Metadata Enrichment for Automatic Data Entry Based on Relational Data Models

Meysam Dolatkia1, Sahar Adabi2, Ali Rezaee3
1,3 Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran.
2 Department of Computer Engineering, North Tehran Branch, Islamic Azad University, Tehran, Iran.
1(mdolatkia@gmail.com)
Received (2017-12-02) Accepted (2018-09-19)

Abstract: The idea of automatic generation of data entry forms based on relational data models is a common and known idea that has been discussed day by day more than before according to the popularity of agile methods as well as improvement of programming tools in software development. One of the requirements of automation methods, whether in commercial products or relevant research projects, could be the concept of metadata as a mediator between database and data entry forms. The metadata usually includes some schemas and constraints of target database, which could be used as a model for automatic generation of data entry forms. However, the most metadata models proposed in relevant researches have simple and undetailed structure. In other words, only the initial requirements of data entry are covered in the proposed metadata. In this study, the main objective is to emphasize the structure of metadata to discuss its enrichment methods to cover more data entry requirements. In this regard, some parts of a metadata model are also presented for objectification of the ideas.

Keywords: Data-Entry automation, Data-Entry requirements, Relational database metadata

I. INTRODUCTION

Over the years, automation of data entry user interfaces based on data models has been one of the methods of implementing data entry requirements in software programs. The main purpose of automation methods is to accelerate the implementation process through prevention of taking repetitive works. In practice, this is done through reuse of designs and logics leading to database data entry. The main requirements include UI design with the capability of CRUD operations on the entities, validation of input values and implementation of data transactions queries in consistence with a target database. In order to implement the logics, the usual solution is using metadata as a guideline model which some parts of it could be perceived through data dictionary resides in relational database management system. However, by considering metadata repository in a DBMS as the only source, the function of designed data entry forms would be very simple and rigidly depended on the physical model of the database. Hence, the possibility of defining lots of high level concepts or design details would be lost. On the contrary, through enrichment of the metadata, a wider range of data entry requirements could be covered.

In this study, has been tried to provide some ideas about the methods of formation, maintenance and enrichment of the metadata devoted to data entry requirements. This is
done through classification of requirements and discussing methods and examples in response to these requirements. In this regard, it has also been tried to decline the dependence of metadata on implementation technologies of application level to develop the use of metadata to more applications and platforms.

II. RELATED WORKS

As this study has emphasized the content of metadata, to review the relevant works, those works are selected that have used metadata as a part of solution to implement data entry forms. In ref [1], the solution for generation of web data entry forms using extracted metadata from database is analyzed; although no specific structure and format is specified to maintain the metadata. In ref [2] with almost similar solution with [1], the idea of storage of metadata in XML format is discussed. The metadata used in these solutions is too simple to generate data entry forms and includes elementary data from independent structure of each table and its columns. For example, in the proposed metadata, the details of relationships, types of relationships and simple requirements as the possibility of defining value set for a column are not addressed. In a different work in ref [3], the same idea of generation of web forms using metadata is presented. Although this metadata is presented in form of an Entity-Relationship model, in addition to simplicity of concepts, it is significantly depended on technical requirements of application (in this case HTML pages).

In general, such applied attitude to metadata to discriminate the ideas and technologies could be observed in other works such as [4], [5] and [6]. In other words, in most researches, the content of metadata is not exclusively considered and emphasis has been on the applied role of metadata to discuss the solutions.

Clearly, with simple or application-depended metadata, there would be no chance to cover a broad range of data entry requirements. Therefore, it would be better to separate the concerns of metadata generation and the way of using it through separating the declarative independence of metadata from decision making processes consuming the metadata as much as possible.

III. A SAMPLE DATA MODEL FOR DATA ENTRY

In this section, for a better understanding of the ideas presented in this paper and also providing better examples, a sample ERD model is presented. This model represents the structure of a sample database which could be the target of data entry forms. It should be noted that this model is not necessarily a reality-based model, but simply covers various types of data entry requirements and concepts. This model keeps track of a company’s employees, departments and projects. The model is divided into two parts presented in Fig.1 and Fig.2. In Fig.1, the main entities of Employee and Department and their relationships are presented. Each employee has a many-to-one relationship with the Department table to determine the related working department. Each department has also a one-to-one relationship with the Employee table that determines the manager of the department. Also, through an IS-A relationship, each employee can be specialized into Engineer and Technician entities. Finally in Fig.1, the ParkingOwner entity is shown. This entity represents the owners of parking lots in a company which has a union relationship with department and employee. In other words, each parking lot could be owned by either a department or an employee.

