ISSN 2229-5518

Algorithmic Resolution of an ER Schema into Relational DDL Statements using Artificial Intelligence

Manuj Darbari, Hasan Ahmed, Sunita Bansal

Abstract—The concept of this paper helps resolve an ER schema into an appropriate relational Data Description Language (herein standard SQL statements) while considering all the inherent complexity of such a schema. The algorithm resolves all the relationships between the concerned entities (even weak ones), their associated keys – primary and foreign, attributes – simple, composite and multivalued and relationships types from binary to n-ary. The algorithm (as implemented in ANSI C) is smart enough to consider the case of cascading weak entities.

Index Terms— Entity Relationship Schemas, SQL statements, relational tables.

1 Introduction

Often a need is felt by applications programmers, computer geeks and the like to translate a given ER schema into SQL statements. Such a translation requires a concerted effort to logically optimize that schema considering all the possibilities of entity relationships, primary and foreign keys, all the variants of attributes, weak and owner entities, etc. This leads to a very cautious effort in devising actual SQL tables.

We have just automated this intelligent effort. The user, just, has to input his ER schema in a defined syntax and with a click of mouse, she could get an optimized set of SQL statements which can be readily applied to a standard SQL engine; thereby, creating suitable database tables [1,4].

2. Theoretical Bases

Normally, we have ER schema which encompasses entities (regular and weak) delved in relationships amongst themselves. These relationships can be varied including not only binary 1:1 ones but also binary 1:N, binary M:N and N-ary relationships.

We, then, bring in the case of weak entities which have regular owner entities. In these cases, due consideration is to be given to a case (or similar cases) wherein the owner entity of a weak entity is, itself, a weak entity whose owner is a regular entity. Now, we ought to reasonably handle primary and foreign keys. Besides, not to be forgotten is optimized appropriation of attributes of all the entities concerned. Attributes can also come in different variations which demand different interpretations. For example, simple attributes of an entity is to be handled differently from the entity's composite attributes which, in turn, needs to be handled differently from multivalued attributes of that particular entity.

Our paper does resolve such complex ER schemas into optimized relational SQL statements which can be applied to database engines. Hence, a great reduction in manual effort would be the result.

In our C code, we have adopted the standard mechanisms of ER schema to relational mapping [1,2,3]. Here, we go on to mention how effect such a scheme considering specific cases, one by one.

2.1. Regular Entities

For every regular entity E, there is a relation R having all the attributes of E. We include simple components of a composite attribute (in that case) and choose one of the keys of E as a primary key of R. If that chosen key is composite, the set of simple attributes that form it will form the primary key of R.

2.2. Weak Entities

For every weak entity W with owner entity E, there is a relation R having all the simple attributes (or simple components of composite attributes) of W. We, also, include as

- *Manuj Darbari is working as Associate Professor with the Department of InfomationTechnology in Babu Banarsi Das National Institute of Technology and Management,Lucknow, India E-mail: manujuma@gmail.com*
- *Hasan Ahmed, Consultant at SaharaNext, Lucknow, India, hasansinbox@gmail.com*
- *Sunita Bansal, Research Scholar, Department of Computer Science, JJTU University, Rajasthan, Sunita_bansal301@rediffmail.com*

foreign key attribute(s), the primary key attribute(s) of the relation that corresponds to the owner entity type. This helps the identifying relationship type of W. Herein, the primary key of R would be the combination of the primary key(s) of owner entity(-ies) and the partial key of W, if any. Here, our code also considers the case of that weak entity whose owner entity is also a weak entity and, in turn, whose owner entity is a regular entity. Our algorithm can handle such a cascading to a good number of weak entity levels. In such cases, the naming and generation of primary key(s) of resultant relations of those weak entities is intelligently handled by our code.

2.3. BINARY RELATIONSHIP

For mapping binary 1:1 relationship, the most optimizing method is to go through foreign key approach, which is employed by us. We, first, identify the two relations S and T participating in relationship R. We choose that entity in the role of S which totally participates in R. And in S, we include as a foreign key, the primary key of T. Also, we include all the simple attributes (or simple components of composite attributes) of 1:1 relationship R as attributes of S.

