ES-SQL: Visually Querying Spreadsheets

Jácome Cunha*†, João Paulo Fernandes*‡, Jorge Mendes*, Rui Pereira*, and João Saraiva*
* HASLab/INESC TEC & Universidade do Minho, Portugal
† CIICESI, ESTGF, Instituto Politécnico do Porto, Portugal
‡ RELEASE, Universidade da Beira Interior, Portugal
{jacome,jpaulo,jorgemendes,ruipereira,jas}@di.uminho.pt

***Abstract*—This paper presents ES-SQL, an embedded tool for visually constructing queries over spreadsheets. This tool provides an expressive query environment which has knowledge on the business logic of spreadsheets, and by this knowledge it assists the user in defining the intended queries.**

I. Introduction

Spreadsheets are one of the most successful and widely used software systems targeted specially at end users. The uses of spreadsheets range from simple single user applications to large business oriented decision making calculations.

Although conceived to be simple, easy, visual, and human-friendly, the concrete uses of spreadsheets in the real world tend to evolve into large and complex data-centric software systems. In this scenario spreadsheets often grow bigger than thousands of lines per thousands of columns and it becomes difficult to extract, query, and reason about their data.

To simplify the spreadsheet querying process, we have proposed an Embedded Spreadsheet-Structured Query Language (ES-SQL) approach for end users [1], which relies on our previous work on model-driven spreadsheets where a concise model abstracts the structure and logic of a potentially large spreadsheet [2]. This abstraction allows queries to be expressed by names, instead of column letters, referencing entities.

ES-SQL, which itself has been built on top of *MDSheet* [2], provides the powerful and expressive information extraction setting of [3], [4], [5], while maintaining a simple to read, write, and understand querying system, all in the user's spreadsheet. Moreover, the visual query building environment is synchronized with the spreadsheet model, maintaining full consistency, after model/instance evolution, between these two.

II. Using ES-SQL

Before we present ES-SQL, let us introduce a running example to use throughout the paper. Figure 1 shows a spreadsheet model written in the *ClassSheet* language [6] containing information about the *Budget* of a research team.

	A	B	C	D	E	F	G
1	Budget		Year			···	
2			year=2005			···	
3	Category	Name	Qnty	Cost	Total	···	
4		name="abc"	qnty=0	cost=0	total=qnty*cost	···	
5	⋮	⋮	⋮	⋮	⋮	···	⋮
6							

Fig. 1. A Budget *ClassSheet* model

The first, and fourth author were funded by FCT: SFRH/BPD/73358/2010, BI3-2013_PTDC/EIA-CCO/116796/2010_UMINHO, respectively.

This **Budget** model contains a **Category** class (with a **Name** attribute), and a **Year** class (with a **Year** attribute), expanding vertically and horizontally, respectively (expressed by the ellipsis). An instance of each class gives us information on the **Quantity**, the **Cost**, and the **Total**, of a **Category** in a given **Year**.

Using ES-SQL, users can write queries in their familiar spreadsheet environment, without the need of learning textual (SQL-like) notation. This is achieved by guiding users in query construction, and is achieved using drop-down boxes to select attributes, filter conditions, and other querying conditions. This eliminates both syntactic and semantic errors.

Let us now address the following question under ES-SQL:

In a budget instance of the model in Figure 1, *What was the total per year, in decreasing order, from 2010 onwards?*

In ES-SQL, we display all the information from our original model-driven query language in a human-friendly way: along with the *ClassSheet* model and instance, the ES-SQL query is also shown in its own worksheet. In Figure 2 we show how to construct a query to answer our previous question.

	A	B	C	D	E	F	G	H	I
1	Attributes	✓	Formula	Sort		Unique rows		Run	
2	Budget					Limit rows	0		
3	Year								
4	year	✓				Conditions			
5	Category								
6	name					Attribute	Op	Value	
7	(Year,Category)					Year.year	>=	2010	-
8	qnty								
9	cost						+		
10	total	✓	Sum		Z↘A				

Fig. 2. ES-SQL representing the answer to the previous question

The steps to construct this embedded query are as follows:

1) Using the drop-down boxes under the *Selected* column (B), click on cell B4 and B10 to select the *Check Mark*, selecting both *year* and *total* to be used, respectively.
2) Under the *Formula* column, click on cell C10 and choose *Sum* from the drop-down box list.
3) Click on the "+" button to add a new condition row.
4) Select the *Year.year* attribute, and >= operation using the drop-down boxes in the new condition row. Afterwards, fill in *2010* in the value cell.
5) Click "Run".

III. Mapping ES-SQL to Plain SQL

This section presents the graphical to textual query translation in the ES-SQL system. Looking at Table I we can see

