Tiled Spreadsheet Query Language

Mihai Dan
Oregon State University
Corvallis, Oregon
danm@oregonstate.edu

Parisa Sadat Ataei
Oregon State University
Corvallis, Oregon
ataeip@oregonstate.edu

ABSTRACT

Our research introduces a new spreadsheet query method reliant on user-defined tiles. We present the necessary changes to existing software to incorporate query functionality. From those changes, we show how to extend the software to allow for spreadsheet queries, namely through TSQL.

KEYWORDS

Spreadsheet, Query, Tile

ACM Reference Format:
Mihai Dan and Parisa Sadat Ataei. 2018. Tiled Spreadsheet Query Language. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Spreadsheets are wildly used by everyday users and corporations to manage and store various data. The data in a spreadsheet is arranged in the rows and columns of a grid, which allows for data to be accessed and manipulated in a systematic manner. Due to their high functionality and relatively low overhead learning cost, spreadsheets have been wildly adopted and modified to fit specific needs. These sheets can sometimes get large and riddled with information, making them hard to parse. Our research proposes a Tiled Spreadsheet Query Language (TSQL), which allow users to query a spreadsheet for desired information and create different visualizations of their data.

Spreadsheets are not trivially composed; the arrangement of values and formulas in cells serve a specific purpose endowed by the user. Imagine a spreadsheet for a large company that tracks employees and their respective productivity level for each month out of a year. The different categories, or tiles, are Q1, Q2, Q3, and Q4 for each quarter of the year, as well as Salary and Hourly for the different type of employees. In order to assess progress over Q1 and Q4, the user would manually have to sort through the data and pull only relevant information out. This manual process is both time consuming and error prone, especially as spreadsheet size increases. Using TSQL, the user would simply have to create a query and the data would be automatically presented.

Specifically, our research aims to answer these questions:

Research Question 1. What changes need be made to existing software to allow the implementation of TSQL?

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
Conference'17, July 2017, Washington, DC, USA
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Research Question 2. What additions must be made to the current software to allow for a query language?

A query language greatly improves the functionality of spreadsheets, both at the individual and enterprise level. TSQL allows the user to view data in a way that is suitable for what they are trying to accomplish. Besides increasing efficiency of attaining data, it would allow for potential new ways of spreadsheet analysis.

2 MOTIVATING EXAMPLE

The spreadsheet briefly discussed in the introduction is shown in Figure 1. This spreadsheet contains information about employees and the number of products that they have sold in each month of the year. Now assume that we want to check the progress of hourly employees over the year to determine whether to sign a yearly contract with them or not. To this end, we want to compare the first and last quarter that they have worked for the company.

Manual Process The spreadsheet contains information pertinent to individual months, as well as individual employees. In order to check progress over the year, a user would have to parse and transform the data manually, selecting only hourly employees data from the first and last quarter. This process requires careful attention to the quarter partition and selection of employees. The example spreadsheet is a simplified version of a real life scenario; in real use cases, these spreadsheets can grow to tremendous sizes, making manual parsing and transforming almost impossible!
Assuming the correct data is collected, the user needs to create a new spreadsheet to represent the desired data. The process is tedious and error prone, making it an undesirable query method.

Automated Query The user has specified the following tiles over this spreadsheet:

- *name*: employee names.
- *hourly*: employees with a hourly rate contract.
- *salary*: employees with a fixed salary contract.
- *Q1*: sales of the first quarter of the year.
- *Q2*: sales of the first quarter of the year.
- *Q3*: sales of the first quarter of the year.
- *Q4*: sales of the first quarter of the year.