For each regular binary 1:N relationship R, we identify the relation S that is the participating entity on the N side of R. Here, we include as foreign key in S, the primary key of relation T (which is the relation representing the other entity of R). Also, we include all the simple attributes (or simple components of composite attributes) of 1:N relationship R as attributes of S.

For the case of binary M:N relationship R, we create a new relationship S to represent R while including as foreign key attributes in S all the primary keys of the relations that represent the participating entity types as their combination will form the primary key of S. Apart from this, we include any/all simple attributes of the M:N relationship (or simple components of composite attributes) as attributes of S.

2.3. MULTI VALUED ATTRIBUTE

For every multi-valued attribute A, we create a new relation R which includes an attribute corresponding to A, with the primary key attribute P as a foreign key in R, of the relation that represents the entity type that has A as an attribute. In such a case, the primary key of R would be the combination of A and P. Of course, if the multi-valued attribute tends to be composite, we include all its simple components.

2.3. N-ARRAY RELATIONSHIP

For every n-ary relationship R (n>2), we create a new relation S to represent R while including the foreign key attributes in S, the primary keys of the relations that represent the participating entity types. Here again, we include simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types. But, we do maintain an exception here. If the cardinality constraints on any of the entity types K participating in R is 1, then the primary key of S should not include the foreign key attribute that references the relation K' corresponding to K.

3. ALGORITHMIC IMPLEMENTATION

The algorithm is implemented in ANSI C. The algorithm deals with the ER schema complexities in the order: regular entities, weak entities, binary relationships, multi-valued attributes and n-ary relationships, respectively. The actual code can be requested from the authors' email contacts.

The algorithm is smart enough to rename and/or automatically generate new table names, attribute names, keys, etc. Also, the algorithm picks up other important characteristics of attributes such as it being null (or not null) and being unique (or not unique).

The algorithm extensively uses linked lists as data structures. On the other hand, it does use file handling in C. The use of counters and accumulators is, also, quite common.

As for demonstrability purpose in the example quoted, only 'int' and 'varchar' types of attributes (of varying sizes/spaces) have been taken. However, the code is scalable to other data types as well.

The constraints are as follows. The input file has to be strongly typed as the parser moves through character by character. Composite attributes have to be given as simple components of the parent attribute. As per current implementation, the attribute has seven fields: attribute name, data type, whether null, whether unique, whether a part of primary key, whether default, whether multi-valued. The only assumption in the input file is, at least, one output file.

```
# include <stdio.h>

# include <stdlib.h>

# define NULL 0


struct attribute_linked_list

        {


                char attribute_name[50] ;// this shall be attribute name

                char data_type[25] ;// INT or VARCHAR etc

                int not_null ;// if 1 , then it is NOT NULL

                int unique ;// if 1 , then UNIQUE

                int w_primary_key ;//if 1 , then participates in primary key

                int def ;// if 1 , then DEFAULT

                int mulv ;// check whether attribute is multivalued

                struct attribute_linked_list *next ;// pointer to next node

        } ;

        typedef struct attribute_linked_list attnode ; /* attnode type defined */


struct foreign_key_list

        {

                char attname[100] ;// foreign key attribute

                char tabatt[100] ;//refernced table

                char refatt[100] ;// referenced  attribute

                struct foreign_key_list *next ;//pointer to the next node

        } ;
```

Figure 1. Some macros and linked list definitions.

The snapshot of parts of algorithm code is given in figures 1 and 2. The sample file is depicted as figure 3 while figure 4 depicts the resultant output file.

```
main ()
{
        FILE *fpread , *fpwrite ;
        // fpread is a pointer for reading an input file
        // fpwrite is a pointer in which final SQL queries shall be written
        char c , * filename , tempfilename[25] ;
        int tabnodeflag = 1 ;
```

