

Translating SQL to Spreadsheet: A Survey

Tasnim T. Hajiwala

Department of Computer Engineering
VPKBIET
Baramati
tasnim.kayamkhani@gmail.com

Prof. Santosh A. Shinde

Department of Computer Engineering
VPKBIET
Baramati
Meetsan.shinde@gmail.com

Abstract—Spreadsheets are the most popular and conventionally databases in use today. Since Spreadsheets are visual and expression based languages, research into the features of spreadsheets is therefore a highly relevant topic to study. Spreadsheet can be viewed as a Relation Database which contains a sheet and its corresponding information in terms of rows, while in RDBMS each table or say relation also represents its contained information in terms of rows. Each row represents a record which belongs to one or more relation. Spreadsheets uses different formulae to extract required information but it need expert knowledge about the tool and its usage. One can extend the usage of Spreadsheet in any direction as it provides great flexibility in terms of data storage and dependency of stored data. We surveyed some of research which took great attention over Spreadsheets and its applicability in different functional cases, such as Data Visualization, SQL Engines and many more. Our survey focuses on QUERY SHEET, ES-SQL, MDSHEET and PrediCalc [3], [5], [4], [8]. These different researches are motivations to our survey and attraction in Spreadsheets and its functional extensibility.

Keywords- Spreadsheet, Relation Database SQL Engines, QUERY SHEET, ES-SQL, MDSHEET, PrediCalc

I. INTRODUCTION

The huge impact of DBMS attracted the business managements to keep their tracks Relational Databases because of its consistency; still we believe much of the community's distaste, however, stems from inspection of existing implementations of the tools which are commonly used to keep data like Spreadsheets; these seem deeply flawed to us. The question remains, then: is the spreadsheet metaphor any good? The true measure of is the problem domain it addresses. It's already well-known that modern spreadsheets are well-suited for standalone business decision making and financial modeling. The work decided to define a mini-language to see if the spreadsheets might be a more general instrument given the right implementation [1].

This analysis seems to be in stark opposite to some of the claims of functional programming advocates, for example, that functional programs are much reliable than, for example, imperative programs, and contain fewer bugs. However, a closer examination states that the increased reliability of functional programs is achieved, at least to some extents, through a cleaner language design, offering powerful abstractions, such as higher level functions, and through sophisticated type systems that help to detect programming bugs early. [2]

A research paper presents QUERY SHEET: a tool that proposes to spreadsheets the query database realm. The tool offers a querying functionality, very similar to SQL, to query spreadsheets. This query language is based on spreadsheet, namely Class- Sheets [1], rather than on the spreadsheet data. By relying on a simple, concise metadata of the spreadsheet data, rather than on a possibly huge and complex spreadsheet data, the paper duplicates the database approach: a database programmer usually reasons about the relational model of the database to code his/her queries, and not on understanding the

huge database. Such an approach has also the benefit of writing queries using attribute names, and not by referring to spreadsheet cells and column letters as provided by Google and Microsoft methods to query spreadsheets. Both approaches also need that the spreadsheet data is represented in a matrix format, that is to say that the data has to be in (or transformed to!) a non-normalized representation. In QUERY SHEET this is executed automatically by using normalization/de-normalization and model inference methodologies [3].

As programming languages, spreadsheets lack the support provided by recent programming environments, like for example, higher abstractions and strong type and modular architectures. As a result, they are vulnerable to errors. In order to increase end-users productivity, several approaches have been proposed, which let the end users to safely and correctly modify spreadsheets, like, for example, the use of spreadsheet templates [2], ClassSheets [3], [4], and the inclusion of visual objects to benefit editing assistance in spreadsheets. All these systems propose a form of end user model-driven software platforms: a spreadsheet business model is defined, from which a custom spreadsheet tool is then generated guaranteeing the consistency of the spreadsheet data with the underlying architecture[4].

Spreadsheet approach has proven to be a long success stories, both in the context of the programming of simple applications for personal use as well as to support business model decisions within large corporate organizations.

