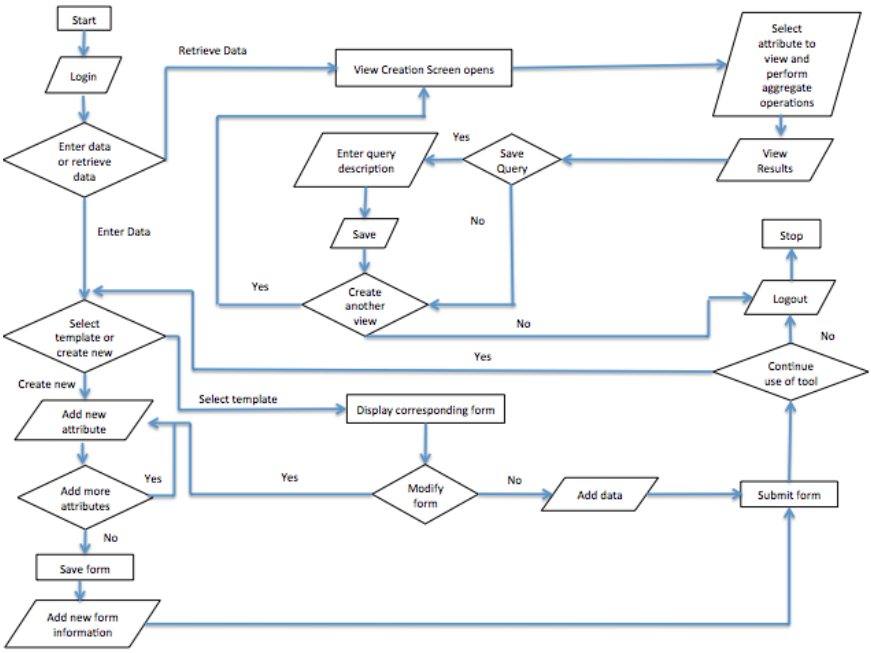

Fig. 1. Process flow of the system

Each modified template form would be stored as a new version of the existing template and will be annotated with the username of the user modifying it. E.g 'Form_A_User1' is 'Form_A' modified by 'User1' and 'Form_A_User2' is 'Form_A' modified by User2. All the versions of a template are shown to every user from which he can select one for data entry.

The user may create a new form from scratch for which he would be required to provide some information about the new type of data he would like to store (See Figure 2). As mentioned before the '?' marks alongside each label would assist the user in filling this form. The 'Form Name' is the name of the new form the user wants to create. The 'Unique Form Field' would be the field which uniquely identifies an observation. This is equivalent to the key for that form. Initially this field would be empty since it is a new form and no fields have been added. The user would use the 'Add New Form Field' screen to add fields to this form. The system allows selection of multiple fields for the cases when a combination of more than one fields uniquely identifies an observation. The 'Related to Form' field would require the user to choose from a list of existing forms to which this form is related to. As shown in the figure 'Patient' form has a relationship with 'Visits' form in the context of patients having visits in a hospital. The 'Cardinality' can have values 1:1, 1:N and M:N according to the standard cardinality ratio concept in database modeling [3]. Based on the nature of the relationship between the forms the user needs to select the cardinality.

In the example shown in figure 2, the selected cardinality 'M:N' denotes that a single patient may undergo multiple procedures and a procedure may be done on multiple patients. The label 'Is there any Attribute for the Relationship' requires the user to specify attributes which are descriptive of the relationship. E.g. 'No. of Hours' are the the number of hours that the patient undergoes a particular procedure. It is neither an attribute of the 'Patient' since a patient may undergo multiple procedures, each for a different amount of time nor of the 'Procedure' since each procedure may be done on different patients, each time for a different number of hours. It is describing the instance of the relationship 'undergoes' between 'Patient' and 'Procedure'. Multiple attributes are allowed via the drop-down menu. Based on this information the schema of the DSD is modified by adding a new table at the back-end and relating it to appropriate tables based on the rules discussed in [3,7]. Appropriate changes are made in the FORMS DATABASE simultaneously. These rules help creating a new table and relate it to appropriate tables in the database maintaining consistency.