Figure 2 : Start of the main Function

The sample file is depicted in figure 3 as:

```
Student S ( |Roll_No :  INT ( 3 ) : 1 : 1 : 1 : 0 : 0 :$|Name : VARCHAR ( 20 ) : 1 : 0 : 0 : 0 : 1
:$|No_of_Electives : INT ( 1 ) : 1 : 0 : 0 : 0 : 1 :& ) ;

Department S ( |Department_name : VARCHAR ( 20 ) : 1 : 1 : 0 : 0 : 0 :$|Department_ID : INT ( 4 ) : 1 : 1 :
1 : 0 : 0 :& ) ;

Employee S ( |Employee_name : VARCHAR ( 20 ) : 1 : 1 : 0 : 0 : 0 :$|Employee_ID : INT ( 4 ) : 1 : 1 : 1 : 0 :
0 :& ) ;

Course S ( |Course_ID : VARCHAR ( 6 ) : 1 : 1 : 1 : 0 : 0 :$|Course_name : VARCHAR ( 20 ) : 1 : 0 : 0 : 0 : 1
:& ) ;

Project W ( |Project_ID : INT ( 3 ) : 1 : 1 : 1 : 0 : 0 :& ) ;

Other S ( |Other_ID : INT ( 3 ) : 1 : 1 : 1 : 0 : 0 :$|Other_name : VARCHAR ( 20 ) : 1 : 1 : 0 : 0 : 0 :& ) ;

Another S ( |Another_ID : INT ( 3 ) : 1 : 1 : 1 : 0 : 0 :$|Another_name : VARCHAR ( 20 ) : 1 : 1 : 0 : 0 : 0 :& )
;

Strong S ( |Strong_ID : INT ( 3 ) : 1 : 1 : 1 : 0 : 0 :& ) ;

Strong2 S ( |Strong2_ID : INT ( 3 ) : 1 : 1 : 1 : 0 : 0 :& ) ;

Binary_M_is_to_N M ( |Relationship_attribute : VARCHAR ( 20 ) : 1 : 1 : 0 : 0 : 0 :& ) ;

Binary_M_is_to_N_second M ( |Dummy : VARCHAR ( 20 ) : 1 : 1 : 0 : 0 : 0 :& ) ;

N_ary N ( |Dummy : VARCHAR ( 20 ) : 1 : 1 : 0 : 0 : 0 :& ) ;

N_ary2 N ( |Date : DATE : 1 : 1 : 0 : 0 : 0 :& ) ;

Relation B ( |Relation_ID : INT ( 3 ) : 1 : 1 : 0 : 0 : 0 :& ) ;

Faculty W ( |Name : VARCHAR ( 20 ) : 1 : 1 : 1 : 0 : 0 :&#) ;

Weak_entities_present

Project Department X ;

Faculty Student Course X @ ;

Binary_non_miston_relationships_present

Strong Relation Strong ;

Strong2 O Another ~ ;

General_binary_present


Binary_M_is_to_N_second Student Strong2 X ;

Binary_M_is_to_N Student Strong X @ ;

N-ary_relations_present

N_ary2 1Faculty Another Student X ;

N_ary Strong 1Student 1Employee Course X @ ;
```

The snapshot of parts of algorithm code is given in figures 1 and 2. The sample file is depicted as figure 3 while figure 4 depicts the resultant output file.

```
CREATE TABLE Student (
 Roll_No  INT ( 3 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Roll_No  ) ) ;
CREATE TABLE Department (
 Department_name  VARCHAR ( 20 )  NOT NULL  UNIQUE ,
Department_ID  INT ( 4 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Department_ID  ) ) ;
```