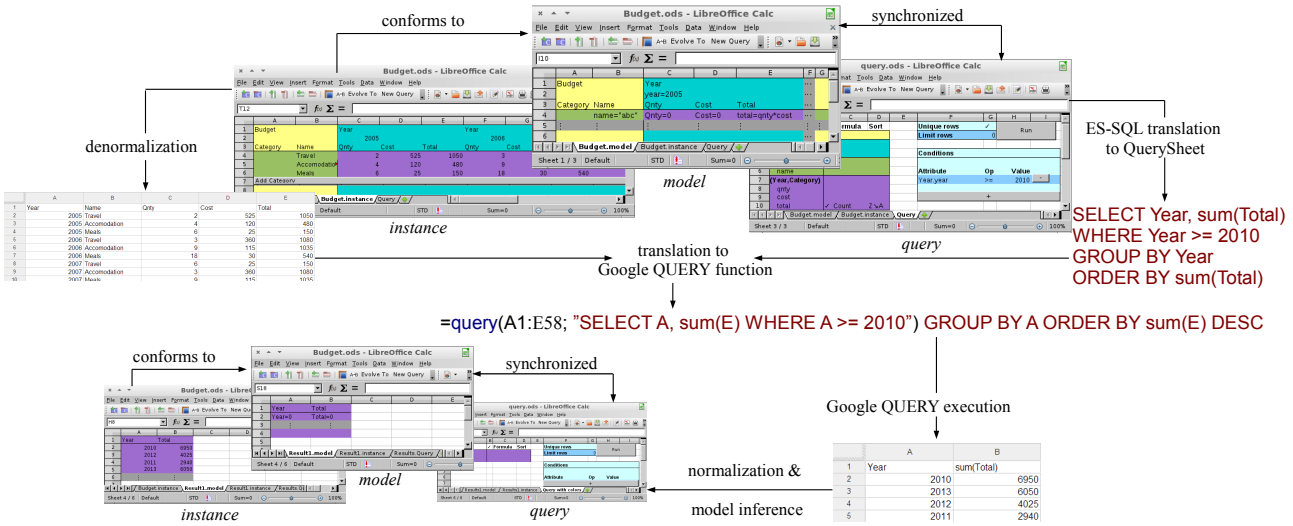


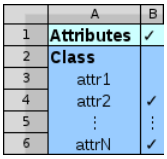
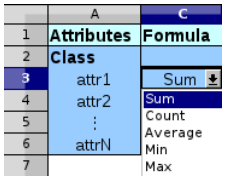
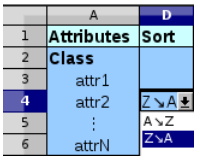
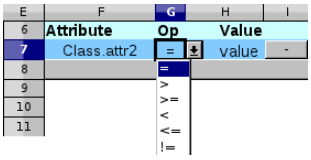
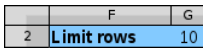
Fig. 3. Architecture of the embedded model-driven query system

the visual query on the left, with its translated textual SQL equivalent on the right. Looking at the right column, we can see bold text, which represents what part of the SQL clause we are generating, and italic text, representing the pseudocode for that translation. The remaining text is part of the query the system is currently translating. Our system automatically calculates where a *group by* is needed, a transformation which is applied to the textual translation before running the query.

IV. Architecture of ES-SQL

ES-SQL builds upon *QuerySheet* [4], and we follow here a similar approach to the one we followed in the past. Indeed, we translate the embedded language to our model-driven *QuerySheet* language, while the rest of the process conceptually remains the same. When the user clicks "Run", the visual language is translated to our model-driven language. Afterwards, the data (spreadsheet instance) is automatically denormalized, and using this data alongside the model-driven query, we translate it to Google's QUERY function using advanced compiler techniques (generalized top-down parsing). Finally, both the denormalized data and Google QUERY function are sent to be executed. The results are returned with a corresponding model/instance and a new embedded query sheet. This architecture can be seen in Figure 3.

TABLE I. Mapping the embedded query to standard SQL

Graphical	Textual
	**SELECT Class.attr2, ..., Class.attrN**
	*SELECT* *SWITCH C3* *Sum:* **Sum(Class.attr1)** *Count:* **Count(Class.attr1)** *Avg:* **Avg(Class.attr1)** *Min:* **Min(Class.attr1)** *Max:* **Max(Class.attr1)**
	*SELECT ...* *if(D4 == Z↘A)* **ORDER BY Class.attr2 DESC** *elsif (D4 == A↘Z)* **ORDER BY Class.attr2 ASC**
	*SELECT ...* **WHERE** *SWITCH C3* *=:* **Class.attr2 = value** *>:* **Class.attr2 > value** *>=:* **Class.attr2 >= value** *<:* **Class.attr2 < value** <=: **Class.attr2 <= value** *!=:* **Class.attr2 != value**
	*if (G1 == ✓)* **SELECT DISTINCT ...** *else* **SELECT ...**
	*SELECT ...* *if (G5 > 0)* **LIMIT G5**

References

[1] J. Cunha, J. P. Fernandes, J. Mendes, R. Pereira, and J. Saraiva, "Embedding model-driven spreadsheet queries in spreadsheet systems," in *VL/HCC'14*. IEEE, 2014, to appear.

[2] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva, "MDSheet: A framework for model-driven spreadsheet engineering," in *ICSE'12*. IEEE Press, 2012, pp. 1395–1398.

[3] J. Cunha, J. P. Fernandes, J. Mendes, R. Pereira, and J. Saraiva, "Querying model-driven spreadsheets," in *VL/HCC'13*. IEEE, 2013, pp. 83–86.

[4] O. Belo, J. Cunha, J. P. Fernandes, J. Mendes, R. Pereira, and J. Saraiva, "Querysheet: A bidirectional query environment for model-driven spreadsheets," in *VL/HCC'13*. IEEE, 2013, pp. 199–200.

[5] R. Pereira, "Querying for model-driven spreadsheets," Master's thesis, University of Minho, 2013.

[6] G. Engels and M. Erwig, "ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications," in *ASE'2005*. ACM, 2005, pp. 124–133.