

Application generation for the simple database browser based on the ER diagram

Guntis Arnicans

University of Latvia
Faculty of Physics and Mathematics
Rainis Blvd. 19, Riga LV-1459, Latvia
garnican@lanet.lv

Abstract

This paper describes a development technique for the rough browser of a database. The offered data browser or data management system can be generated automatically from the a physical data model represented by an ER diagram. The ER diagram used to generate a target application source text is described by common simple concepts and by some additional attributes with default changeable values. All the ER diagram elements are mapped to standard screen object groups and are the main components in the target system screens. Various screen templates for generated applications are defined depending on the entities, the relationships between them and an acceptable user interface. The generated application can be used for database browsing, data manipulating, system prototyping, fast developing of simple information systems and data analyzing.

1. Introduction

There are many strategies for information system development and project management in nowadays. The development of very advanced CASE tools lets us use the Rapid Application Development (RAD) methodology. This approach includes several steps - business modeling, data modeling, process modeling, application generation, testing and turnover [1]. In this paper the simple technique is described that allows to develop specific information system - database browser and data manager. The main attention is turned to the generation of application.

Many powerful tools already exist to assist in system development with RAD technology, for instance, Oracle Designer/2000 [2], [3]. But practice shows that these tools sometimes are not useful. The reasons are that they are expensive and require high educated and trained specialists to work with them. And we have to work hard for some time. The quality of the information system mostly depends on the data model.

Especially this data model (object model) is critical when we use Object Modeling Technique (OMT) [4]. Let us assume that we already have designed the physical data model for our database. Like a conceptual data model the physical data model can be described by Entity-Relationship Diagram (ER diagram). This is popular instrument to describe data model or database and these diagrams are known for most programmers.

Our goal is offer to the user a technique that allows to create a database browser from the physical data model described by the ER diagram. What does the developer have to do? He has to create a simple ER diagram for the existing or the planned database. We do not care in whether he makes a serious analysis and design, whether he creates the ER diagram “on the fly”. He obtains quickly generated database browser, a simple information analysis and filtering tool, a data entering and editing tool, a prototype for the most serious business application, simple database testing tool. Thus, while the real system is developed, a robust information system is obtained.

2. ER diagram - the source for generation

2.1 The elements of the ER model

The ER diagram is a source for application generation. We consider only the ER diagrams that represent physical data models. The main objects we manipulate are the ER diagram descriptor (describing common features of database), the entities (representing tables in our database), the relationships (representing the relations between the entities), the fields (representing the data fields in the record of the physical table).

Developers use various variants of the ER diagrams. Let us take a diagram that is not too simple and not very complex. The ER diagram can be described by the object model shown in the Figure 1. We choose the following elements in the ER diagram:

- **diagram descriptor** - *DiagramName*;
- **entity** - *EntityName*, [*EntityType*], *PrimaryKey*, *UniqueKey*, *Index*;
- **field** - *FieldName*, [*FieldType*], *DataType*, [*Visibility*], [*ShortView*], [*LongView*];
- **relationship** - *EndEntity_1*, *EndEntity_2*, *Cardinality_1*, *Cardinality_2*, *Role_1*, *Role_2*, *ForeignKey_1*, *ForeignKey_2*.

The attributes in brackets are introduced for generation better applications. For simplicity we assume that primary key and foreign key are represented only by one field. In our simplified model we assume that an index is created from a field without using any function. It is not so hard to expand the model to use a combination of fields as the primary key (as the foreign key respectively) and a function of fields as the index.

