Griselda Conejo Lopez

*gcl6@gatech.edu*

Yuxiang Liu

*yliu858@gatech.edu*

Kaushik Santhanam

*ksanthanam7@gatech.edu*

CS6400 – Fall 2015

TEAM #3: Reverse engineering of relational   
databases: Mapping from relational model to ER model

INTERIM PROJECT REPORT

Design Document

**Reverse engineering of relational   
databases: Mapping from relational model to ER model**

* **MOTIVATION AND GOAL**

The conceptual modeling is an important phase in the design of successful database applications. The Entity-Relationship model (known as ER model) has become a good way to document this conceptual modeling required for databases. The ER diagram provides information related to the database and it can be used to build the database tables of the relational model. Unfortunately, during the mapping process from ER model to relational database, a considerable amount of information regarding the entities and the relationships between them gets lost as some domain semantics might not be captured, or else are captured but removed due to representation and performance limitations of the database [2].

In some cases, people who works with databases do not see the importance to use ER diagrams, however, the maintenance of an existing database depends primarily on the depth of understanding of its own characteristics, and without the proper documentation this understanding is easily lost, especially when the developers separate.

The approach to resolve this case is the use of reverse engineering to turn from relational database schema to ER diagram. This operation will be helpful to provide missing or alternative documentation, to assist with database maintenance, to make a change of the data model and to offer a deeper understanding of the database itself. The goal of this project is to develop an application that performs the reverse engineering operation on the relational database schema in order to display its Entity-Relationship diagram.

Database Design and Maintenance

Mapping and semantic degradation

Conceptual Model

Mapping and semantic degradation

Reverse Engineering

Logical Data Model

Figure 1. Database reverse engineering process

* **INTENDED APPROACH**

Based on the methodologies proposed by Alhajj [1] and Chiang et al. [2] a summarization of the major steps for our approach are display in the figure below.

INFORMATION EXTRACTION

Foreign Keys

Entities (Relations)

List of Primary Keys

List of Attributes

DBMS

Rev\_Eng

ER CONSTRUCTION

Cardinalities

Relational Intermediate Directed Graph/Optimization

ER Diagram

Relationships

Figure 2. Diagram that summarizes the major steps of our approach

There are two components in the proposed methodology in Figure 2, the information extraction and the ER construction. During the information extraction process, based on the information provided by the user through the data schema, entities will be extracted and classified (strong, weak, etc) and foreign keys will be determined to set relationships between entities and determine cardinality constraints. In the ER construction, the information obtained during the extraction previous steps will be use to extract the ER diagram and mapped it representing entities, attributes and relationships using the notation taught during the class for the ER diagram.

*Relations*

**Example 1:** Relational schema of the COMPANY database (course textbook [3])

**EMPLOYEE (**Fname, Minit, Lname, **Ssn,** Bdate, Address, Sex, Salary, Super\_ssn, Dnumber)

**DEPARTMENT(**Dname, **Dnumber,** Mgr\_ssn, Mgr\_start\_date)

**DEPT\_LOCATIONS(Dnumber,** **Dlocation**)

**PROJECT (**Pname, **Pnumber,** Plocation, Dnumber)

**WORKS\_ON(ssn,** **Pnumber,** Hours)

**DEPENDENT(ssn,** **Dependent\_name,** Sex, Bdate, Relationship)

Relations would be classified based on the properties of its primary key when compared with keys of other relations:

* Strong entity relation: where the primary key of the relation does not contain a key of any other relation (for Example 1: EMPLOYEE, DEPARTMENT, PROJECT)
* Weak entity relation: relation that does not have a key attribute and that is identification dependent on another entity type (for Example 1: DEPENDENT)
* Regular relationship relation: relation whose primary key has been formed by the concatenation of keys of other entity relations (for Example 1: WORKS\_ON)

*Attributes and Relationships*

Not all attributes in an entity relation play a role to establish the relationship between entities. Attributes must be classified first based on their participation on the relation’s primary key, and then, depending on the type of relation, the rest of the attributes will be classified as foreign key attributes or non-key attributes.

* Attribute as Primary Key (PK): this value is used to uniquely identify each tuple in the relation.
* Attributes as Foreign Key (FK): an attribute is considered a foreign key if a subset of a non-primary-key attribute of a relation appears as key of another entity relation.
* Non-key Attributes (NK): attributes that are neither primary key nor foreign keys.

Since the user is providing the attribute that plays the role of the primary key, the next step is to determine the presence of these keys in other relations in form of foreign keys. To achieve this, we will implement the algorithm design by Alhajj in [1] that finds out for each relation, which keys potentially represent a foreign key in any of the given relation. The definition of the foreign keys is essential to define the relationships between entities.