In Fig.2, four new tables have been added to the previous model. First, the Project table that represents the projects of a department and has a many-to-one relationship with Department table.
Also, this table uses the Employee_Project table to establish a many-to-many relationship with the Employee table. In this way, for each department’s project the employees involved in the project are identified. Finally, the Location table is presented which specifies the address of each department and also has a many-to-one relationship with the City table.

**Fig.2 Sample database ERD model (Second part)**

**IV. EXTRACTION OF THE INITIAL ELEMENTS OF METADATA**

In order to form the initial elements of metadata and as the beginning point, the data dictionary of target database residing in DBMS could be used. One way to extract the existing metadata is to query the data dictionary using the known standards. However, the ways of using these standards may be different in each RDBMS. For example, in relational databases of SQL Server and MySQL, the concept of information schema is supported; although different concepts are used in almost similar methods in Oracle database. In table 1, the way of extraction of tables, columns and the referential constraints in SQL Server and MySQL are presented. Similar queries of the same objects in Oracle are presented in table 2. In ref [6], [7] and [8], technical documents about the metadata management of mentioned relational databases are presented.

**Table 1 Using the concept of information schema in fetching database objects**

<table>
<thead>
<tr>
<th>Extraction of</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>Select * from Information_schema.tables</td>
</tr>
<tr>
<td>Columns</td>
<td>Select * Information_schema.columns</td>
</tr>
<tr>
<td>Referential Constraints</td>
<td>Select * from Information_schema.referential_constraints</td>
</tr>
</tbody>
</table>

**Table 2 Fetching the same database objects in Oracle**

<table>
<thead>
<tr>
<th>Extraction of</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>Select * from user_tables</td>
</tr>
<tr>
<td>Columns</td>
<td>select * from user_tab_columns</td>
</tr>
<tr>
<td>Referential Constraints</td>
<td>Select * from user_constraints</td>
</tr>
</tbody>
</table>

After extracting the desired data from the database, a structure is needed to maintain the metadata. In this solution, a relational model is proposed. The advantages of providing relational model for the metadata are as follows:

- It is in consistence with the target data entry relational database.
- It could be visualized through entity-relationship models.
- If needed it could be converted simply to other formats like XML using common available tools.
- Using today’s ORM technologies, the metadata could be used effectively. In this way, in addition to separate the technical concerns of metadata manipulation, the key requirement of metadata survey (through entities and relationships) could be done with more facility.

Finally, the initial extracted objects like tables, columns and relationships could be illustrated in an ERD model as figure 3.

**Fig.3 Core structure of the metadata**
 Obviously, this entity-relationship metadata model or the sample data entry model presented in previous section can be implemented in any commercial relational database. By implementation of these two models in SQL Server and using the commands like Table 1 to reproduce the sample database model in form of proposed metadata model, the data of three tables of Table, Column and Relationship would be similar to figures 4, 5, and 6.

V. METADATA ENRICHMENT

In this section, the methods of extending the core metadata are discussed. With more coverage of data entry requirements, a metadata could be enhanced from a repository of some physical or structural properties to a source containing higher level conceptual or behavioral concepts. The practical method used to enhance the structure of metadata is modifying its current structure (the structure extracted from the database in previous section) through adding entities, relations and attributes based on some requirement analysis and inferences. These changes could be the result of considering some general requirements of data entry or some specific strategy of data entry. Firstly, some general requirements (generalizable to other solutions) are discussed.

General and common requirements

When consuming the physical model of a relational database to design data entry forms and logics is discussed, recognized elements such as tables, columns and relationships come to mind. Assessing common requirements of these elements, which can be involved in the designs decision-makings of data entry forms (directly or indirectly), could be a good beginning point. Therefore, for the purpose of better separation of requirements, they could be divided into table requirements, column requirements and relationship requirements.