Further extension of the output gives the result as:

```
CREATE TABLE Employee (
 Employee_name  VARCHAR ( 20 )  NOT NULL  UNIQUE ,
 Employee_ID  INT ( 4 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Employee_ID  ) ) ;
CREATE TABLE Course (
 Course_ID  VARCHAR ( 6 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Course_ID  ) ) ;
CREATE TABLE Project (
 Department_ID  INT ( 4 )  NOT NULL  UNIQUE ,
 Project_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Project_ID  , Department_ID )
 , FOREIGN KEY ( Department_ID  ) REFERENCES Department ( Department_ID  ) ) ;
CREATE TABLE Other (
 Other_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 Other_name  VARCHAR ( 20 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Other_ID  ) ) ;
CREATE TABLE Another (
 Another_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 Another_name  VARCHAR ( 20 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Another_ID  ) ) ;
CREATE TABLE Strong (
 Super_Strong_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 Relation_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 Strong_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Strong_ID  )
 , FOREIGN KEY ( Super_Strong_ID  ) REFERENCES Strong ( Strong_ID  ) ) ;
CREATE TABLE Strong2 (
 Another_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 Strong2_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Strong2_ID  )
 , FOREIGN KEY ( Another_ID  ) REFERENCES Another ( Another_ID  ) ) ;
CREATE TABLE Binary_M_is_to_N (
 Strong_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 Roll_No  INT ( 3 )  NOT NULL  UNIQUE ,
 Relationship_attribute  VARCHAR ( 20 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Roll_No , Strong_ID )
 , FOREIGN KEY ( Roll_No  ) REFERENCES Student ( Roll_No  )
 , FOREIGN KEY ( Strong_ID  ) REFERENCES Strong ( Strong_ID  ) ) ;
CREATE TABLE Binary_M_is_to_N_second (
 Strong2_ID  INT ( 3 )  NOT NULL  UNIQUE ,
 Roll_No  INT ( 3 )  NOT NULL  UNIQUE ,
 PRIMARY KEY  ( Roll_No , Strong2_ID )
 , FOREIGN KEY ( Roll_No  ) REFERENCES Student ( Roll_No  )
```

Further extension of the output gives the result as:

```
, FOREIGN KEY ( Strong2_ID ) REFERENCES Strong2 ( Strong2_ID ) ) ;
CREATE TABLE N_ary (
Course_ID VARCHAR ( 6 ) NOT NULL UNIQUE ,
Employee_ID INT ( 4 ) NOT NULL UNIQUE ,
Roll_No INT ( 3 ) NOT NULL UNIQUE ,
Strong_ID INT ( 3 ) NOT NULL UNIQUE ,
PRIMARY KEY ( Strong_ID , Course_ID )
, FOREIGN KEY ( Strong_ID ) REFERENCES Strong ( Strong_ID )
, FOREIGN KEY ( Roll_No ) REFERENCES Student ( Roll_No )
, FOREIGN KEY ( Employee_ID ) REFERENCES Employee ( Employee_ID )
, FOREIGN KEY ( Course_ID ) REFERENCES Course ( Course_ID ) ) ;
CREATE TABLE N_ary2 (
Roll_No INT ( 3 ) NOT NULL UNIQUE ,
Another_ID INT ( 3 ) NOT NULL UNIQUE ,
Name VARCHAR ( 20 ) NOT NULL UNIQUE ,
Roll_No INT ( 3 ) NOT NULL UNIQUE ,
Course_ID VARCHAR ( 6 ) NOT NULL UNIQUE ,
Date DATE NOT NULL UNIQUE ,
PRIMARY KEY ( Another_ID , Roll_No )
, FOREIGN KEY ( Course_ID , Roll_No , Name ) REFERENCES Faculty ( Course_ID , Roll_No , Name )
, FOREIGN KEY ( Another_ID ) REFERENCES Another ( Another_ID )
, FOREIGN KEY ( Roll_No ) REFERENCES Student ( Roll_No ) ) ;

CREATE TABLE Faculty (
Course_ID VARCHAR ( 6 ) NOT NULL UNIQUE ,
```

Figure 4: Generated SQL File

4. Conclusion and Future Scope

The concept of this algorithm can be applied for further resolution of complexities like specialization and class diagrams (as in OOP and data modeling). Also, with the help of GUI, this could be time-saver tool from advanced and naïve users to scholars and practitioners.

References

[1] Chen, P. 1976. The Entity-Relationship Model--Toward a Unified View of Data. In: ACM Transactions on Database Systems 1/1/1976 ACM-Press ISSN 0362-5915, S.

[2] Elmasri, Navathe, Somayajulu, Gupta. 2006. Fundamentals of Database Systems. Pearson Education. ISBN 81-7758-476-6.

[3] CODD, E.F. Normalized data base structure: A brief tutorial. Proc. ACM-SIGFIDET 1971, Workshop, San Diego, Calif., Nov. 1971, pp. 1-18.

[4] Codd, E.F. (1990). The Relational Model for Database Management (Version 2 ed.). Addison Wesley Publishing Company. ISBN 0-201-14192-2.