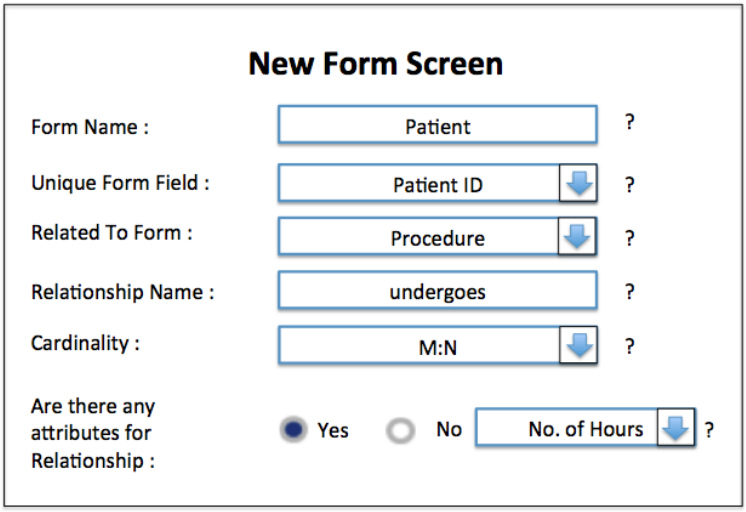

Fig. 2. New Form Screen

4 Approach to Dynamic Schema Management and Maintenance

Our system has two databases, one is the DSD, which primarily stores the actual data, that is entered by a user using the forms provided, and the other one is the FORMS DATABASE that stores the metadata about the forms. The FORMS DATABASE is the one primarily responsible for the dynamics of the system and is explained in more detail in the following section.

4.1 Meta-Database Schema

This database (see Figure 3) consists of all the information about the form structure such as the form name, the fields it consists of, the label of each form field, the type of form fields, etc. needed to build a form. The arrows in the figure stand for foreign-key to primary-key referential integrity constraints. In our present implementation the users are provided with already existing form templates pertaining to the data for a local neurosurgery practice. The database we used for our test was the already pre-populated ALIF database with information about the template forms.

The table 'Form' records a list of all forms provided by the system by default in a well-annotated format. Any new form created by the user gets added to the list. The modified form gets stored as a new form and there is a record of which form it originated from. This table is used to pull out all the forms existing in the current state of the database for the user to choose from to fill data or to select data elements to aggregate data as explained before. The table 'Form Field' stores information about the different fields present in the form along with the forms that they are a part of. It also stores the label of the form fields and a detailed explanation of the field which would translate into a tool tip description in the UI to assist the user in filling the data. The table 'Form Field Type' is created where values such as textbox, radio-button, check-box, drop-down,etc are stored. The table 'Form Modifications' is needed to keep a record of modifications made to the existing form. If a form field is added or deleted by a user to create his customized form then this table will keep track of the changes and pull out the customized form fields for users. The 'Form Field Option' table stores the enumerations of values for the aforementioned types of form fields. For instance, the form field 'Gender' has options 'Male' and 'Female' which are stored in the 'Form Field Option' table.

The table 'Table Information' stores the tables, which are present in the actual database holding the data, which in our case is the ALIF DATABASE. This table is required for the purpose of aggregating data using the view creation mode where users have an option of aggregating data elements, which they want to view together. The table 'Attribute Information' stores all the attributes present along with their data-types and default values in the actual database holding the data. We also keep track of attributes which are primary keys or foreign keys in a particular table. This enables the system to decide the table joins when user selects data elements to be aggregated in the view creation mode explained in the following sections. If a user selects some data elements from a set of tables, which cannot be joined due to the absence of Primary Key-Foreign Key relationship then the user is prompted against the action

This FORMS DATABASE is populated by acquiring the definition of the underlying DSD, say as an SQL file, with CREATE TABLE statements. The level of automation is being improved by providing this functionality. The system currently has the ability to filter out attributes which need not be displayed on the forms. Our strategy for dealing with schema evolution as the database creation progresses is explained below. The algorithm can be used to modify an