A fascinating contribution to the great success of spreadsheets is appreciated to their multi-purpose-ness which can be observed, for example, by the fact that many spreadsheets are actually used as data storage systems or databases [1], in the sense that no formulas are defined in them. [5]

Spreadsheets are one of the most appreciated and widely used software systems targeted towards the end users. The uses of spreadsheets range from simple standalone applications to large business oriented decision making calculations. Although conceived to be simple, easy, visual, and human friendly, the basic uses of spreadsheets in the actual world tend to begin into large and complex data-centric software tools. In this scenario spreadsheets often grow larger than thousands of lines per thousands of columns and it became difficult to extract, query, and reason about their stored data. To simplify the spreadsheet querying approach, authors have proposed an Embedded Spreadsheet-Structured Query Language (ES-SQL) approach for end users [1], which provide their previous work on model-driven spreadsheets where a precise model abstracts the structure and logic of a potentially large spreadsheet [2]. This abstraction allows queries to be written by names, instead of column letters, referencing entities [6].

II. DATA SOURCES

There are various ways to collect or generate the Spreadsheet data. In literature the authors tend to generate and working example transaction data through script execution, which automatically generates the required data in Spreadsheets.

A. Transaction Dataset

Different business activity needs to be consistent in their day-to-day budget and resources; the database they generate is stored in Spreadsheets in the form of rows and columns. We consider the columns as attributes and rows as tuples.

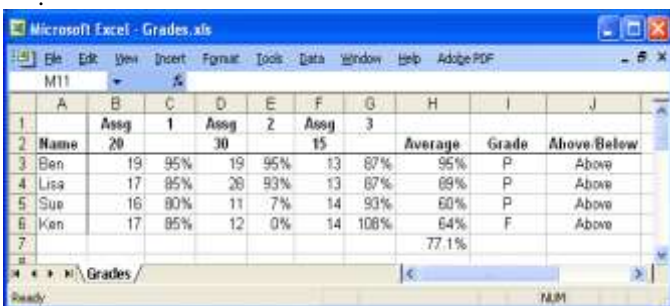


Fig. I Example of Spreadsheet

B. Management Datasets

Various government and non-government offices tend to store their data in Spreadsheet format for future use. This data can be a source of Relation Database which can be used to demonstrate the research in literature. Each of these data sources is useful in visualizing the extensibility of Spreadsheets.

III. RELATED WORK

A. QUERY SHEET

QUERY SHEET is designed as part of MDSheet [4]. Because developing a powerful and effective query engine is very tricky, the research expresses the semantics of their query language using Google's QUERY model. To make the

language more powerful and user friendly, authors added extra features such as a JOIN clause. Their query language provides more human readable queries to be written with attribute names, not column letters, and supporting ClassSheets. To do so, authors designed a translator from their query language to the Visualization API Query Language defined by Google for use in the QUERY model. They have also used the de-normalization process needed to automatically re-structure the data in their environment to the allowed tabular de-normalized format. Illustrated in Figure 2, is QUERY SHEET's architecture. The top left part shows their spreadsheet model/instance of their previous example. The QUERY SHEET 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 models of naming the attribute to eliminate conflicts[3].

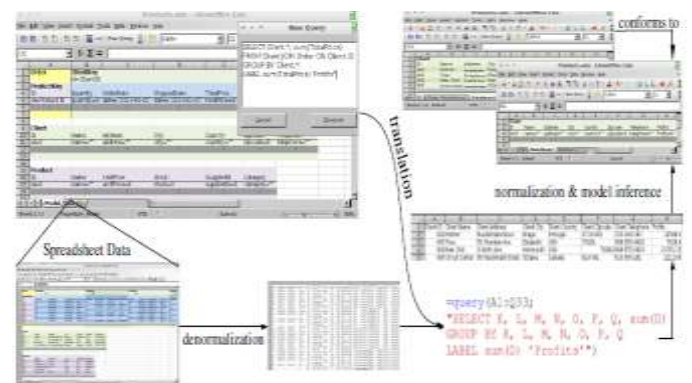


Fig. II The Architecture of QUERY SHEET

B. ES-SQL

In ES-SQL, users can write queries in their known spreadsheet environment, without the need of learning textual (SQL-like) notation. This is brought in by providing users in query writing, and is achieved using drop-down lists to select attributes, filter conditions, and other querying designs. This avoids both syntactic and semantic false rate. Let us now see the following question under ES-SQL: In a data represented as in Figure 1 What was the total per annum, in decreasing order, from 2010 onwards? In ES-SQL, the work shows all the information from their original model-driven query language in a human-friendly way: along with the ClassSheet model and current state, the ES-SQL query is also displayed in its own worksheet. In Figure 3 show how to write a query to answer their previous question[6].