Once the extraction of the foreign keys is completed, the information is sufficient to proceed with the reverse engineering process. The classification of the attributes will be used to create all possible links between relations which will be later used to construct the ER model. Alhajj proposed the construction of the Relational Intermediate Directed (RID) Graph (Definition 4.1 in [1]), which includes all possible uniary and binary relationships between the relations present in the relational database schema. We will relay in the concept of nodes and links of the RID graph to connect the relations (where the foreign key in one relation represents the primary key of the other relation) using the foreign keys information.

*Assumptions*

The limited literature in this topic has shown that the relational database reverse engineering is not an automatic process, and in order to be implemented, several assumptions have to be made. Based on the methodologies proposed in [1] and [2], two main assumption will be made to develop our application:

1. There must be a consistent naming of key attributes: if key attributes have the same domain and represent the same meaning, then they need to have the same name. Otherwise it becomes highly complex to classified attributes in the proper categories
2. Relations should be 3NF: this assumption is made to simplify the extraction process since each relation will correspond to one entity type or one relationship type, rather than corresponding to more than one entity type or a mixture of entity and relationship types. Therefore the application will assume that the initial input is in at least 3NF.

Although these assumptions will be made to develop the application and to test our code and the application’s functionality, the objective is to eliminate these assumptions and still be capable to construct a suitable ER diagram. This means, to works with inconsistent naming of key attributes and work with relations that are 1NF and 2NF.

*Extraction rules*

These rules were established to regulate the extraction process and to build a consistent methodology. Following these rules will only result in one step ahead in the construction of the ER diagram, not the final version of it.

* For foreign keys it is necessary:
* That referring attribute name must match with referred attribute name.
* That the values of the referring relation should be a subset of the referred field
* To examine if the primary keys given by the user are being referred by other relations
* For strong and regular entities: (it is important to remember that relation names maybe an entity or relationship between two entities)
* Relation that contain linked attributes will be initially mapped as entities.
* Assuming that relation 1 is referring relation 2, we will consider relation 2 as foreign key and relation 1 as primary key, which will indicate the existence of a relationship between these two relations
* For weak entities extraction:
* Relations with at least two attributes, in which one of these attributes is key dependent on the primary key of another table will be initially mapped as weak entity.
* If the existence of relation 1 depends on the existence of relation 2, the key of relation 1 is consider partial and part of the primary key of relation 1.
* If the partial key is not associated to the primary key of the other relation, the existence of this relation will be cancelled.
* For relationship extraction:
* Relations that have a primary key related to another relation as foreign key will be mapped as a relationship between those two entities.
* Relations that have an attribute related to another relation as primary key will be mapped as a relationship between those two entities
* Cardinality rule:
* If the primary key of relation 2 is placed in relation 1 as foreign key the used assumption is to set cardinality as (1:M). To set cardinality (1:1) it will be necessary the user’s input regarding participation of the entities participating in the relationship.
* **SOURCE OF DATA**

The information extraction process is the most critical of the entire application and in order to function, the application needs to have information about the data schema available. The user’s initial input to begin the process is the relational data schema. This schema should include relation names, attribute names, and primary keys. (Schemas to test the functionality of the application will be taken from the examples shown in [1] and [2] and the course textbook)

Employee(**essn**, name, age, address) PK: essn

Project(**pno**, manager, location, budget) PK: pno

Works(**essn**, **pno**) PK: essn, pno

Part(**part\_no**, date, color) PK: part\_no

Supply(**pno**, **part**\_no, quantity) PK: pno, part\_no

Figure 3: Example of relational schema

It has been stated that the relational model possess no semantic expressiveness, therefore, the reverse engineering process cannot be a totally automated process. The user’s involvement will be necessary whenever there are ambiguities which cannot be solved mechanically such as dependencies between non key attributes or in the classification of relations.

A sample of the UI has been provided below, where we would be able to add all the relations in the relational schema using interactive methods. After we add the all the required relational schemas, we can start the transformation process by following the steps in the architectural design and utilizing the user input. The user will provide clarifications in places where ambiguities are detected and the application will take the user’s input and proceed further in these situations.