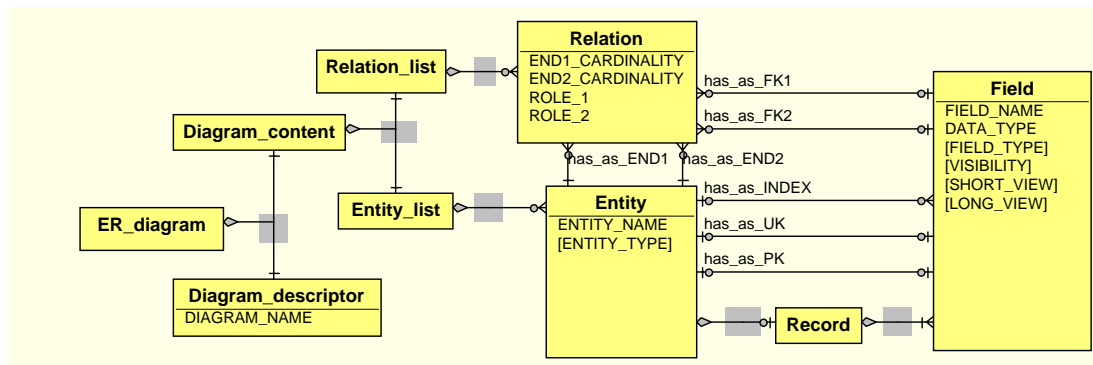


Figure 1 The conceptual object model for the ER diagram

2.2 Additional elements of the ER diagram

Let us introduce several new attributes for the ER diagram. These attributes provide the additional information for the application generation program to generate a more convenient application.

Entity type is a special attribute of an entity that allows us to generate application screens with a specific information layout and data manipulation means. This attribute is stated automatically and depends on the relationships between the entities. The user can correct it while automatic type fixing.

Field type is an automatically calculated attribute of the field. If the field is defined as the primary key of *Entity* via relationship *has as PK* (Figure 1) then the field has type **PK**. Similarly we define type **UK** (via relationship *has as UK*) and type **FK** (via relationship *has as FK1* or *has as FK2*). Otherwise the field type is **Attribute**.

Visibility is a feature of a field. It states whether the information associated with the field is or is not displayed to the user. The default value of *visibility* is TRUE for fields with types **UK**, **FK** and **Attribute** but FALSE for type **PK**.

ShortView and **LongView** are field attributes that define how the entity record can be best displayed on the screen.

2.3 Entity types

Entity type is an important concept in our application generation ideology. Let us define the following *entity types*.

- **Domain** - list of standard data elements, allowed values for an attribute or an object property.
- **SimpleEntity** - simple object that is determined by a set of attributes or standard data elements.
- **ComplexEntity** - complex object is similar to *SimpleEntity* but it includes other simple or complex objects.
- **Link** - logical relation between at least two simple or complex objects.

2.4 Algorithm for entity type determination

1. Scan through all the entities and fix those that have no field with type *FK* (foreign key) and all the incoming ends of whose relationships are either of cardinality 1 or 0..1. We have to assign the type *Domain* or *SimpleEntity* to the fixed entities. The type *Domain* is assigned by default.
2. Scan through all entities without a fixed type and fix any one that has fields with type *FK* referenced only to entities with type *Domain* and all the incoming ends of whose remaining relationships are either of cardinality 1 or 0..1. The type *SimpleEntity* is assigned to the fixed entities.
3. Scan through all entities without a fixed type and fix each one which at that moment is referenced by a foreign key from any entity with type *SimpleEntity*, *ComplexEntity* or undefined type. The entity can also reference to itself. The type *ComplexEntity* is assigned to the fixed entities.
4. Scan through all entities without a fixed type and fix any one that has at least two fields with type *FK* that reference to entities with type *SimpleEntity* or *ComplexEntity*. We have to assign the type *Link* or *ComplexEntity* to the fixed entities. The type *Link* is assigned by default.
5. The type *ComplexEntity* is assigned to the any remaining entity without fixed type.

The user decides on the entity type (1. and 4. step) according to the semantics of the entity and on how he wants to see the information on the screen.

2.5 Textual visualization of entity record

We need to define several textual visualizations of an entity record in our system. A textual visualization of an entity record is mapping the field values to text. These texts (or entity views) are used to display an entity on the screen. Let us define three functions: *shortView()*, *longView()*, *allFields()*. The *shortView()* displays some of the record fields in an ordered sequence. The order is defined by assigned an order number to attribute *ShortView*. If the field is not included in short view the 0 is assigned to *ShortView*. The similar approach is used for *longView()*. The *allFields()* displays all fields.

