# User-Friendly Spreadsheet Querying: An Empirical Study*

Rui Pereira, João Saraiva
HASLab/INESC TEC & Minho University, Portugal
{ruipereira,jas}@di.uminho.pt

Jácome Cunha
Universidade Nova de Lisboa Portugal
jacome@fct.unl.pt

João Paulo Fernandes
RELEASE, Universidade da Beira Interior, Portugal
jpf@di.ubi.pt

## ABSTRACT

Spreadsheets are nowadays used in a variety of contexts, including in in manipulatin large and complex data. This data is stored in a large unstructured matrix, which is hard to understand and to manipulate. Recent research has been done to manipulate and query such unstructured data, namely by proposing different query approaches to spreadsheets. In this paper we present an empirical study evaluating three recent query approaches to spreadsheets assessing their usage to query spreadsheets. The results of our study show that the end-users' productivity increases when using visual, model-driven queries are used.

## CCS Concepts

•**Human-centered computing** → **Empirical studies in HCI;** •**Information systems** → *Query representation;*

## Keywords

Spreadsheets; Querying; Empirical Studies

## 1. INTRODUCTION

Spreadsheets are the programming language of choice for non-professional software programmers, like teachers, managers, engineers, accountants, etc, (often called "end-user" programmers) who write programs mainly for their own use. In fact, spreadsheet systems are not only used to implement spreadsheet specific tasks (for example, to implement companies' budgets), but also to manipulate large and complex data. Spreadsheets can be seen as the *"poor man's database programming language"*, used to store data in a simple and visual way, but not having the specific features of a database system. It is surprising to see that 56% of spreadsheets in the large EUSES corpus do not contain formulas, only data! In order to efficiently manipulate such data, spreadsheets need to provide the data normalization mechanisms and a powerful query language, as databases do. However, it may be hard and time consuming for inexperienced users to pose structured queries that satisfy their query intent, since the users are required to be proficient in writing the query languages and have a thorough understanding of how query mechanisms work. On the other hand, they may encounter comprehension difficulties, formulation problems, and incorrect queries. There has been several research work studying the usability of query systems mainly in the context of databases [1, 2]. To the best of our knowledge there has been no work on studying how queries make spreadhseet users more productive, and which query framework (*textual/visual*) they find easier to understand.

In this paper we assess the usability of three approaches that support querying of spreadsheet data: Google' QUERY function [3], and the two model-driven querying systems: the textual QuerySheet [4, 5] and the visual Graphical-QuerySheet [6]. The three query systems provide powerful abstractions to query spreadsheets, and, as a consequence, it is important to assess how end users understand and take advantage of them. We conduct an empirical study with spreadsheet users in order to compare end user productivity, and the framework understandability and intuitiveness, when using the three query systems.

## 2. SPREADSHEET QUERYING

This section presents our running example: a spreadsheet used to store information relative to the budget of a research group (Fig. 1). This spreadsheet contains information about the Category of budget used and the Year. The relationship between Category and Year provides us with the information on the Quantity, Cost, and the Total Costs (defined by spreadsheet formulas), per year per category.

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | **Budget** | | **Year** | | | **Year** | | | **Year** |
| 2 | | | 2005 | | | 2006 | | | 20( |
| 3 | **Category** | Name | Qnty | Cost | Total | Qnty | Cost | Total | Qnty |
| 4 | | Travel | 2 | 525 | 1050 | 3 | 360 | 1080 | |
| 5 | | Accomodation | 4 | 120 | 480 | 9 | 115 | 1035 | |
| 6 | | Meals | 6 | 25 | 150 | 18 | 30 | 540 | |

Figure 1: Spreadsheet Example

Using this spreadsheet, we will answer the next question: *In what category did we have the most expenses (last 5 years)?*

*This work has been supported by Fundação para a Ciência e a Tecnologia, under grant SFRH/BPD/112733/2015.



*SAC 2016,April 04-08, 2016, Pisa, Italy*



DOI: http://dx.doi.org/10.1145/2851613.2851910

Next, we introduce the three systems to query spreadsheets

**Google QUERY function:** Google provides a querying function called QUERY. This QUERY function performs a query, using a SQL-like syntax, over an array of values such as the Google Docs spreadsheets, in which the function is built in. Google's QUERY function (GQF) is a two argument function, consisting of a range and query string. The range argument is used to state the range of the data cells to be queried, for example A1:Q13 in our spreadsheet. The query string is the SQL-like query written by the user.