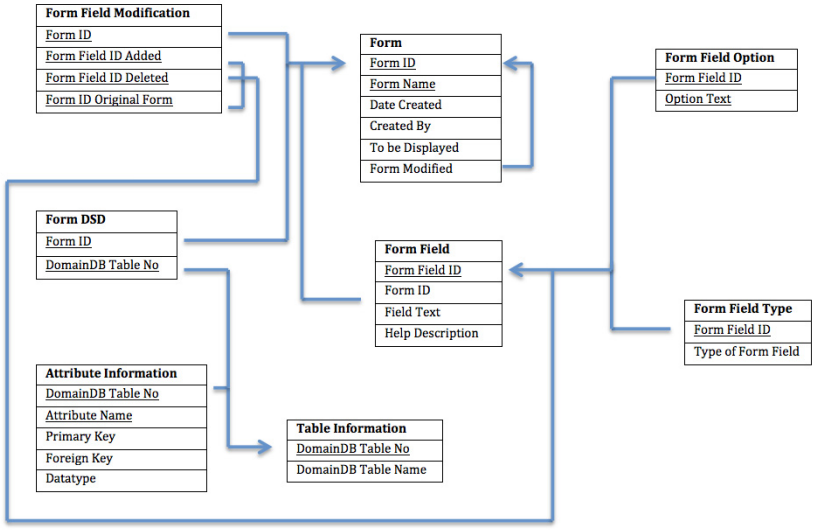

Fig. 3. Schema of the metadata database. The arrows show referential integrity constraints.

existing database at the back-end with the help of a dialog based UI or create a new database from scratch. When modifying an existing database by adding additional forms we already have the metadata database and the DSD at the back-end contrary to the case when a new database is created. In the latter case, an unpopulated metadata database exists at the back-end. As and when the user creates forms at the front-end, the DSD gets developed. We explain our approach for the 2 cases below:

Case 1: Database Schema Modification of the DSD
This is the case when a user chooses to modify existing forms to customize based on his requirements in turn leading to modification of the DSD schema. The modified form would be treated as a new form with a new 'Form ID'. The creation of this new form is recorded in the 'Form' table. In this table the 'Form ID' of the original form is stored, which was modified to create the new form. The modifications would be stored separately in another table 'Form Field Modification' where we store the form fields that were added or deleted from a particular form to create a new form. Any form field which gets added to a form first needs to be added to the 'Form Field' table, 'Form Field Type' table and the 'Form Field Option' table appropriately. For every new form field which gets added to a form, a corresponding attribute gets added to the existing table which has other attributes corresponding to the other form fields of the form. This requires population of the 'Attribute Information' table with all the information about the new attributes. The 'data entry' feature is used to populate the table

with data. If there is data already existing in the table corresponding to the form being modified then default values of the newly added attributes would be inserted for these existing observations.

Case 2: Database Schema Creation
This is the case when a user does not want to use any of the existing templates and instead creates a new set of forms. This may occur if the data that the user wants to store pertains to a different domain. He would be required to create new forms from scratch which would guide the creation of a new domain specific database. At the metadata level this involves populating the 'Form' table with information about the new forms. The 'form modified' field would be 'NULL' since we are not modifying any existing forms to create the new forms. In this case we would also populate the 'Table Information' table with information about the new tables that would be formed in the new DSD corresponding to the new forms created. Subsequently the 'Form DSD' table and the 'Attribute Information' table would also be populated as and when new form fields are added in the forms. The process of creation of new form fields has been explained in CASE 1.

In both cases, when adding a form field to a form if the form field is supposed to have atomic values then the corresponding table in the DSD is updated but if the form field is expected to have multiple values then a new table is created in the DSD which references the original table corresponding to that form.

4.2 Guranteeing Consistencies during Schema Evolution

In our approach a lot of flexibility has been provided to the user in terms of freedom of choice of creating new forms when the existing form templates do not seem to be suitable. This can lead to a lot of redundancy at the back-end since a user may choose to build a new form instead of modifying existing templates even though his requirements differ from the existing forms by a small amount. For example, a user needs only 8 out of 10 form fields of a particular form and wants to add more fields of his own choice. We would assume that the user would use our feature of modifying this form by deleting the two irrelevant fields and adding the new extra fields. But instead it is possible that he creates a new form and adds all the form fields he needs to this new form. At the back-end this would result in an extra table in the DSD which would store data being entered via this new form. Periodic reorganization of the DSD and the metadata database is required. This would involve manually identifying such redundant tables and integrating them into one table by performing a full outer join between them. This would be done when the primary keys of the two tables which are getting integrated are the same. The two keys may have different labels but if they have the same semantics, we would go ahead with the join. Such merging would be done only with human approval. If the keys are different then we would keep the tables as they are. A full outer join may result in a lot of null values in the integrated tables. Due to the increase in the volume of data and creation of multiple redundant tables, the decision to go ahead with the integration would

depend on the relative advantage of querying a single table with a lot of null values over querying multiple tables to get the data.

