QuerySheet: A Bidirectional Query Environment for Model-Driven Spreadsheets

Orlando Belo[*†], Jácome Cunha[*‡§], João Paulo Fernandes[*‡¶], Jorge Mendes[*‡], Rui Pereira[*‡], João Saraiva[*‡]
[*] Universidade do Minho, Portugal [†] Algoritmi R&D Centre, Portugal [‡] HASLab/INESC TEC, Portugal
[§] CIICESI, ESTGF, Instituto Politécnico do Porto, Portugal [¶] RELEASE, Universidade da Beira Interior, Portugal
{obelo,jacome,jpaulo,jorgemendes,ruipereira,jas}@di.uminho.pt

***Abstract*—This paper presents a tool, named QuerySheet, to query spreadsheets. We defined a language to write the queries, which resembles SQL, the language to query databases. This allows to write queries which are more related to the spreadsheet content than with current approaches.**

I. Introduction

Like most software artifacts, spreadsheets start as simple software systems and rapidly evolve into large and complex data-centric softwares. In such complex systems it is very important to have good support to manipulate and reason about data. Database systems use well-known techniques, namely the relational model, and language support, namely SQL, to query, extract and reason about their data. Unfortunately, spreadsheet systems do not offer such support to query their data!

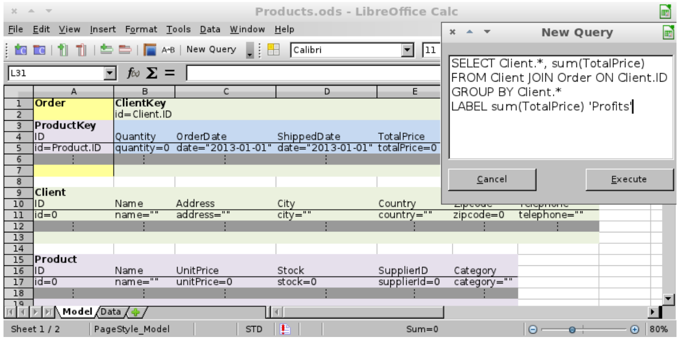
This paper presents QuerySheet: a tool that brings to spreadsheets the query database realm. The tool offers a query language, very similar to SQL, to query spreadsheets. This query language is based on spreadsheet models, namely ClassSheets [1], rather than on the spreadsheet data. By focusing on a simple, concise description of the spreadsheet data, rather than on a possibly large and complex spreadsheet data, we mimic the database approach: a database query writer usually reasons about the relational model of the database to express his/her queries, and not on understanding the large database. Such an approach has also the advantage of expressing queries using attribute names, and not by referring to spreadsheet areas and column letters as provided by Google and Microsoft approaches to query spreadsheets. Both systems also require that the spreadsheet data is represented in a single matrix, that is to say that the data has to be in (or transformed to!) a non-normalized representation. In QuerySheet this is performed automatically by using normalization/denormalization and model inference techniques [2].

II. QuerySheet

In order to present our spreadsheet query language, let us consider a spreadsheet storing information about products, clients and orders. The ClassSheet model defining the business logic of this spreadsheet is shown in Figure 1 (Model worksheet). Suppose that we would like to know:

- How much have we profited from each client?
- How much have we profited from USA clients (with its histogram)?

In a regular spreadsheet system it would be very difficult to extract this information from the spreadsheet data. Both Microsoft's query system and Google's QUERY function provide a basic form of expressing queries. However, they require users to denormalize the data (that is organized in three different tables), and to use column letters in the queries themselves, which makes their use complex and error-prone.



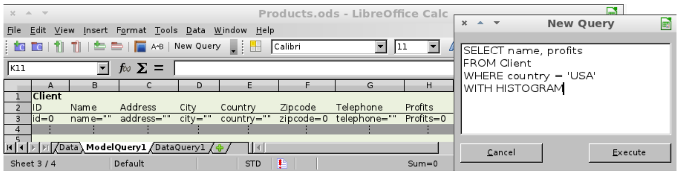
```
SELECT Client.*, sum(TotalPrice)
FROM Client JOIN Order ON Client.ID
GROUP BY Client.*
LABEL sum(TotalPrice) 'Profits'
```