- Table requirements: for each table, the following types of general requirements could be applied in the metadata:
  - Defining custom alias name for each table;
  - Determining the table data entry mode in group or single form (one by one record);
  - Determining whether a table is a reference table; For example, the City table in the sample model is a reference table.
  - Determining whether a table is an associative table; For example, the Employee_Project table in the sample model is an associative table.
  - Determining table independent data entry: this property specifies that whether a table
can be the beginning point of a data entry scenario or not and it should be accessed just through relationship with other entities. For example, logically the Location table cannot be used as an independent data entry form and is considered as a piece of complementary information along with other entities.

- **Column requirements:** similarly, the following characteristics could be counted for each column:
  - Defining custom name for each column;
  - Determining the data entry ability of column;
  - Determining the search ability of column;
  - Determining the view ability of column in result of search operations.

- **Relationship requirements:** for each relationship, regardless of type of relationship, the following characteristics could be defined:
  - Defining custom name for the relationship;
  - Determining data entry ability of relationship;
  - Determining the searching ability of relationship;
  - Determining the view ability of relationship;
  - The mandatory constraint of relationship;
  - The transferability quality of relationship;
  - Determining the creation ability of other side of the relationship; For example, is it possible to create a new city when entering location information, or the city item is just selectable?
  - Capability of creating the other side of relationship directly in the current form or in a separate form.

The model after making these modification is shown in Fig.7.

### Specialization of elements

In previous section, some simple general requirements were discussed which could be responded through adding some attributes to the initial metadata model. However, a desirable enrichment process of metadata is not limited to adding some properties. In other words, some advanced requirements should be covered through adding new entities and relationships to the model. The first proposed technique to cover more advanced requirements in the model is specialization of the existing elements. Using this technique, in addition to make the model developable to host more complicated requirements, the core model, derived from physical structure of relational databases, could be preserved. In order to specialize the concepts in the model, a few ISA-Relationship types on tables, relationships and columns could be added.

**Specialization of tables**: each table in a database could be resulted by realization of several higher level entities. For example, in implementation of ISA-Relationships in form of accumulation of superclass and subclass in a single table, the table could encompass structure and data from several entities simultaneously. Through specialization of tables by making its columns, relations and even its data selective, chance of introducing new entities in the metadata is provided. For example, in Fig. 8 by adding a new entity called TableDerivedEntity, each previously detected table can be specialized. In other words, columns and relationships of each existing table can be selected and grouped in frame of a new entity. Moreover, using the criteria property in TableDerivedEntity, a distinctive range of data rows for each entity could be defined.
For example, using this method, the department records of Department table can be classified using the Type field. Then it is possible to allow just some specific types of departments to have projects.

Specialization of relationships: through separation of types of relationship in metadata, specific properties could be defined for each type of relationship. For example, in Fig. 10 one-to-many and one-to-one relationships have been specialized through adding new entities inheriting from the base Relationship entity.

Specialization of columns: types of columns could be also specialized based on data type. Hence, not only separated specifications and structures are defined for each data type, but also it is possible to define structure for custom types of data in the metadata.

Higher level relationships

One of the main capabilities of a desirable metadata could be possibility of defining conceptual and higher level relationships of target database like ISA relationships or Union relationships. Without prediction of these relationships in metadata, a big gap remains between metadata model and the object-oriented designs of entities behind the data entry forms. One technique to cover higher-level relationships is defining conceptual relationships in the metadata model and adjusting existing physical relationships with these conceptual relationships. Through classification of physical relationships in form of conceptual relationships, a bridge could be created between physical implementation of relationships and their underlying higher-level concepts. For example, as in Fig. 12, a set of physical relationships of superclass to subclass in database level (defined in SupertoSubRelationshipType) could be mapped to an ISA-Relationship entity in the metadata level. In this way, it is possible to define the known ISA-Relationship properties, such as total participation or disjoint quality of the relationship in the metadata.

In another example, many-to-many relationships could be defined in the metadata through the ManyToManyRelationshipType table. Since all the many-to-one relationships in physical level are converted to one-to-many relationships there would be a link between ManyToManyRelationshipType and OneToManyRelationshipType to detect all the implementing physical one-to-many relationships of each many-to-one relationship.
In this way, all the physical or conceptual relationships defined in the sample model can be defined in the metadata. In tables 3 and table 4 all the relationships of sample model and their definition class in the metadata are shown.