While the GQF is a powerful query function, it still has some flaws. To run this function, the user needs to represent his/her spreadsheet information in a single table, with each attribute represented in each column (in other words, with headers). This means that someone, who has their spreadsheet divided into various entities with or without relations, would first need to manually denormalize their data (as shown in Fig. 2). Such a process is difficult enough for professionals, let alone for end users.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Year | Name | Qnty | Cost | Total |
| 2 | 2005 | Travel | 2 | 525 | 1050 |
| 3 | 2005 | Accomodation | 4 | 120 | 480 |
| 4 | 2005 | Meals | 6 | 25 | 150 |
| 5 | 2006 | Travel | 3 | 360 | 1080 |
| 6 | 2006 | Accomodation | 9 | 115 | 1035 |
| 7 | 2006 | Meals | 18 | 30 | 540 |
| 8 | 2007 | Travel | 6 | 25 | 150 |
| 9 | 2007 | Accomodation | 3 | 360 | 1080 |
| 10 | 2007 | Meals | 9 | 115 | 1035 |

Figure 2: Spreadsheet example denormalized

In QUERY, the query writer must use column letters in the query, instead of column names/labels as is normal in database querying. Regardless, the query engine is very efficient, being able to handle very big spreadsheets. To answer our previous question, we would have to write in a spreadsheet cell, the following query function:

```
=query(A1:E58;"SELECT B,sum(E) WHERE A>=2010 GROUP BY B
                              ORDER BY sum(E) DESC LIMIT 1")
```

**QuerySheet system:** To overcome the issues identified with the GQF, researchers turned to model-driven engineering methodologies to design a query language and system for spreadsheets. In this case model-driven spreadsheet models were used, specifically ClassSheets [7]: a high-level and object-oriented formalism, using the notion of classes and attributes, to express business logic spreadsheet data. Using ClassSheets, one can define the business logic of spreadsheet data in a concise and abstract manner. This results in users being able to understand, evolve, and maintain complex spreadsheets by just analyzing the ClassSheet models, avoiding the need to look at large and complex data [8]. Indeed, as shown in [9], users need a bridge between spreadsheet data and the real world.

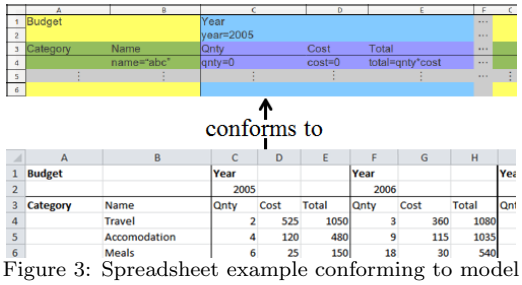



Figure 3: Spreadsheet example conforming to model

Fig. 3 presents a ClassSheet model and conforming instance for our running example. In this ClassSheet model, a Budget has a Category (with a Name attribute) and Year class (with a Year attribute), expanding vertically and horizontally, respectively. The joining of these two gives us a Quantity, Cost, and Total of a Category in a given Year, each with their own default values.

Using this spreadsheet model concept, we designed a querying language based on the attributes/labels in classes, as done in the database realm when using attribute names from tables. This querying system was named QuerySheet is built on top of GQF: it automatically denormalizes the spreadsheet models, translates the model-driven query to the query function counterpart and sends both translated function and denormalized data to be executed on Google's system. In QuerySheet, by just look at the (concise) model, it is very simple to answer our question:

```
SELECT Name, sum(Total) WHERE Year >= 2010
       GROUP BY Name ORDER BY sum(Total) DESC LIMIT 1
```

**Graphical-QuerySheet System:** Because end-users are not familiar with SQL syntax (and textual programming, in general), we developed a graphical user interface for visual query construction [6]. This visual interface, named Graphical-QuerySheet, hides syntax peculiarities and lets users to choose attributes based on the spreadsheet's model.

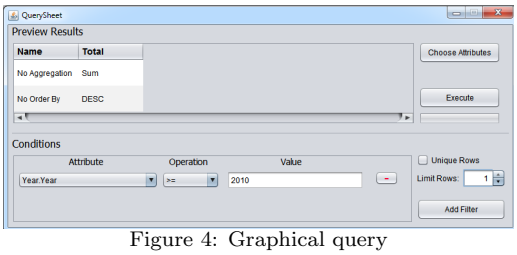


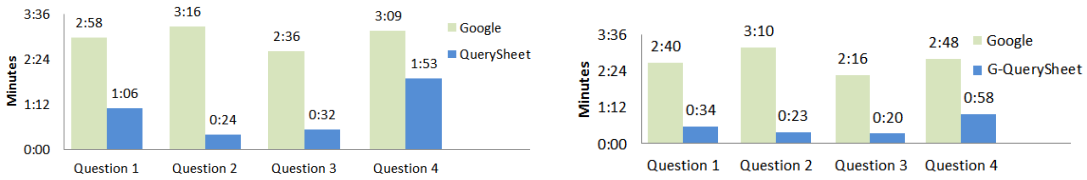

Figure 4: Graphical query

Fig. 4 shows Graphical-QuerySheet where we can easily answer the running question: Click on Choose Attributes and check Name and Total; Click on the aggregation combo box (it is visible when using the tool) under Total and choose Sum; Click on the order by combo box under Total and choose DESC; Click on Add Filter; Select the Year.Year attribute and greater or equal to operation using the combo boxes, and fill in 2010 in the text box; Finaly, click Execute.