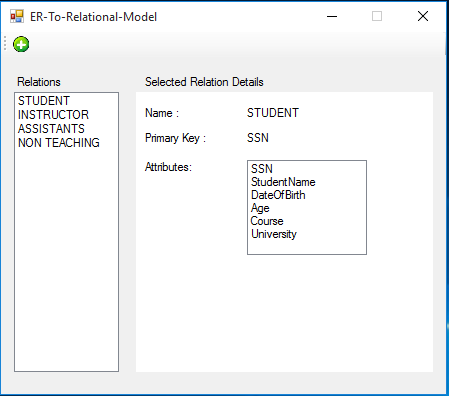


Figure 4: Sample mockup of UI containing all Relations

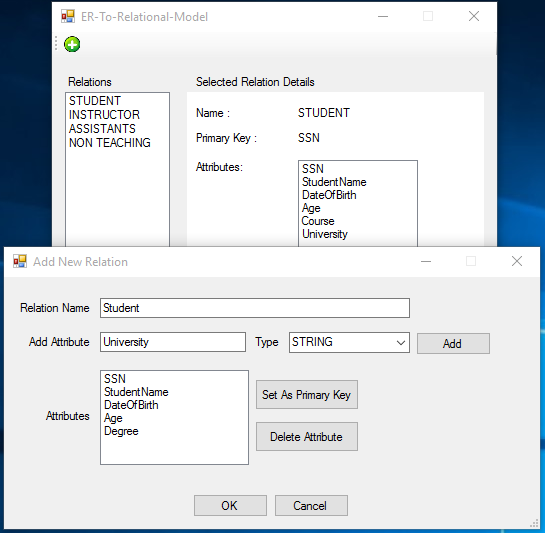


Figure 5: Adding a new Relation to the existing set of Relations

* **ARCHITECTURAL OVERVIEW**

The primary development language in which the application will be built is C# on the .NET Framework. C# was developed using the strength of other languages while avoiding their flaws which gave rise to a fully functioning object-oriented language. The .NET Framework provides high quality in-built libraries and data structures. Development using C# and .NET will help avoid addressing trivial tasks like re-inventing pre-existing data structures and algorithms.

The user will interact through a WinForms based user interface which will provide the user with a GUI to add his/her input and clarifications and also view the resulting ER Models.

The application development will be split in two parts:

1. Business Logic Development
2. UI Development

*Business Logic Development*

Business Logic Development contains developing a core set of classes and routines which will function as the backbone of the application.

*UI Development*

UI Development will contain the classes responsible for interacting and receiving input from the user.

* **ALGORITHMIC OVERVIEW**

The algorithms proposed by Alhajj [1] will be used as a reference to implement during the extraction process in our application. For example, algorithm 1 decides on the presence of candidate key(s) of a given relation R as foreign key(s) within any relation in the relational schema, including relation R itself. Relation R is considered for the possibility of having its candidate key represented within its attributes by foreign key(s). This way, all uniary relationships, i.e., self-references from an entity to itself are detected. Notice that, Algorithm 1 compares attributes from the same syntactic domain even if they have different semantic domains. The reason is that most relational DBMS provide only a very primitive notion of syntactic domains as basic data types. So, user confirmation is required for the derived foreign keys.

**Algorithm 1 (Find Candidate Foreign Keys)**

**Input:** A relation R and all its candidate keys.

**Output:** Attributes in foreign keys(s), that represent a candidate key of R in any relation, including R, are added to the *ForeignKeys* table.

**Steps:**

Consider candidate keys of R in ascending order by their number of attributes;

For every candidate key of R do

Let = Select from R; //Project R on attributes in .

Consider Relations in the relational schema in ascending order by their number of attributes;

For every relation S in the relational schema do

Consider from P(S) each element s, which contains the same number of attributes as candidate key and the same underlying domains;

Let Link# = 1;

For every s do

Let = Select s from S; //Project S on attributes in .

If () then

/\*Attribute(s) in correspond to attribute(s) in , i.e., is a candidate key and is a foreign key.\*/

For i = 1 to n do //n is the number of attributes in each of and

Add the tuple (R, , S, , Link#) to *ForeignKeys*;

// and refer to corresponding attributes in and , respectively.

EndFor

Link# = Link# + 1;

EndIf

EndFor

EndFor

EndFor

**EndAlgorithm**

The number of projections required for Algorithm 2 is (2\*e), where e is the number of edges in the RID graph. Two projections correspond to each foreign key to find out whether it has null values and/or duplicates. It solely utilizes the already computed projections to decide on the cardinalities of the links in a given RID graph. Each link is classified as either 1:1 or M:1: Further, Algorithm 2 decides on the minimum and maximum cardinalities at both sides of the link by investigating whether the link is optional or mandatory on each side. It also indicates candidate is-a links; these are 1:1 links classified as mandatory from at least one side.

**Algorithm 2 (Find Cardinalities)**

**Input:** A relational database and the corresponding RID graph

**Output:** The cardinalities of all the links in the given RID graph

**Steps:**

For every edge (,) in the RID graph do

Let be the primary key of and be a corresponding foreign key in ;

Let R = Select from where the value of is not nil; //keep duplicates.