Table 3 Definition of sample model relationships in the metadata (Part 1)

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Metadata Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Department-Project (Works for)</td>
<td>One to Many / Physical</td>
<td>OneToManyRelationship</td>
</tr>
<tr>
<td>City - Location</td>
<td>One to Many / Physical</td>
<td>OneToManyRelationship</td>
</tr>
<tr>
<td>Department-Project</td>
<td>One to Many / Physical</td>
<td>OneToManyRelationship</td>
</tr>
<tr>
<td>Employee-Department (Manages)</td>
<td>One to One / Physical</td>
<td>OneToOneRelationship</td>
</tr>
<tr>
<td>Department - Location</td>
<td>One to One / Physical</td>
<td>OneToOneRelationship</td>
</tr>
</tbody>
</table>

Table 4 Definition of sample model relationships in the metadata (Part 2)

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Type</th>
<th>Metadata Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee-Project</td>
<td>Many to Many / Conceptual</td>
<td>ManyToManyRelationship</td>
</tr>
<tr>
<td>Employee-Technician</td>
<td>One to Many / Physical</td>
<td>OneToManyRelationship</td>
</tr>
<tr>
<td>Project-Employee_Project</td>
<td>One to Many / Physical</td>
<td>OneToManyRelationship</td>
</tr>
<tr>
<td>Employee-Engineer/Technician</td>
<td>IS-A Relationship / Conceptual</td>
<td>ISA relationship</td>
</tr>
<tr>
<td>Employee-Engineer</td>
<td>Superclass to Subclass / Physical</td>
<td>SuperToSub-RelationshipType</td>
</tr>
<tr>
<td>Employee-Technician</td>
<td>Superclass to Subclass / Physical</td>
<td>SuperToSub-RelationshipType</td>
</tr>
<tr>
<td>ParkingOwner-Employee/Department</td>
<td>Union Relationship / Conceptual</td>
<td>UnionRelationshipType</td>
</tr>
<tr>
<td>ParkingOwner - Employee</td>
<td>Union Superclass to Union Subclass / Physical</td>
<td>UnionToSubUnion-RelationshipType</td>
</tr>
<tr>
<td>ParkingOwner - Department</td>
<td>Union Superclass to Union Subclass / Physical</td>
<td>UnionToSubUnion-RelationshipType</td>
</tr>
</tbody>
</table>

More complicated requirements

With emphasis on more complicated requirements and some creativity, more dynamic requirements and constraints of data entry could be predicted in metadata. Some of these requirements might be as follows:

- The value range of a column: the feature of defining valid range of values for a column;
- Arc relationships: selecting a single relationship from several specified relationships of an entity;
- Conditional relationship: the condition of making or not making of a relationship based on the entry value of an attribute of the entity;
- Conditional range of values: the dependence of value range of an attribute on the entered value of another attribute of the same entity;
• The condition of converting the values: dependence of values an attribute could be changed to on the current value of the attribute.

For example, through adding three simple entities to the model (figure 13), value range of a column, conditional relationship, conditional range of values and conditions of value conversion could be covered. Using the ColumnValue entity, the value range of a column is defined. Using the ColumnValue_Relationships entity, disabling or enabling of relationships based on column values is possible and with the ColumnValue_Columns entity, it is possible to determine the list of valid values for a column based on another column's value (or based on the same column's value in case).

Fig. 13   Adding some complicated requirements of data entry to the model

For example, by placing the “male” and “female” values in the ColumnValue table for the Gender column of the Employee table, in the data entry process these values would be selectable rather than entered by user. Now presume we need another rule that do not allows male employees to involve in projects. This rule can be defined easily in the metadata by adding a record in the ColumnValue_Relationship table for previously defined “male” value and then setting the project relationship disabled.

Generic designs in metadata

Definitely, with more coverage of data entry requirements using the mentioned methods, the obtained model would be more efficient and applicable. However, such modifications in metadata model could also lead to some complications. Complexity of metadata structure or its non-general structure would bring some drawbacks such as less understandability and less generality of the model. In order to meet the problem, some generic designs could be used to define some of the requirements in the model if needed. Using these generic designs, there would be no need to define all required attributes of tables, relationships and columns explicitly in the metadata. Moreover, it might be necessary to define many application level attributes in the model. For example think of a requirement that needs to make emphasis on some columns in the UI by making the font of the column's label bold. This requirement is a trivial application level requirement and it is not wise to put it directly in the metadata model.