2.6 ER diagram for example

The ER diagram for example is given in Figure 2. The attributes of each field are given in the following sequence - *field name, data type, field type, visibility* (T for TRUE, F

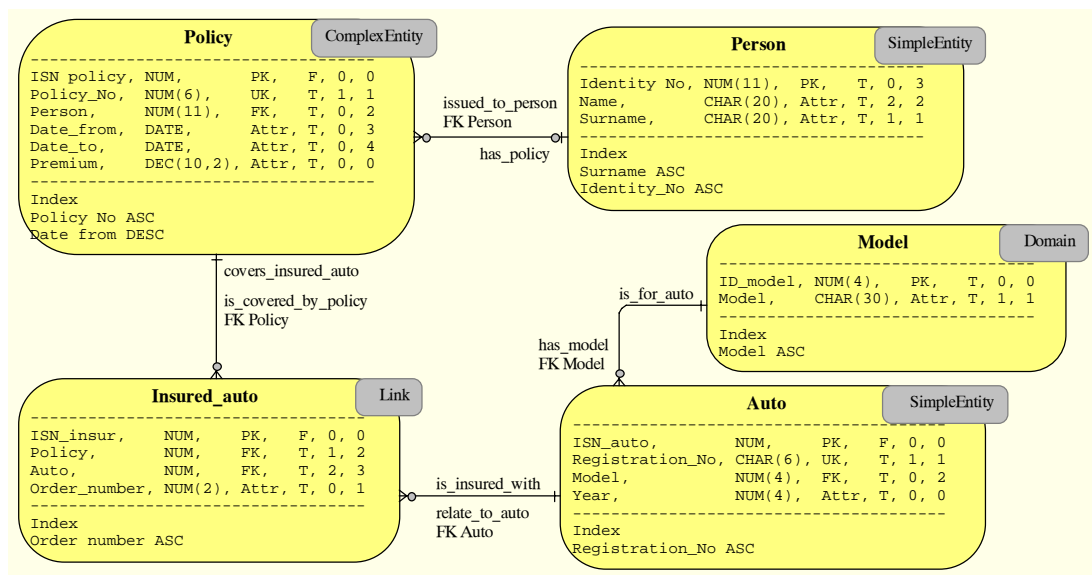


Figure 2 ER diagram for example

for FALSE), *ShortView*, *LongView*.

3. Application generation

The general idea of generation is to create a system with a predefined user interface and functionality. The features of the system depend on the generator. We can generate the whole application for the ER diagram or only some components for this application.

3.1 Screens and menu system

The quality and usefulness of the generated system depends mainly on the generated screen system. Let us define several standard screens that allow us to handle data in the database. The basic generation principle in our approach is to generate the specific data editor for one or several tables connected by relationships. We generate a set of related screens and this enables direct transition from one screen to another.

The primary objects in our system are entities and relationships between them. We define some screen types with different user interface and different functionality for any entity or relationship. For instance, we can display on the screen entity, links to other entities (relationships in the ER model), information about related entities or display related entities by some relationship. The screens of all types are generated for each entity (accordingly to entity type) and for each relationship if the generation options do not define another behavior.

The menu provides access to any generated screen and standard operation defined for any application. The screens are organized in a hierarchy for easy orientation.

3.2 Screen components

Every screen logically consists of two large sets with different screen objects.

- **Information group** - screen objects that display information stored in the database and objects that are generated from the ER model. This group mainly consists from table fields and relations between entities. The basic *information subgroups* are: **Field group** (screen objects that display visible record fields), **Entity presentation group** (screen objects that display the record of the entity or the list of records), **Relationship presentation group** (screen objects that display relationship between entities), **Order button group** (radio button group that determines the order in what records are ordered).
- **Management group** - screen objects that provide additional management over the data stored in the database. Their generation depends on the screen type. The following *management subgroups* are defined: **Edit button group** (a group of buttons for entity record editing - *New, Edit, Save, Delete, Cancel* buttons), **Locate button group** (a group of buttons for locating the desirable record - *First, Next, Previous, Last, Find* buttons), **Print button** (a button for printing the current record

or a record list), **OK button** (a button for leaving the window), **Control button group** (specific buttons included in the screens of specific type).