	A	B	C	D	E		
1	Order	ClientKey					
2		id=Client.ID					
3	ProductKey						
4	ID	Quantity	OrderDate	ShippedDate	TotalPrice		
5	id=Product.ID	quantity=0	date="2013-01-01"	date="2013-01-01"	totalPrice=0		
6	⋮	⋮	⋮	⋮			
7							
8							
9	Client						
10	ID	Name	Address	City	Country	Zipcode	Telephone
11	id=0	name=""	address=""	city=""	country=""	zipcode=0	telephone=""
12	⋮	⋮	⋮	⋮	⋮	⋮	⋮
13							
14							
15	Product						
16	ID	Name	UnitPrice	Stock	SupplierID	Category	
17	id=0	name=""	unitPrice=0	stock=0	supplierId=0	category=""	
18	⋮	⋮	⋮	⋮	⋮	⋮	

Fig. 1. A model-driven spreadsheet representing orders, clients, and products.

In QuerySheet we express the query based on the ClassSheet model, and not on the spreadsheet data. The tool provides a *New Query* button, that opens a text box, where the query is defined. As we can see in Figure 1 the query (expressing the first question) looks very much like SQL: it uses the same keywords and syntactic structure. Moreover, the queries use ClassSheet labels to identify the entities involved.

When executing the query, the QuerySheet generates the result as a ClassSheet-driven spreadsheet. In fact, two new worksheets are added to the original spreadsheet: one containing the spreadsheet data that results from the query (DATAQUERY1), and the other (MODELQUERY1) contains the ClassSheet model, as shown in Figure 2.

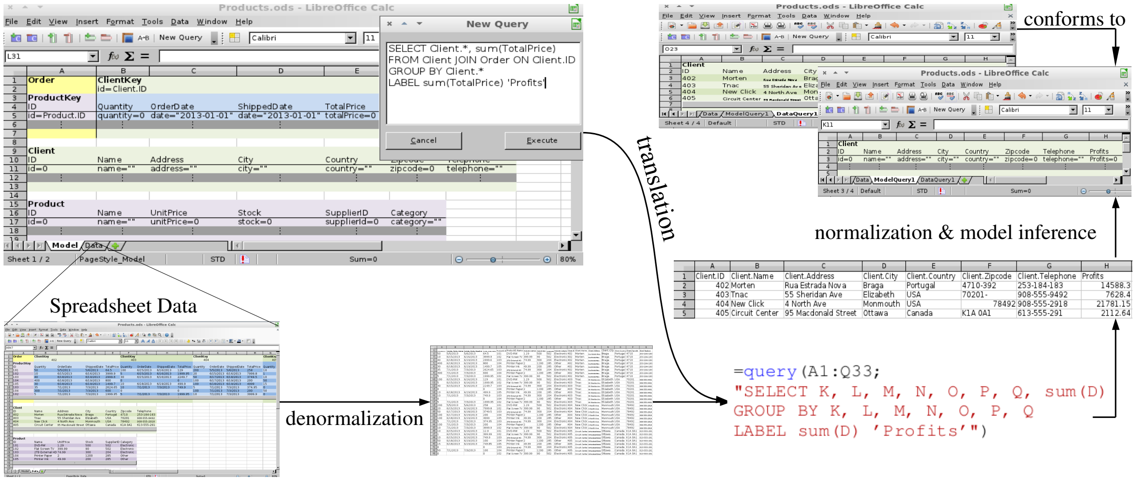


```
SELECT name, profits
FROM Client
WHERE country = 'USA'
WITH HISTOGRAM
```

	A	B	C	D	E	F	G	H
1	Client							
2	ID	Name	Address	City	Country	Zipcode	Telephone	Profits
3	id=0	name=""	address=""	city=""	country=""	zipcode=0	telephone=""	Profits=0
4	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
5								

Fig. 2. The model-driven spreadsheet produced after executing the query.

Figure 2 also shows the query to answer our second question. When executed, this query is applied to the results of the first one, that is to say, we can combine queries. In Figure 3 we show the (data) results of this second query.