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

Fig. III ES-SQL representing the answer generated

C. ClassSheet

ClassSheets [4] are a high-level, object-oriented program to specify the business logic of spreadsheets. Class-Sheets provide end users to express business objects structures within a spreadsheet using UML concepts. In fact in [7] the paper has proposed a method to automatically map ClassSheets to UML. Using the ClassSheets model, it is possible to define spreadsheet relations and to give them names, to define labels for the table's columns, to specify the types of the values such columns may contain and also the way the table expands (e.g., horizontally or vertically).

Besides textual (and formal) notations, ClassSheets also have a visual representation which very much resembles spreadsheets [8]. Authors have embedded such visual model representation that mimics the well-known embedding of a domain specific language in a general purpose one. Like in such embedding's, the paper inherit all the needful features of the host language: in their case, the powerful interactive interface offered by the (host) spreadsheet system. This approach has two key benefits: first, the paper does not have to build and maintain a complex interactive tool. Second, the system provides ClassSheet model developers the programming environment they are used to: a spreadsheet environment

	A	B	C		A	B	C
1	Planes			1	Planes		
2	N-Number	Model	Name	2	N-Number	Model	Name
3	N2342	B 747	Magalhães	3	n-number="N" model="B" name="Magalhães"		
4	N341	B777	Cabral	4	!	!	!

(a) Planes table.

(b) Planes ClassSheet model.

Fig. IV ClassSheet example

D. Predicalc

The research prototype has proven the feasibility of their methodologies on spreadsheets containing thousands of cells and hundreds of constraints. In fact, the limiting factor is not the underlying algorithms, but is instead the GUI, which is designed in a naive way and does not scale enough. One can, of course, analyze examples where a small number of interconnected constraints will result in an unacceptably slow response time for the spreadsheet; however, simple examples with little interconnection provides for large numbers of constraints to be handled very quickly. Their prototype uses a naive implementation of Existential W-entailment; however, authors have found significantly better algorithms that scale with the amount of inconsistency; details will be provided in a forthcoming paper. An unnoticed question is how to incrementally maintain materialized Existential W-entailment consequences as the cell values are modified. Also, experiments proves that unary relations do not scale particularly well, and so authors extended their model to allow for n-ary cells as well as unary ones in future versions of their spreadsheet.

IV. APPLICATIONS

Spreadsheets provide various ways to extend its applicability in different research and business needs. We have mentioned some its application in research and business fields.

1. Data Management: Logical spreadsheets provide the entry and editing of symbolic data governed by symbolic

constraints. 'Correct on capture' data entry tools and resource management systems are examples of this capability. The constraint 'only senior managers can reserve the third floor conference room' is an example of this. Since rules and policies can be expressed using logic, a logical spreadsheet can be thought of as a simple kind of computational law system (Love & Genesereth, 2005).

2. Interactive documents: Systems can provide 'interactive answers' to end users, e.g., simulations, which allow a user to analyze by varying certain inputs while the system automatically propagates the consequences of those variations. Let us, for example, a student learning how lenses refract light by experimenting with various lens shapes. Or consider a spreadsheet used by an insurance agent to determine if a client is eligible for a specific kind of insurance. The rule that 'insurance applicants who make at least \$60,000 and are under 50 years old are approved' is an example of this. Essentially, an interactive document provides one to perform the 'what if' analyses that spreadsheets are famous for, although there need not be a distinction between the cells used as input parameters and the cells used to output results.

3. Design and Configuration: Configuration tools are good examples of the use of logical Spreadsheets. Consider, for example, a configuration tools to help consumers design their own cars, which might have the limitations 'if the car's exterior color is blue, then the car interior color may be gray, tan or black'. Or consider a student preparing his course schedule, which might have the constraint 'students must take at least two math courses to graduate'.