A general strategy to make such changes in the model might be using dynamic properties in form of tag/value. In Fig. 14, an example of the mentioned design technic is presented. As shown, the extra attributes of tables and columns could be defined in form of dynamic data rows rather than static physical attributes.

Fig. 14   Generic constructs to define attributes of tables and columns
Model customization

Using the techniques mentioned in the previous sections, an enriched metadata would be achieved that can be generalized to many common data entry solutions. The next supplementary step of metadata enrichment, could be customization of metadata to fit any specific data entry approach. These subjective customization could include adding optional types of attributes, entities or relationships to the metadata. For example, in Fig. 15, according to directed nature of data entry scenarios in applications, instead of one physical relationship between each pair of entities, two directed relationships could be defined in the metadata. This way, in addition to possibility of separating attributes of each direction, the process of navigation through entities and relationships in data entry forms are facilitated.

VI. INITIALIZATION OF CONSTRUCTS

A metadata with no data has no use and meaning in practice. As it was mentioned, data for core elements of metadata could be derived from the database. However, to initialize the extended parts of metadata what strategy could be considered? According to key role of metadata in automated data entry solutions and need to careful, quick and integrated management of all details, it is recommended to use a software program to manage metadata. With such an administrative program, in addition to apply control rules in metadata initialization, the involvement of user in lots of technical complexities (for example implementing metadata manipulation queries) could be avoided. In designing such a metadata management program, a combination of three methods of coding, rule engine and design expert intervention is generally proposed.

- Coding: for initialization of some constructs of metadata, fixed code snippets could be applied. Algorithms of these codes would not change a lot over the time. The examples could be the previously mentioned technics for extraction of initial structure of metadata from the database, extracting list of available values of a column in database (for example to fill the ColumnValue entity automatically) or converting types of relationships to each other in the metadata.

- Rule engine: experience have proved that there are semi-fixed rules and techniques in designing databases. These rules may be observed by a specific designer team or in a specific solution; although they may vary from a problem to another or from a designer team to another. For example, the rules of naming tables and columns are in this group. To extract some information of metadata due to these changeable designs, it is recommended to use rule engine environments. Hence, the way of implementation of these rules in extraction of information from the database is not depended on principal code of the program.

As an example, suppose a database that has 30 reference tables all starting with “Ref_” like “Ref_City”. Think of a metadata management program that has a module to detect reference tables. In facing such a situation the mentioned module
can be implemented in two ways; hardcoded or dynamically programmable. Since a metadata management program is supposed to work with various database designs it would be better to use the dynamic rules.

![Sample dynamic rule for detecting reference tables](image)

**Fig. 16 Sample dynamic rule for detecting reference tables**

Design expert intervention: a known challenge with the perception of concepts from the physical implementations of databases or literally reverse database engineering could be ambiguousness and mismatches in translating different design layers to each other (Conceptual, Logical and physical layers). In ref [10], the necessity of interpretations of user in validation of the concepts perceived from the database and in ref [11], the necessity of intervention of user in the process of conceptualization of database reverse engineering are emphasized. In general, the necessity of applying user's comment (as complementary part of mechanized methods) could be caused by lack of certain correspondence in mapping models from conceptual to physical level and vice versa. Moreover, it has been observed that in implementation of lots of concepts in the database, implicit structures (e.g. Triggers) are used. Regardless of these explanations and considering the defined metadata, there are some configurations in the metadata, which should be initialized in customized way due to user's decision. For example, the way of setting alias names for tables and columns or setting the display order of columns are in this group.

### VII. SOLUTION IN PRACTICE

In this section, the proposed solution for the formation and enrichment of metadata and also its effectiveness in covering some of the fundamental data entry requirements are examined. For this purpose, the real database of AdventureWorks is used which is a well-known sample database used in many of the Microsoft's documentations. The version of the AdventureWorks database used in this paper is 2012 and it is available on [12]. Also [13] helps to understand the database design as it describes all the tables in the AdventureWorks.

In order to manage the metadata model as well as its use in designing data entry forms, existence of two programs is presumed; one manages the information in the metadata, and the other uses this information in generating the data entry forms of the target database. It is necessary to mention that the examples provided of the features of these two programs are merely due to the visualization of the proposed concepts and the details of the design or implementation of the mentioned programs - due to irrelevance with the main theme of this research, which is the enrichment process of the metadata - will not be covered. Figure 17 shows how the four concept of metadata management program, data entry program, metadata database and target database interact with each other.

