

DERSQL, GENERATING SQL FROM AN ENTITY-RELATION DIAGRAM

Andrea Domínguez-Lara and Wulfrano Arturo Luna-Ramírez
Universidad Autónoma Metropolitana-Cuajimalpa
Vasco de Quiroga 487, Santa Fe Cuajimalpa, Cuajimalpa de Morelos,
Mexico City, Mexico

ABSTRACT

The automatic code generation is the process of generating source code snippets from a program, i.e., code for generating code. Its importance lies in facilitating software development, particularly important is helping in the implementation of software designs such as engineering diagrams, in such a case, automatic code generation copes with the problem of how to obtain code from a graphic representation, for instance an UML diagram or a Relational Diagram. Some advantages of automatic code generation are: a) to obtain the source code more quickly and to do it with lower margins of error; b) it is promising to be applied in teaching contexts, whilst provide instructors with a tool to teach, the expected results of assignments can be assessed by comparing the results of students and the automatic generated code.

Furthermore, one of the most frequently tasks in classrooms when teaching relational databases is the design of Entity-Relationship Diagrams which eventually become SQL code. The manual transition from an Entity-Relationship Diagram to SQL code is a time-consuming process and requires of a skilled eye to be successfully performed.

In this paper, we present *DerSql*, an extension of the DIA Diagrammer, a well-known free software engineering tool, to automatically generate SQL code from an Entity-Relationship Diagrams. The results are tested for the case of 1 – 1 and 1 – n arities relationships. We consider that *DerSql* represents a remarkable tool for teaching while it is a promising advance in developing DIA as a 4th Generation software engineering application.

KEYWORDS

Automatic Code Generation, Entity-Relationship Data Model

1. INTRODUCTION

In the current society Data Bases (DB) are imprescindible tools and the right design and usage is a critical activity in software development. As in any other software engineering project, in the design of a DB, a methodology is followed starting from knowing the need to be satisfied and to propose a solution for a specific customer. So, DB methodology fulfills different phases such as planning, analysis, and creation.

Once the trouble is defined and identified it is necessary to represent it in a *conceptual model*. The conceptual model provides a clear and more real idea of the problem to be solved. There are many models to represent conceptual model, one of the modeling tools more used by either designers or students that are learning relational databases is the Entity-Relationship Diagram which eventually will be transformed into SQL code.

2. ENTITY RELATIONSHIP MODEL

Entity relationship model is a well-known approach to design DB. Its importance lies in representing data from the real life in an abstract way to understand the user requirements and reach a good representation of the related data which eventually will be transformed into SQL code. One of the main tools of this model is the Entity-Relationship Diagram (ERD) which represents different elements as symbols (Bertone, & Thomas, 2011): a) entity: it is represented by a rectangle, and it refers to an element or object of the real world; b) attribute: it is represented by an ellipse, they reflect characteristics belonging to entities and they are

connected to entities by a line; and c) relationship it is represented by a rhombus, and it means some sort of association between entities.

In accordance with Sanchez (2019), cardinality is the number of relationships in which an entity can appear. Given a set of relationships between entities, the correspondence of the cardinality can be: one to one (1:1), one to many (1:m) and many to many (n:m). Other elements of the ER-D are: Primary key (PK) which is an attribute of an entity that distinguishes it in a particular way. The PK is selected by the designer.



















3. RELATED WORK

Every day is more frequently that BD designers use applications to obtain SQL code from a RD because they offer different benefits that make this task easier and quickly, on the other hand the student's use it to understand better the process and for practice purposes.

Currently, there are different applications that translate a Relational Diagram to SQL. Such a diagram differs from ERD mainly because they add more details about data, like data type of the fields, and they are more used in another design approach, i.e., the relational model. So, until the investigated, there are no application generating SQL code directly from an ERD. Consequently, in this project we propose to make an extension to DIA to generate semi-complete SQL code from an ERD.

In Table 1 it can be seen a comparison between the applications that are the more related to the work we presented here, *DerSql*.

Table 1. Comparative table of applications related with automatic SQL code generation from DB design diagrams

Applications	Multi platform capability	Cost	ERD to SQL	Languages used	Diagram used
Rational Rose				?	Relational
MySQL Workbench	 			C#, C++, Objective C	Relational
DIASQL	 			Python	Relational
ERD PLUS	 			?	Relational
DERSQL				Python, XML(VDX)	ERD

4. DERSQL TOOL: DESIGN AND IMPLEMENTATION

So, this section describes and details the design, the components, the architecture, and operations of *DerSql*, our proposal to extend DIA.

Currently, our extension allows to have semi-complete SQL code from an ERD with arities 1 – 1 and 1 – n, and it is available for Windows operative system.

We say that the SQL code obtained is semi-complete because the ERD does not specify some elements about data, because of ERD abstractly represents entities and their attributes, that is to say, the attributes does not represent the data types, neither the Foreign Keys (FK).

Figure 1 shows the sequence diagram. In this diagram it can be seen the user and their interaction with DIA, that can be summarized as the following:

1. User takes an ERD designed in DIA.
2. In the menu of export select *DerSql*.
3. *DerSql* performs the conversion of the ERD to a tabular representation and the SQL code.