4. Mapping ER model objects to application objects

4.1 Mapping sequence

A rough algorithm for application generation is the following.

- Map the ER diagram name to application name.
- Generate the screens for each entity (all screen types allowed for given entity type):
 1. Generate screen name from entity name.
 2. Generate each information subgroup needed for the given screen type. The layout of this group (horizontal, vertical, tabular or other) is not the subject of this paper.
 - a) Generate all field groups for given entity.
 - b) Generate information about all related entities via foreign keys.
 - c) Generate information about all related entities via relationship without a foreign key at the end of given entity.
 - d) Generate order button group for given entity.
 3. Generate all management subgroups necessary for the given screen type.
- Generate screens of all allowed types for each relationship.
 1. Generate screen name from related entity names and role names.
 2. Generate information group as for the entity screens.
 3. Generate management group as for the entity screens.
- Generate menu system organized by screen types. The deepest menu items are generated from the entity name (for entity based screen) or from two entity names and role names (for relationship based screen). The menu item will open the window for the specified screen type and entity (or relationship) as the central object.
- Generate additional menu items for standard operations.

4.2 Field mapping

A field with type *Attribute* maps to screen object group *AttributeInfo* (Figure 3).

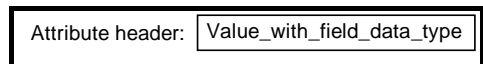


Figure 3 Screen object group AttributeInfo

Attribute header is TextBox object with value generated from *FieldName* and EditText object contains the value with type defined by field *DataType*. The EditText length depends on the data type but is limited by some reasonable maximal length. If necessary scrolling through field is provided.

A field with type *PK* (primary key) maps to screen object group *PkInfo* (Figure 4). It is similar to *AttributeInfo* but it also has the button **Gen**. The user can manually enter a value for the record primary key or generate a value automatically by pressing the button *Gen*. Key generation depends on the selected default rule. When the user leaves EditText the system checks whether the value is unique.

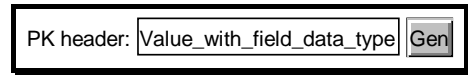


Figure 4 Screen object group PkInfo

A field with type *UK* (unique key) maps to screen object group *UkInfo* (Figure 5). This group is similar to the screen object group *PkInfo*.

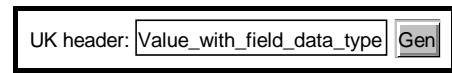


Figure 5 Screen object group UkInfo

A field with type *FK* (foreign key) maps to screen object group *FkInfo* (Figure 6). The content of this group depends on the screen type, relationship type and connected entity type. *Fk header* is TextBox object with value generated from *FieldName*. CheckBox is an optional element and is generated when corresponding relationship at the opposite end has cardinality 0..1, otherwise (cardinality is 1) CheckBox is not generated. We can assign an empty value to the foreign key field by turning off CheckBox. *EntityInfo* is another screen object group (see Entity presentation). The button **Go** is optional and its generation depends on the screen type.

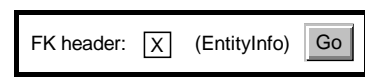


Figure 6 Screen object group FkInfo

4.3 Entity presentation

An instance of entity is represented by screen object group *EntityInfo*. Let us define several subgroups for *EntityInfo*.

Entity with type *Domain* is represented by *DomainInfo* (Figure 7).

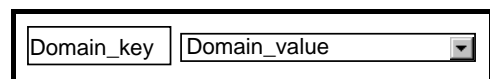


Figure 7 Screen object group DomainInfo

ComboBox provides the selection of domain value and shows the current value. This screen object is obligatory part of the group. EditText *Domain_key* is optional. It is generated by the following rules. At first,

if entity has the visible unique key then *Domain_key* gets the data type from this field. Otherwise, if entity has the visible primary key then it gets the data type from this field. If entity has no visible unique or primary key then *EditBox* is not generated. Both objects always reference to the same table record. *EditBox* can be used for selecting the domain value by entering the key in the *EditBox*. *Domain_value* is the entity representing text depending on the default text function.