## 3. EMPIRICAL STUDY

To assess the three querying systems in practice, we executed an empirical study with 14 users, obtaining and recording the results of their experiences and productivity. All participants were male, between ages 19-28, with background in computer sciences/informatics. Their knowledge of SQL also varied between no/little knowledge to very experienced users. All have previously worked with spreadsheets, with different levels of experience. The 14 participants were randomly divided into two groups. One was to test GQF vs. QuerySheet, and the other to test GQF vs. Graphical-QuerySheet. The study was done with one participant at a time, in a think-aloud session. Doing this allowed us to see each participant using the systems and learn the difficulties they were facing. These results are shown in Figure 5.

For the study, we used a real-life spreadsheet obtained from our hometown food bank. This spreadsheet represents the




(a) GQF vs. QuerySheet (b) GQF vs. Graphical-QuerySheet

Figure 5: Result comparison

distribution of basic products to specific institutions: it contains 85 institutions, with 14 types of products, and over 1190 lines of unique information. We denormalized the spreadsheet data for the participants to use with Google's QUERY function, and prepared the spreadsheet model and conforming instance in the model-driven environment.

In the study, we asked participants to implement queries to answer the following four questions regarding the information present in the distributions spreadsheet: *What is the total distributed for each product?*, *What is the total stock? What are the names of each institution without repetitions? Which were the products with more than 500 units distributed, and which institution were they delivered to?*

In answering each question, the participants had to implement a query using both systems (either GQF vs. QuerySheet or GQF vs. Graphical-QuerySheet); alternating between the starting systems (the initial starting system was chosen by each participant). We recorded the time answering the questions and after they were asked to answer a short questionnaire to choose which system they felt was more: Intuitive, Faster (constructing the queries), Easier (constructing the queries), and Understandable (easy to interpret the queries).

## 3.1 Results

Fig. 5a and Fig. 5b can be interpreted as follows: The Y-Axis represents the average number of minutes the participants took to answer the questions. The X-Axis represents the question the participants answered. The green bars represent the GQF, and the blue bars represent the QuerySheet and Graphical-QuerySheet system respectively.

For easy referencing, we will refer to the first study group, Google QUERY function vs. QuerySheet, as Group A, and for the second study group, Google QUERY function vs. Graphical-QuerySheet, as Group B.

As we can see in both groups, the participants spent substantially less time to construct the queries, and in turn were more productive, using the model-driven approach (both QuerySheet and Graphical-QuerySheet). In Group A, participants spent 62.8% to 87.8% less time, averaging to an overall of 67.5%. Looking at the graphical querying approach, in Group B, the participants spent 65.5% to 87.9% less time, averaging to an overall 79.4%. Almost all chose the model-driven querying approaches, in terms of the four previously mentioned points (Intuitive, Faster, Easier, and Understandable). In Group A, 111 out of 112 (4 points * 4 questions * 7 participants) chose QuerySheet. While in Group B 104 out of 112 chose Graphical-QuerySheet, providing us with interesting information, allowing us to detect some of the drawbacks of the graphical system, which will be explained further on.

In general the query error rates were low. There were 9 (out of 11) and 6 (out of 7) errors, Group A and B respectively, using Google's QUERY function. These errors varied between incorrect column letters chosen, bad query construction and incorrect ranges.

# 4. CONCLUSION

In this paper, we compared one visual and two textual DSL for querying spreadsheets, two of which are model-driven based. We have presented an empirical study with real users where they were asked to perform similar tasks in the three languages. Overall, the model-driven querying approaches have proven themselves to be much more efficient and easier to use than their counter part data oriented one.

# 5. REFERENCES

[1] J. Fan, G. Li, and L. Zhou, "Interactive sql query suggestion: Making databases user-friendly," in *Int. Conf. Data Eng. (ICDE)*, April 2011, pp. 351–362.

[2] H. Lu, H. C. Chan, and K. K. Wei, "A survey on usage of sql," *ACM SIGMOD Record*, vol. 22, no. 4, pp. 60–65, 1993.

[3] Google, "Google query function," https://developers.google.com/chart/interactive/docs/querylanguage, September 2015.

[4] 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.

[5] J. Cunha, J. Fernandes, J. Mendes, R. Pereira, and J. Saraiva, "Design and implementation of queries for model-driven spreadsheets," in *CEFP School*, ser. LNCS. Springer, 2015, vol. 8606.

[6] J. Cunha, J. P. Fernandes, R. Pereira, and J. Saraiva, "Graphical querying of model-driven spreadsheets," in *Int. Conf. on Human-Computer Intercation*, ser. LNCS, vol. 8521. Springer, 2014, pp. 419–430.

[7] G. Engels and M. Erwig, "Classsheets: automatic generation of spreadsheet applications from object-oriented specifications," in *ASE'05*. ACM, 2005, pp. 124–133.

[8] J. Cunha, J. Fernandes, J. Mendes, and J. Saraiva, "Embedding, evolution, and validation of model-driven spreadsheets," *Software Engineering, IEEE Trans. on*, vol. 41, no. 3, pp. 241–263, March 2015.

[9] B. Kankuzi and J. Sajaniemi, "An empirical study of spreadsheet authors' mental models in explaining and debugging tasks," in *VL/HCC'13*. IEEE, 2013, pp. 15–18.