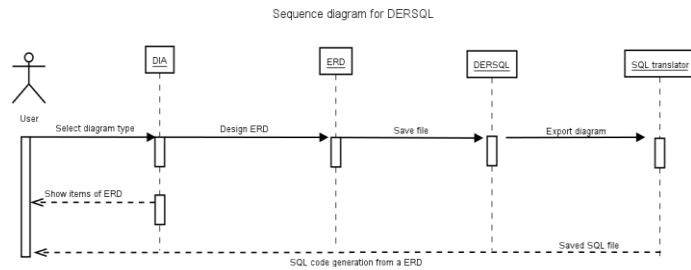


Figure 1. Sequence that user most perform to obtain tabular representation and SQL code from an ERD using DerSql in DIA

A description of the files, the architecture, the integration, and the interaction of the components is provided below.

BatDerSql.py. This Python 2 file is in charge to invoke another file that is called *DerSql.bat*.

DerSql.bat. DIA can be run from the command line. This allows certain DIA capabilities to be used in batch script file (Breit et al., 2007). With a sequence of commands in this file (batch programming) it was possible to export the *.dia* file into another file format, in this case we exported to a *.vdx* format and the file that manages the exportation is *vdx-export.c*.

File.vdx. This file is the result of the exportation of the *vdx-export.c* file. It is the XML representation and contains all the information and structure to represent ERD.

DerSql.py. This file was developed on python 3. It is in charge of extracting, reading and parsing important information from the *.vdx* file.

To read the *.vdx* file we choose and use a python’s API named *xml.etree.ElementTree* (ET in short).

The process performed in *DerSql.py* result in two files:

1. The tabular representation of the ERD designed, and its extension is (.txt). It contains a) for each set of entities, a table that contains their respective attributes; b) The identified PK; b) The creation of FK on their respective tables; and d) A short explication of the identified relations to explain why intermediate tables are not created.
2. The SQL code generated. This process can be seen on Figure 2.

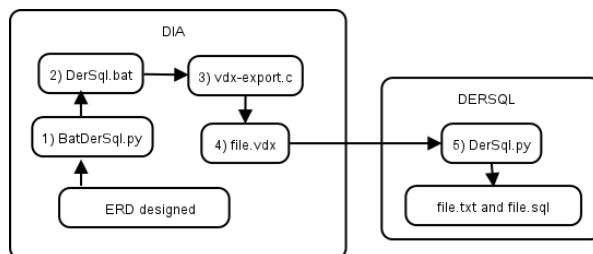


Figure 2. Architecture and interaction between the components of DerSql

The whole procedure and interaction between user and *DerSql* can be seen on Figure 3.

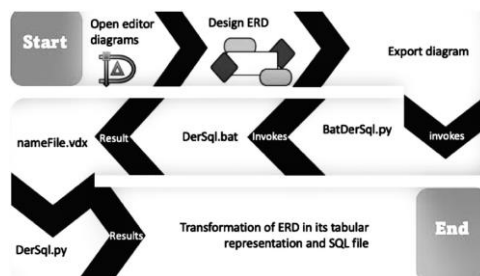


Figure 3. Operating flow and interaction between user and DerSql

5. EXAMPLES

We present here an example to illustrate the using of *DerSql*. To do so, we operate according to the steps described in Figure 4. The outcomes are shown in the next figures.

In Figure 4 on the left side, it is an ERD designed and represents three entities and their attributes. Every entity is connected to each other by a relationship. The arity of every relationship is one-to-many, and every entity has a PK attribute. On the right side it can be seen the tabular representation in a .txt file and a short explanation of the relationships and the SQL code generated. Every table created is the entity with their attributes and the FK created on their respective tables. The users the only thing that must change is dataType to the corresponding data type of the attribute and specify the length.

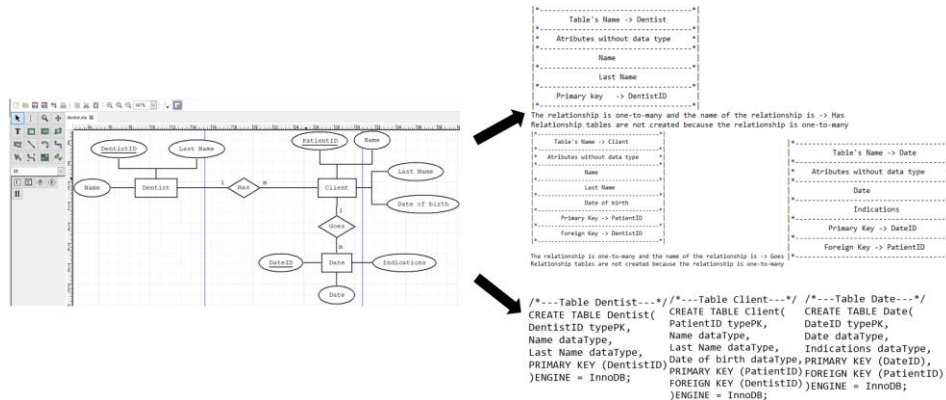


Figure 4. Results after applying DerSql are: the tabular representation and the SQL code from an ERD designed in Dia

6. TESTS AND RESULTS

We pursued a set of user tests with undergraduate students with skills in DB. They were asked to install DerSql on their computers and then to reproduce some ERD given by us. After the test, participants were required to fill in a questionnaire to register their user experience and to assess the obtained results when using DerSql.