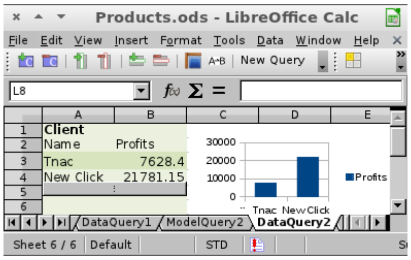


```
SELECT Client.*, sum(TotalPrice)
FROM Client JOIN Order ON Client.ID
GROUP BY Client.*
LABEL sum(TotalPrice) 'Profits'
```

	A	B	C	D	E	F	G	H
1	Client.ID	Client.Name	Client.Address	Client.City	Client.Country	Client.Zipcode	Client.Telephone	Profits
2	402	Morten	Rua Estrada Nova	Braga	Portugal	4710-392	253-184-183	14588.3
3	403	Tnac	55 Sheridan Ave	Elizabeth	USA	70201-	908-555-9492	7628.4
4	404	New Click	4 North Ave	Monmouth	USA	78492	908-555-2918	21781.15
5	405	Circuit Center	95 Macdonald Street	Ottawa	Canada	K1A 0A1	613-555-291	2112.64

```
=query(A1:Q33;
"SELECT K, L, M, N, O, P, Q, sum(D)
GROUP BY K, L, M, N, O, P, Q
LABEL sum(D) 'Profits'")
```

Fig. 4. The architecture of QUERYSHEET.



	A	B
1	Client	
2	Name	Profits
3	Tnac	7628.4
4	New Click	21781.15

Fig. 3. The spreadsheet (data) result after executing the second query.

QUERYSHEET is implemented as part of MDSheet [3]. Because implementing a powerful and efficient query engine is very complex, we express the semantics of our query language using Google's QUERY function. To make the language more expressive we added extra functionality such as a *JOIN* clause. Our query language allows more humanized queries to be written with attribute names, not column letters, and supporting ClassSheets. To do so, we implemented a translator from our query language to the *Visualization API Query Language* defined by Google for use in the QUERY function. We have also implemented the denormalization process needed to automatically transform the data in our environment to the needed tabular denormalized format.

Illustrated in Figure 4, is QUERYSHEET's architecture. The top left part shows our spreadsheet model/instance of our previous example. The QUERYSHEET language is based on the SQL language, while allowing some of the QUERY function's clauses such as LIMIT and LABEL. The language supports the JOIN clause, ClassSheet attribute selection, and multiple ways of naming the attribute to avoid conflicts. More information about the syntax can be found in [4].

As stated before, we need to denormalize the data into a single table (bottom-center of the figure). After we obtain the data from our model-driven environment, we begin placing the data into its denormalized state carefully grouping the correct row of information, while dealing with the *problems* caused by denormalized data querying such as *derived data* and *attribute aggregation*. In fact, these are well-known and well-studied problems in the database realm [5]. Our query is then translated to its exact counterpart for the QUERY function, automatically calculating the *range* input, and substituting the attribute names to their counterpart column letters using a lookup function, as shown in the bottom-right of the figure. Finally, we use the QUERY function to obtain the results expected. Afterwards we apply a technique introduced in previous work [2] to automatically infer a ClassSheet from the resulting spreadsheet data, as shown in the top-right of the figure.

The model-driven query language and the techniques proposed in [4] were the building blocks used when developing QUERYSHEET. Such techniques were then extended to support a bidirectional query environment where end users can update the original and target spreadsheet, and all software artifacts are automatically synchronized. This is due to the result and original data being a view over the now denormalized data, much like what happens with views in database systems.

ACKNOWLEDGMENT

This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within projects FCOMP-01-0124-FEDER-010048and FCOMP-01-0124-FEDER-020532. Some authors were funded by FCT: SFRH/BPD/73358/2010, BI3-2012PTDC/EIA-CCO/108613/2008, BI2-2012PTDC/EIA-CCO/1086613/2008.

REFERENCES

[1] G. Engels and M. Erwig, "ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications," in *Proc. of the 20th IEEE/ACM Int. Conf. on Aut. Sof. Eng.* ACM, 2005, pp. 124–133.

[2] J. Cunha, M. Erwig, and J. Saraiva, "Automatically inferring classsheet models from spreadsheets," in *IEEE Symp. on Visual Languages and Human-Centric Computing*. IEEE CS, 2010, pp. 93–100.

[3] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva, "MDSheet: A framework for model-driven spreadsheet engineering," in *Proc. of the 34rd Int. Conf. on Software Engineering*. ACM, 2012, pp. 1412–1415.

[4] J. Cunha, J. Mendes, J. P. Fernandes, R. Pereira, and J. Saraiva, "Querying model-driven spreadsheets," in *IEEE Symp. on Visual Lang. and Human-Centric Comp.* IEEE CS, 2013, to appear.

[5] D. Maier, *The Theory of Relational Databases*. Computer Science Press, 1983.