Let P = Select distinct from R; //Project R on with duplicates eliminated.

//Decide on the the minimum and maximum cardinalities on the side.

If size(P) = size() then

Link (,) as mandatory from the side and its cardinality is 1.;

Else

Link (,) as optional from the side and its cardinality is 0 or 1.;

EndIf

//Decide on the the minimum and maximum cardinalities on the side.

If size(P) < size() then

If size(P) = size() then

Link (,) as mandatory from the side and its cardinality is M 1.;

Else

Link (,) as optional from the side and its cardinality is M 0.;

EndIf

ElseIf size(P) = size() then

If size() = size() then

Link (,) as mandatory from the side and its cardinality is 1.;

Else

Link (,) as optional from the side and its cardinality is 0 or 1.;

EndIf

EndIf

EndFor

**EndAlgorithm**

In the worst case, Algorithm 3 checks all links in the RID graph to locate pairs of symmetric links. For each two symmetric references (, ) and (, ), the algorithm checks whether or is the tail of a 1:1 link. The algorithm eliminates the symmetric link directed out-of the node that satisfies this property, if any. Otherwise, Algorithm 3 checks whether or has any other candidate keys and eliminates one of the symmetric links accordingly. Finally, in Algorithm 3, the user is consulted when it is necessary to select among multiple choices.

**Algorithm 3 (Eliminate Symmetric Reference)**

**Input:** The two tables *ForeignKeys* and *CandidateKeys.*

**Output:** The partial optimized *ForeignKeys* table; only symmetric references are eliminated.

**Steps:**

For every candidate key in *ForeignKeys* do

Let be a candidate key of and let be a corresponding foreign key in any relation, say ;

// and may represent the same relation

If and appear in *ForeignKeys* and are symmetric then

If is the tail of an is-a candidate link according to Algorithm 2 then

Eliminate Link (, );

ElseIf is the tail of an is-a candidate link according to Algorithm 2 then

Eliminate Link (, );

Else

Locate other candidate key(s) of and in *CandidateKeys*;

If has other candidate key(s) then

Remove from *ForeignKeys* all tuples where appears as a candidate key of ;

ElseIf has other candidate key(s) then

Remove from *ForeignKeys* all tuples where appears as a candidate key of;

Else

Consult user to suggest relation ,, where is not a candidate key of ;

EndIf

EndIf

EndIf

EndFor

**EndAlgorithm**

Whether R is classified as a relationship depends on two factors, the number of links connected to R and the number of candidate keys of R: R is classified as a relationship, in case that R has a single candidate key, which is a combination of the primary keys of all the relations connected to R: Otherwise, if R has more than one candidate key, then there are three possibilities to consider; R may be classified as a relationship, a sub-entity, or a weak entity. The first is applicable only if one of the candidate keys of R is the combination of the primary keys of all the relations connected to R. Under such circumstance, the user is consulted to check whether to classify R as a relationship.

**Algorithm 4 (Identify Relationships)**

**Input:** The optimized *ForeignKeys* table and the corresponding optimized RID graph*.*

**Output:** Relationships with attributes, M:M and n-ary relationships.

**Steps:**

For every relation R that appears only in the third column in *ForeignKeys* do

Let w be the number of links connected to R; //All these links are directed out of R.

If w then

Let ,,…, be the relations connected to R;

If the only candidate key of R is a combination of the primary keys of ,,…, then

R represents a relationship between ,,…,;

ElseIf the combination of the primary keys of ,,…, is a candidate key of R then

Consult the user to decide whether R represents a relationship;

Else

R is not a relationship. It may be either a sub-entity, or a weak entity.

EndIf

If R has been classified as a relationship then

Its cardinality is :: … :, where ,, is specified as either 1 or M, as follows

is 1, if the link connecting R and has been classified as 1:1 by Algorithm 2;

is M, otherwise

EndIf

EndIf

EndFor

**EndAlgorithm**

* **DEMO PLAN**

*Demo Plan: What you think you will be able to show as a demo (no need for fancy interface work) – you need to demonstrate the capability and features you have tried to implement.*

**REFERENCES**

|  |  |
| --- | --- |
| [1] | R. Alhajj, “Extracting the Extended Entity-Relationship Model from a Legacy Relational Database”, *Information Systems*, 28(6), pp. 597–618, 2003. |
| [2] | R. H. L. Chiang, T. M. Barren, V. C. Storey, “Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database”, *Data & Data Engineering*, 10(12):107–142, 1994. |
| [3] | R. Elmasri and S.B. Navathe, “Data Modeling Using the Entity-Relationship (ER) Model”, Chapter 7 in *Fundamentals of Database Systems*, Addison Wesley, Edition 6, 2011 |