![Proposed applications and databases interaction pattern](image)

**Fig. 17 Proposed applications and databases interaction pattern**

As previously mentioned, in order to extract the primary elements of the metadata, the data dictionary of target database could be used. For example, by running the queries shown in figures 18 and 19 in SQL Server, the details of all relationships in AdventureWorks are accessible, including relationship names and involving primary and foreign key tables and columns.
In this way, by adding a feature to a metadata management program, all 72 tables, 91 relationships, and 749 columns of the AdventureWorks database could be extracted and then transformed into the core metadata model. In Fig. 20, a view of a sample metadata management application and the entities extracted from the AdventureWorks database is displayed.

Interestingly, this basic information in the metadata can be used to design some data entry forms and the relationships between them. For example, in Fig. 21 the UI of the ProductCategory form in a data entry application is displayed.

Considering the role of metadata in explaining automation solutions, related works like [1], [2], [5] and [6] have optimistically defined the metadata structure at the current depth, in other words, a repository of elementary information about tables, columns and relationships. But, as mentioned earlier, this paper attempts to focus on the metadata and its capabilities in more depth.

Now, using the metadata extension process described previously, more data entry requirements could be applied in the ProductCategory form. In the first step, metadata could be extended according to the "Common requirements" method discussed previously. Thus, it is possible to define some simple attributes for any concept in the metadata core and then managing them in the metadata management program. For example, suppose it is intended to apply three new requirements in the ProductCategory form. The first requirement is to change the form name from the "ProductCategory" term to the "Product Category", the second requirement is to set the columns of rowguid and ModifiedDate to read-only mode and the third requirements is to hide the relationship with the ProductSubcategory entity. All these requirements can be resolved by adding simple attributes to the current metadata elements and then setting them in the metadata application. The methods of setting the three mentioned requirements in the metadata management program are shown respectively in Figures 22, 23 and 24.
After applying these changes in the metadata, the ProductCategory form will be changed to Fig. 25.

In the next step of metadata enrichment, the concept of specialization of the core elements such as tables, relationships, and columns was discussed. For a better understanding of the capabilities added to the data entry forms by specializations in the metadata, data entry form of the Person table is considered. In this form, two requirements are defined; the first one is to apply a validation rule to the PersonType field and expresses that the field should only accept string values composed of 2 characters. The second requirement is the ability of defining only one email address for each person. In other words, to set the relationship between the Person entity and the EmailAddress as a one-to-one relationship and not one-to-many. As shown in Fig. 26, currently it is possible to set a PersonType value with a length of more than two characters. It is also possible to define multiple e-mail addresses through the link to the EmailAddress form.

In order to meet the requirements, the metadata can be extended according to the previously proposed specialization methods. For
the first requirement, specialization of the string columns in a new entity is proposed (inherited from column entity as Fig.11). Hence, attributes such as the length and the format of a string column could be defined.

For the second requirement, the same way applies. More precisely, some specialized entities for relationship types should be added to the model and then assigned to existing relationships in the metadata application. Figure 28 shows how to specialize the relationship between Person and EmailAddress into a one-to-one relationship.

After applying these changes in the structure and content of the metadata, the Person form will be changed to Figure 29. Firstly, the PersonType field would not accept values with more than two characters; secondly, only one email address can be entered for each person.

In order to examine the higher-level relationships in the AdventureWorks database, we consider the most significant inheritance relationship which introduces BusinessEntity table as the superclass and the tables of Person, Vendor, and Store as subclasses. In order to define this inheritance relationship in the metadata, we act as described previously in "Higher level relationships" section. In other words, we classify the physical relationships between the mentioned tables in the form of a higher-level conceptual relationship. Fig. 30 shows how to define these relationships in the metadata management program.

Now, the properties like Disjoint or TotalParticipation can also be defined for inheritance relationships. More important, when designing a data entry form containing ISA relationships (like the BusinessEntity data entry form), instead of visiting ISA relationships separately according to their physical level design, they can be implemented as a group of...
relationships realizing a higher level concept.

However by considering a database with data like *AdventureWorks*, there is a more efficient way to identify and initialize the majority of relationships types rather than assessing them one by one by a system expert; through implementing a simple algorithm in the metadata management program, it is possible to identify most of the relationship types automatically. This algorithm is shown in figure 32.