Entity with type *SimpleEntity* or *ComplexEntity* is presented by *EntityTextInfo* (Figure 8). *TextBox* contains the entity representing text depending on the default text generation function.

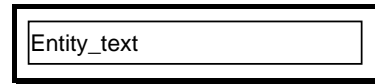


Figure 8 Screen object EntityTextInfo

Several similar entities with type *SimpleEntity* or *ComplexEntity* are presented by *EntityListInfo* (Figure 9). *ListBox* contains entities representing texts depending on the default text generation function. *EntityListInfo*

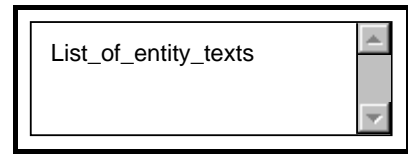


Figure 9 Screen object EntityListInfo

represents also entities with the type *Link* but in the text generation function can exclude one field with type *FK*.

4.4 Relationship representation

One direction of relationship is represented by *RelationshipEndInfo* (Figure 10). Let us suppose that we represent relationship from *Entity_1* to *Entity_2* with role *Role_1*. *TextBox Relation_role* contains text 'Role_1' and *TextBox Relation_end* contains text 'Entity_2'. Button *Go* provides going to the screen that represents entity *Entity_2*.

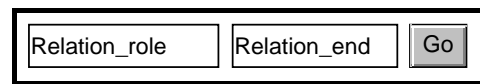


Figure 10 Screen object RelationshipEndInfo

4.5 Mapping of the indexes

If the entity has at least one index then all indexes are mapped to Order button group represented by *OrderButtonGroup* (Figure 11). The first button *None* allows to remove any previously used record sequence. Every next button corresponds to some index.

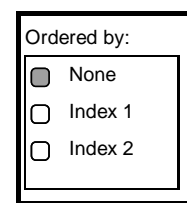


Figure 11 OrderButtonGroup

4.6 Traversing through screens

Traversing through screens is performed by button **Go**. This button is usually attached to the screen object group representing the entity. The button brings us to another screen that belongs to this entity. The button can work in two modes - with filter or without one. If the filter option is chosen then in the newly opened screen we can access only those records that are logically tied with the record or records in the previous screen. For instance, if we fix any car model in the table Model then in the table Auto only cars with this model are accessible.

5. Screen types

The design of screen types depends on the user's needs. Let us define screen templates that can be regarded as basic screen types. The screen examples correspond to Figure 2.

- **Simple entity view**

This screen type can be used for the representation of any entity (Figure 12). The information group contains all visible field groups and *OrderButtonGroup* if any index is defined. The management group contains Edit button group, Locate button group, Print button, OK button.

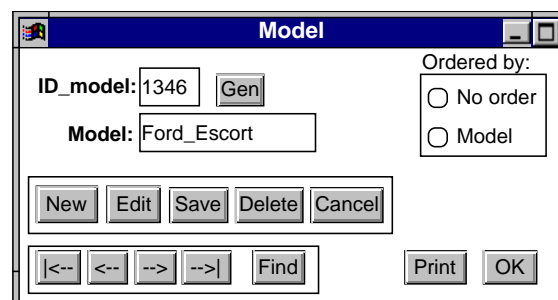


Figure 12 Screen for entity Model

- **Entity view extension with links**

This screen extension can be added to screens of entities with type *SimpleEntity* or *ComplexEntity*. All entities that have type *Link* and are directly connected via relationship with the given entity are shown on the screen. The presentation is performed by screen object groups *RelationshipEndInfo* and *EntityListInfo*. Figure 13 contains a screen fragment for the entity Policy with insured cars (*LongView* option is used) for the current policy.