The questions we divided in three sections and they are: a) usability; b) competitiveness and c) feedback and some of the questions we asked are: a) How was your experience using *DerSql*?; b) How do you consider the use of *DerSql*?; c) Do you think *DerSql* could have been useful in the database class?; c) Did you know the Diagrammer DIA?; d) Do you know any tool that allows you to do what *DerSql* does?

In Figure 5 we summarize the student's response, in the first graphic most of the user responses considered *DerSql* as very easy to use, we can deduce that students who does not know DIA find a little bit more difficult to use it. However, as can be seen in the second graphic, most of the user responses answered that their experience was satisfactory.

After running tests with a group of 20 users and every user replicated five different ERD, we consider that we have successful results because most of the students obtained the expected outcome after converting the DER to SQL by using *DerSql*. Additionally, their comments on the user experience were positives. The students commented that they would have liked to have *DerSql* in their DB courses at university. Finally, they stated that having these examples helps them to better understand the SQL sense and syntax when performing the translation from a DER to SQL code. The latter is an important point in helping to teach the design and implementation of DB and the Entity-Relationship Model.

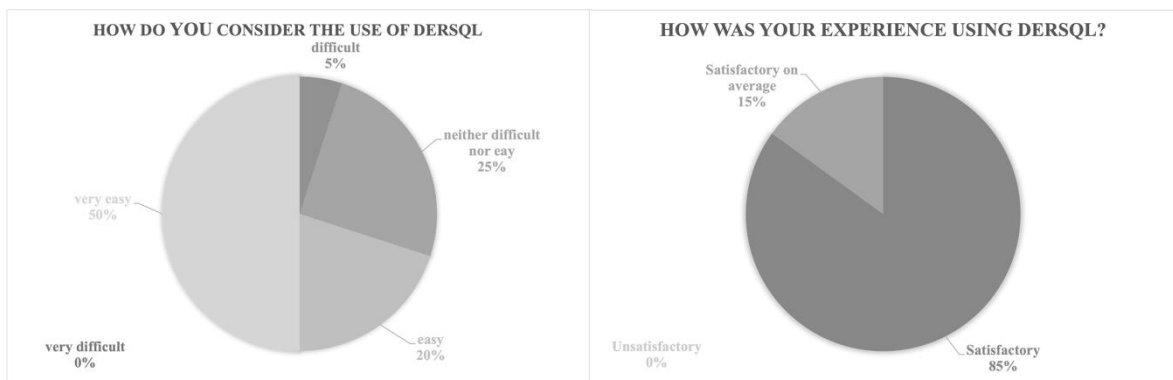


Figure 5. Graphics That Represents the User Experience after Using DerSql

7. CONCLUSION AND FUTURE WORK

In the development of applications, a pattern is followed to achieve an objective that ranges from design, implementation, and the execution. With *DerSql* we reduced time in creating source code from an ERD and allows to obtain the code with a lower margin of error. In addition to benefit software development, this project benefits instructors who teaches subjects related with relational databases because the students will have a better understanding in the process and transitions. *DerSql* allows to trace the entire process of designing and coding, i.e. from the generation of the ERD to the obtention of the SQL code that represents the data restrictions in an appropriate way with a noteworthy possibility: the explanation of the translation of the graphical representation to the tabular one, where the data model is applied.

Accordingly, to the user responses, they considered that *DerSql* could be a useful tool during the DB courses.

With this project we obtained a system integration as we integrated different programming languages, from batch programming to Python and XML. We also performed reengineering because we analyze the XML file, the functions in DIA source code to understand how it works. It was a hard task due the lack of sufficient information or documentation to comprehend its functioning, their scripts, or the existing extensions. Furthermore, we learnt how to run DIA since the source code using Meson (open-source build system). Despite these difficulties, we achieve a successful integration between different scripts.

For future work, there are some avenues to be endeavored: a) to expand the functionality of our tool to allow ERD with many to many cardinalities; b) to be able to read more elaborated ERD's and c) to make *DerSql* available to another operating systems.

ACKNOWLEDGEMENT

We are grateful to Cintia Ameyalli Solis Nuñez for her contribution to the design of some of the graphics to this paper, and to Mtra. Lucila Mercado Colin for her contribution of planning the user tests, and DTI/DCCD of UAM-C for supporting this work.

REFERENCES

- Bertone, R. and Thomas, P. (2011). *Introducción a Las Bases De Datos*. Pearson.
- Breit, K., et al, (2007). *DIA USER MANUAL*. [online] Available at: <http://dia-installer.de/doc/en/dia-manual.pdf>
- Sanchez, J. (2019). *Manual De Gestión De Bases De Datos. Modelo Entidad/Relación*. [online] Available at: <https://jorgesanchez.net/manuales/gbd/entidad-relacion.html>.