5 Data Retrieval: Query Creation and Validation

The system also supports data retrieval by giving the user a choice of either selecting existing result sets previously created by users or creating his own. The user is presented with four types of operations namely 'Select', 'Aggregation', 'Group By' and 'Having' to build a query.

1. **Select Operation**
   The user can select fields, which need to be displayed together in the result set. This may be done by selecting a table from a drop down list and on doing so the corresponding attributes of the selected table would be shown.
2. **Aggregation Operation**
   The operation may be used by user to perform aggregate functions like COUNT, MIN, MAX, AVG, etc on the fields.
3. **Group By and Having Operation**
   If the user decides to use the Aggregation operation then the 'Group By' and 'Having' operations would be enabled. Using them the aggregated results may be grouped with respect to certain fields which may or may not be based on some condition.

The system maintains consistency of the user request as well as of the back-end operations as follows. If the selected attributes belong to multiple tables then the system would perform a join between those tables using the Primary Key and Foreign Key constraints. A 'Where' clause is appended to the query being created, to reflect the join. If there are any attributes that the user has selected to group his result by then those attributes would be added to the attribute selection list already created by the 'Select' Operation in the first step. If the attributes that the user has selected require a join of tables, which cannot be joined due to absence of a primary key-foreign key relationship, then the user would be prompted to change the selection. If the user has selected some attributes and is also performing aggregation then the system would remove the selected attributes other than the ones he is grouping by from the SELECT clause. The user would be prompted of this action. The system would also check for validity of attributes selected for aggregation. E.g. Calculating 'Average' of a non numeric attribute is prevented. After validation the query is executed and the result is displayed. The user has the choice of saving the result set with a description of the same. At the back-end the query is stored in the DSD along with the description provided and can be executed again when selected by its query label.

6 Related Work

Some research has been done in the area of dynamic schema modification with respect to a clinical dental relational database [12]. Their approach primarily

focuses on handling One-to-Many and Many-to-Many relationships between tables via concepts of 'detail' and 'link' tables. The user interface developed by the authors is limited to the dental domain. In our approach the metadata database remains unchanged and there exists a provision for plugging in any DSD which in turn can be modified by users. Our application can also be used to create a new database from scratch and store data while it is created in real time. In addition to this, in [12] the interface to manage the addition of datasets requires the user to be familiar with the dataset being loaded after which it is the responsibility of the user to map it to an existing domain or create a new domain. In our approach the user is oblivious about the back-end structure. While he creates new form fields or a whole new form, our application automatically begins to modify the existing schema by creating new attributes or relations respectively to accommodate the new data elements. In addition to the dynamic schema modification approach, a feature of aggregating data and presenting the results to the user, which he can store for future use is also supported by our system. Palisser et al [1] discuss drawbacks of the systems called Orion [5] and Encore [11]. The former constructs a version of the database state every time any transformation in the schema takes place. This leads to the problem of managing multiple versions. The latter focuses on versioning of object types when design environment object types change in an object oriented database. In our approach on the other hand the original schema is modified based on the changes requested by the user but the user is kept unaware of these changes. For instance, a user might modify an existing form to create a new customized version but at the back-end the original schema accommodates the new data elements in an appropriate manner to avoid resorting to versioning. Kim et al. [6] have handled versioning of object types as well as schemas for single as well as multi-user design environment in Orion and also provided semantics of versioning the schema. Ferran et al. [4] discuss an approach to handle schema and database evolution in O2 object database system. The algorithm proposed by the authors automatically makes the database consistent on any update operation performed on the schema. However, depending on what the updates are, either immediate or deferred transformations are made. Deferred transformations cause problems while implementing complex conversion functions,that the user needs to specify if the default functions do not suit their needs. Our approach on the other hand does real time modification of the original schema based on the changes requested by the user and avoids user intervention to a large extent.

7 Use Case

Let us describe the current interface for this prototype used by ALIF practitioners at a local neurosurgery practice.