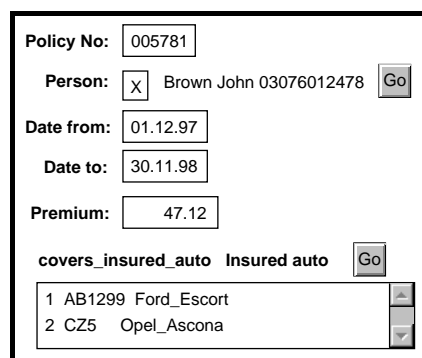


Figure 13 Fragment of screen for entity Policy

- **Entity view extension with relations**

This screen extension can be added to the screens with types *Domain*, *SimpleEntity* or *ComplexEntity*. We represent all relationships that have foreign key at the opposite end (it means that the other entity references to the given entity via foreign key) and the corresponding entities. For the presentation screen object groups *RelationshipEndInfo*, *EntityTextInfo* or *EntityListInfo* are used. All the policies (*LongView* option is used) are displayed for the current person in Figure 14.

005781	01.12.97	30.11.98
010014	05.12.97	04.12.98

Figure 14 Fragment of screen for entity Person

- **Simple link view**

This screen is the special view to the entities with the type *Link* that links together exactly two entities with type *SimpleEntity* or *ComplexEntity* (Figure 15). The *ShortView* option is used to represent linked entities in *EntityListInfo*. A special *Control button group* determines the main ListBox. In this case for a fixed policy 005781 all insured cars are displayed in the other ListBox with label Auto.

Figure 15 Fragment of screen for entity Insured auto

- **Embedded entities view**

This screen type is useful only for the entity with type *ComplexEntity*. Let us take *Simple entity view* as a base for such an entity. Instead of each field group *FkInfo* we incorporate all visible fields from the related entity. We can imagine each embedded entity as a subwindow where it is displayed with *Embedded entities view* for complex entity or *Simple entity view* for the simple entity. E.g., in Figure 13 the objects group with header *Person* is replaced by three screen object groups - *Identity_No*, *Name*, *Surname* from entity *Person*. During the generation process we must beware of cyclic embedding and stop embedding when we discover the cycle.

- **Relationship view**

This screen provides a special view to relationship and entities connected by it. Related entity is represented by *RelationshipEndInfo* and *EntityListInfo*. The main entity is selected by the radio button. Figure 16 shows the fragment for relationship [Model] is_for_auto /

Figure 16 Fragment of screen for relationship between entities Model and Auto

has_model [Auto] with *ShortView* option.

6. Conclusion and future directions

This approach is based on the common ER diagram elements mapping to some screen object constructs. It is not hard to create templates for generation - the standard code for the whole screen and the standard SQL based code fragments for each generated screen object group. Generation basically is code compiling from prepared code templates. This technique partly is applied in practice - the real business applications are developed but screen code is written by hand.

This approach has several future directions that seem very interesting. The screens can be generated dynamically while application is running. E.g., the appropriate HTML page can be generated and displayed. This improvement enables the information view to be changed dynamically.

An ER diagram can be described by context-free grammar (E.g., in BNF notation). The generator is an interpreter that reads the ER diagram as a program and creates a source code for the screens [6]. Other graphical tools, e.g., GRADE [5] can be used to prepare the ER diagram as input statements according to this grammar for the generator.

7. References

- [1] Tucker, A.: The Computer Science and Engineering Handbook, CRC PRESS, 1997.
- [2] Billings, C., Billings, M., Tower, J.: Rapid Application Development with Oracle Designer/2000, Addison-Wesley, 1997.
- [3] Anderson, W., Wendelken, D.: The Oracle Designer/2000 Handbook, Addison-Wesley, 1997.
- [4] Rumbaugh, J.: Object-Oriented modeling and Design, Prentice-Hall, 1991.
- [5] Barzdins, J., Kalnins, A., Podnieks, K. et al.: GRADE Windows: an Integrated CASE Tool for Information System Development, Proceedings of SEKE'94, pp.54-61, 1994.
- [6] Arnican, V., Arnicans, G., Bicevskis, J.: Multilanguage Interpreter, Proceedings of the Second International Baltic Workshop, pp.173-174, 1996.