Sr. No	Authors	Title	Year	Methodology
1	Orlando Belo, J'acome Cunha	<i>QuerySheet: A Bidirectional Query Environment for Model-Driven Spreadsheets</i>	2013	Based on the SQL language, while allowing QUERY function's clauses such as LIMIT and LABEL
2	Zhen Hai, Kuiyu Chang	<i>Querying Model-Driven Spreadsheets</i>	2013	Model-driven query language in a human-friendly way: along with the ClassSheet
3	J J'acome Cunha, Jo'ao Paulo Fernandes	<i>MDSheet: A Framework for Model-Driven Spreadsheet Engineering</i>	2012	It is possible to define spreadsheet relations and to give them names, to define labels for the table's columns
4	M. Kassoff, L.-M. Zen, A. Garg	<i>PredicCalc: a logical spreadsheet management system</i>	2005	Extended their model to allow for n-ary cells as well as unary ones

Table.I Methodologies used in Literature

V. CONCLUSION

In this survey paper we have studied some fascinating techniques for extensibility and applicability for Spreadsheets. The literature researches are quiet attracted towards the Spreadsheets, specially its storage structure and capability of managing the data in terms of records. The formulae and expressions provided by Spreadsheet are also contains great flexibility towards its extensibility. Basically Relational Database Management has a very resembling data storage structure to the Spreadsheet, so eventually the Spreadsheet is pulled over Structured Query Language (SQL) and its implementation.

VI. ACKNOWLEDGEMENT

I express great many thanks to Prof.Santosh A. Shinde for this great effort of supervising and leading me, to accomplish this fine work. To college and department staff, they were a great source of support and encouragement. To my friends and family, for their warm, kind encourages and loves. To every person who gave me something too light along my pathway. I thanks for believing in me.

REFERENCES

- [1] Alan G. Yoder and David L. Cohn, "Real spreadsheets for real programmers," in *ICCL*, H. E. Bal, Ed. IEEE Computer Society, 1994, pp. 20–30.
- [2] R. Abraham and M. Erwig, "Type inference for spreadsheet" in *PPDP '06: Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*. New York, NY, USA: ACM, 2006, pp. 73–84.
- [3] Orlando Belo, J'acome Cunha, "QuerySheet: A Bidirectional Query Environment for Model-Driven Spreadsheets," 2013 IEEE Symposium on Visual Languages and Human-Centric Computing, 2013 IEEE
- [4] J'acome Cunha, Jo~ao Paulo Fernandes, "MDSheet: A Framework for Model-Driven Spreadsheet Engineering" 978-1-4673-1067-3/12/\$31.00c 2012 IEEE
- [5] Zhen Hai, Kuiyu Chang, Jung-Jae Kim, and Christopher C. Yang "Querying Model-Driven Spreadsheets", 978-1-4799-0369-6/13/\$31.00 ©2013 IEEE
- [6] J'acome Cunha_y, Jo~ao Paulo Fernandes, "ES-SQL: Visually Querying Spreadsheets". 978-1-4799-4035-6/14/\$31.00 ©2014 IEEE
- [7] P. W. P. J. Grefen and R. A. de By, "A multi-set extended relational algebra - a formal approach to a practical issue,"in *ICDE*. IEEE Computer Society, 1994, pp. 80–88.
- [8] M. Kassoff, L.-M. Zen, A. Garg, and M. Genesereth, "PrediCalc:a logical spreadsheet management system," in *VLDB'05: Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 1247–1250.
- [9] B. Liu and H. V. Jagadish, "A spreadsheet algebra for a direct data manipulation query interface," in *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 417–428.
- [10] J. Tyszkiewicz, "Spreadsheet as a relational database engine," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 195–206.
- [11] D. Wakeling, "Spreadsheet functional programming," *J. Funct. Program.*, vol. 17, no. 1, pp. 131–143, 2007
- [12] A. G. Yoder and D. L. Cohn, "Architectural issues in spreadsheet languages," in *Proceedings of the international conference on Programming languages and system architectures*. New York, NY, USA: Springer-Verlag New York, Inc., 1994, pp. 245–258.