1. **Basic menu for user [Figure 4]:** This figure shows the menu which will be presented to the user after logging in. It consists of all the template forms consisting of data elements currently in the back-end database. The last bullet is "Create New Form" which would be explained ahead. It has a

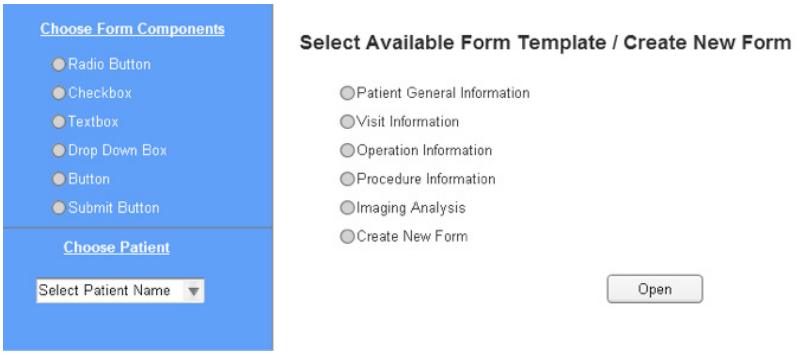

Fig. 4. Basic Menu

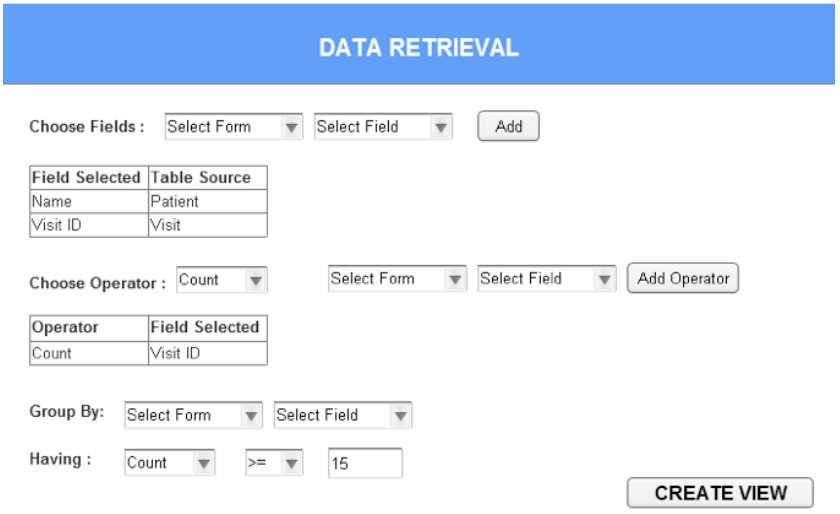

Fig. 5. View Creation Screen

'Form Field Selection' panel which the user can use when creating new form fields when needed. The main menu also consists of a 'Choose Patient' option which will help pulling up existing patients in the database when additional data needs to be entered about them.

2. **Ability to use existing forms :** On choosing a particular template the corresponding form is displayed for the user to fill in data. This data gets saved in the appropriate tables at the back-end.
3. **Creating / Updating forms :** If the user feels the need to update existing templates to accommodate data elements, which are not in the form, he can modify the form using the 'Add New Form Field' feature. This creates a new version of the existing form incorporating the changes requested by

the user. The new form fields added get saved in the back-end database at appropriate places. The "Create New Form" feature can be selected from the screen shown in Fig. 4 to create a new form from scratch and add new form fields.

4. **View Creation [Figure 6]:** The data retrieval feature allows users to select the fields they wish to integrate together, perform aggregation operations on them and also group the result set fields of their choice. The result set can be saved for future use. All screens could not be shown for space reasons.

8 Discussion and Conclusion

8.1 Why not NoSQL?

A lot of organizations which collect vast amounts of customer, scientific, sales data have traditionally stored data in a relational structure, but recently some of these organizations are tending to use various types of non-relational databases called NoSQL databases since they have been found to be efficient in handling unstructured data where there is no fixed schema [8]. In our case we also have a schema that is flexible since the user can modify it based on his requirement, but we chose not to go for NoSQL due to the following reasons.

1. Consistency, availability, and partition tolerance are the three properties taken into consideration when designing a database system. According to the CAP theorem [2] it is only possible to have two out of these three properties together in a database system at once. Traditional relational databases maintain consistency and availability but have trouble with partitions whereas NoSQL databases are able to maintain either consistency or availability along with partition tolerance. In our case the main goal of the system is providing the user with all the data in a consistent state and available in user's expected form.
2. The primary goal of our system is to add flexibility to existing schemas and dynamically modify them. Since most of the healthcare databases have a relational structure at the back-end, we formulated our approach using relational databases.

8.2 Metadatabase Flexibility

The metadata database in our system plays an important role in driving the dynamic nature of the DSD which stores tha actual content entered using the forms. The schema of the metadata database is designed in such a way that the DSD can be modified without any manual interference. It guides the UI formation and the selection process of the different UI components other than providing the forms with a particular structure under different circumstances. The metadata also has the capability to form queries based on information entered by the users along with verifying the correctness of those queries.