By running this algorithm on the *AdventureWorks* database, 73 relationships were identified as one-to-many, 11 relationships as one-to-one and 7 relationships as ISA relationships. The ISA relationships are shown in Table 5 and the one-to-one relationships are listed in Table 6. Rest of the relationships are one-to-many relationships.

Table 5  Detected ISA relationships in the *AdventureWorks* database

<table>
<thead>
<tr>
<th>No</th>
<th>Superclass</th>
<th>Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BusinessEntity</td>
<td>Person,Store,Vendor</td>
</tr>
<tr>
<td>2</td>
<td>Person</td>
<td>Employee,Password</td>
</tr>
<tr>
<td>3</td>
<td>Employee</td>
<td>SalesPerson</td>
</tr>
</tbody>
</table>

By closer examination of these relationships, we find that the proposed method succeeds in identifying about 95% of the relationship types correctly, in other words, 86 relationships out of 90 relationships. More precisely, all of the one-to-many relationships are correctly detected and from the one-to-one relationships, only three of them (marked with * in the table) - in spite of their one-to-one data match in the database - require to change to one-to-many relationships. Also, among the detected inheritance relationship, it would be conceptually better to change the relationship between *Person* and *Password* tables to a one-to-one relationship. As mentioned earlier, these conversions can be easily done in the metadata managing program.

The next idea referred in enrichment methods was the possibility of including more complex requirements in the metadata. Using these definitions in metadata, it is possible to improve the raw data entry forms to forms containing more and more business rules. For example, remember the *PersonType* field in the Person form. In fact this field can only accept values from a certain range of two-character strings (‘IN’, ‘EM’, ‘SP’, ‘SC’, ‘VC’ and ‘GC’). To cover this issue, a change in the metadata model and application can be done that allows to define a set of values for a column. In Figure 33, the feature of setting a set of values for a column is displayed. Note that these values can be directly extracted from the target database automatically.
Therefore, the user can only select the value of the PersonType field from a predefined set of values.

At the end of this section, the proposed idea of inserting some of the requirements in the metadata in form of general and dynamic construction would be considered. These categories of requirements can be either application level or technical requirements that are not directly related to the business rules of data entry forms. For example, setting a distinctive color for a column’s header might be a requirement. In Figure 35, the proposed method of defining the color property as a tag and assigning it to columns through the ColumnTag table is presented. Also in Figure 36, the effect of these definitions in the metadata on the data entry form is shown.

VIII. CONCLUSION

In this study, the idea of generation and enrichment of metadata for automatic implementation of data entry forms based on database models is discussed. According to the mentioned, the structure of metadata could be systematically enhanced from a simple source of database schema depended on data dictionary to a more comprehensive model. A key point in discrimination of the solution is commitment to the known schema of relational databases and development of metadata based on the extracted initial elements.

The enrichment process begins from the extraction of common elements such as tables, columns and relationships and representing them in a relational model. Afterwards, through adding some properties to these elements, some common requirements of data entry would be covered. In the next step, for purpose of more structural development of model, some specialization techniques on the existing elements of the metadata are discussed. Then, the idea of considering some higher-level concepts of target database in the metadata design like ISA-relationships is proposed. The possibility
of adding more complex rules and validation requirements in the metadata is the next discussed extension. However, there would be no need to apply all requirements and modifications explicitly in the metadata structure. To decrease the size of metadata, some attributes could be defined and initialized in form of tag/value using generalization techniques.

Moreover, for purpose of initialization of the proposed structures in metadata and generally metadata management, necessity of implementing a controlling software program is referred. According to this idea, by means of this program, the data in metadata could be initialized and managed through using a combination of three methods of coding, rule engine and direct intervention of user. In Fig. 37, the function of a metadata management program in a general automation approach of data entry is illustrated.

In the final section, in order to observe the functionality of the proposed solution, examples of initialization, extension and application of the appropriate metadata for data entry of a standard database is provided.

Considering the data nature of metadata, the enrichment of metadata is not limited to mentioned requirements and it is possible to extend it over other directions and applications. For example adding user profile concepts and related access controls for users or designing schemas for configuring a variety of reports in the metadata can be a supplementary goal.

REFERENCES