8.3 Summary and Future Work

In this paper we proposed a UI and metadata based approach to dynamically modify and create a database schema which would address the problem of dynamic data entry in data-rich environments where the schema can be "built" incrementally as new data becomes available. It is intended for users with needs for managing data for operational and research purposes but who have no access to DBAs or database design experts. Template forms are provided to the user but since there may be difference in requirements between users, the system facilitates modification of existing forms or create new forms from scratch resulting in appropriate real time modifications in the database schema keeping the user oblivious to the back-end The changes at the back-end involve adding new attributes to existing tables, adding new tables, creating relationships between these new tables and existing tables by assigning appropriate cardinality, etc. Unlike other systems such as Encore and Orion, our approach does not create versions of the schema whenever there is a change; instead it creates different versions of a single form if needed after modifying the schema appropriately. Our system also has a data retrieval feature which helps users to aggregate and extract data from the database, perform aggregate operations on them and storing the result sets for future use. This feature does not require the user to write queries instead it automatically generates and validates queries based on the data elements selected.

There is still a lot of scope for improvement in this system like making the system usable for novice users who do not have any knowledge of database modeling concepts. Given a good domain ontology about synonyms such as WordNet [9] we would like to automate the process of removing redundant tables. There is a possibility that a user modifies a form to add a form field, which has semantic equivalence with one of the existing fields in the form. This would result in redundancy. To avoid such cases we would like to incorporate semantic validation in the future. The paper represents preliminary work awaiting field experimentation with small group practices of physicians. It addresses the problems related to relational schema evolution in real-time which opens up a lot of room for improvement. We have been motivated for the need of such system in data-rich environments with relatively limited volume such as medical practices. We are addressing a large user population where the typical user is not very knowledgeable about database concepts but would like to be able to create robust databases on the fly. We would like to like to address the aforementioned ideas of improvement before making it available to the medical community.

References

1. Andany, J., Leonard, M., Palisser, C.: Management of Schema Evolution. In: VLDB, Barcelona, Spain (1991)
2. Brewer.E.A: Towards robust distributed systems (Invited Talk) Principles of Distributed Computing, Portland, Oregon (July 2000)
3. Elmasri, R., Navathe, S.: Fundamentals of Database Systems, 6th edn. Addison Wesley (2011)

4. Ferrandina, F., Ferran, G.: Schema and Database Evolution in the O2 Object Database System. In: VLDB, Zurich, Switzerland (1995)
5. Kim, W., Ballou, N., Chou, H.T., Garza, J.F., Woelk, D.: Features of the ORION Object-Oriented Database System. In: Kim, W., Lochovsky, F.M. (eds.) Object-Oriented Concepts, Databases and Applications. ACM Press Frontier Series, New York (1989)
6. Kim, W., Chou, H.: Versions of schema for object oriented databases. In: VLDB, Los Angeles (1988)
7. Kolp, M., Zimanyi, E.: Enhanced ER to relational mapping and interrelational normalization. Informaton and Software Technology 42, 1057–1073 (2000)
8. Leavitt, N.: Will NoSQL Databases Live Up to Their Promise? Technology News (February 2010), `http://www.leavcom.com/pdf/NoSQL.pdf` (retrieved)
9. Miller, G.A.: WordNet: A Lexical Database for English. Communications of the ACM 38(11), 39–41 (1995)
10. Sasso, R.C., Reilly, T.M.: Anterior Lumbar Interbody Fusion: Threded Bone Dowels Versus Titanium Cages. In: Resnick, D.K., Haid Jr., R.W., Wang, J.C. (eds.) Surgical Management of Low Back Pain, 2nd edn. American Association of Neurosurgeons, Rolling Meadows (2008)
11. Skarra, A., Zdonik, S.B.: The Management of Changing Types in an Object-Oriented Database? In: OOPSLA, pp. 483–495 (1986)
12. Taylor, D.,Naguib, R. N. G., Boulton, S.: A Dynamic Clinical Dental Relational Database. IEEE Transactions on Information Technology in Biomedicine 8(3) (September 2004)
13. Zhou, L., Rundensteiner, E.A.,Shin, K.G: Schema Evolution for Real-Time Object-Oriented Databases IEEE TKDE 9(6) (November 1997)