In order to assess progress for hourly employees, the user would have to take a few steps to build a correct query. Since TSQL is compositional, we can build the query incrementally, as shown below. Note that the figures used in the motivating example are simply visual aids to demonstrate the functionality of TSQL and do not reflect the result of running a query.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Alice	99	85	61	63	53	74	76	93	95	88	73	62
Bob	97	54	89	90	47	67	62	92	78	75	92	48
Carl	95	45	31	29	53	88	90	77	83	49	90	78
Dan	38	30	44	57	90	82	56	95	78	64	84	94
Ed	81	46	61	45	25	37	36	43	49	76	54	82

Notes:
hourly employees: Alice - Bob
salary employees: Carl - Ed

Figure 1: Productivity Spreadsheet

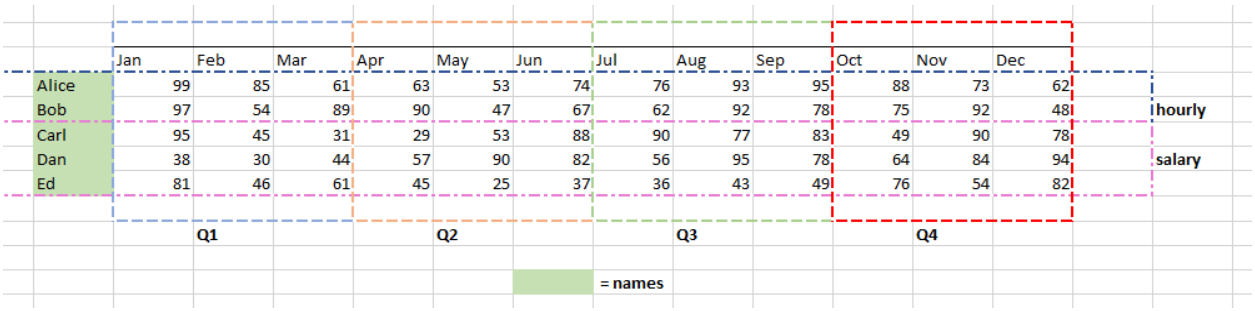

Figure 2: Productivity Tiled Spreadsheet

First, we need to extract employee names. We simply do this by calling the name associated to the correspondent tile: *name* and we get the tiled spreadsheet shown in Figure 3.

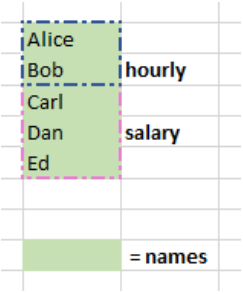

Figure 3: Output of the *name* query.

Then, we need to extract hourly employees. We simply query: *name*.*hourly* and get the result shown in Figure 4.

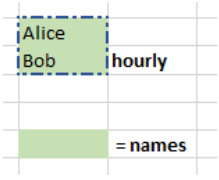

Figure 4: Output of the *name*.*hourly* query.

Then, we need to get the number of products they have sold in the first quarter of the year. We query: *Q1*.*hourly* and get the tiled spreadsheet shown in Figure 5.

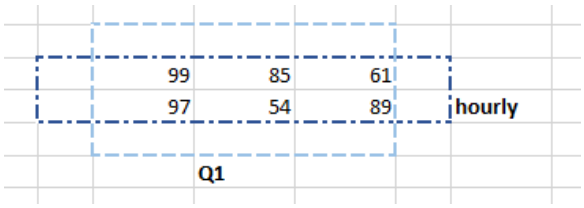

Figure 5: Output of the *Q1*.*hourly* query.

Similarly, we query *Q4*.*hourly* to get the number of products that hourly employees have sold in the last quarter of the year and we get the result shown in Figure 6.

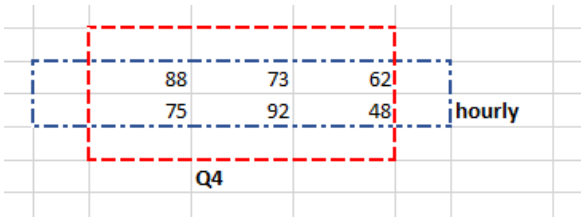

Figure 6: Output of the *Q4*.*hourly* query.

Now, we need to combine these spreadsheets to have a consistent useful spreadsheet that allows us to track the progress

that hourly employees have made throughout the year. We use our horizontal composition operation and query: $Q = name.hourly \triangleright Q1.hourly \triangleright Q4.hourly$ to get the result shown in Figure 7.

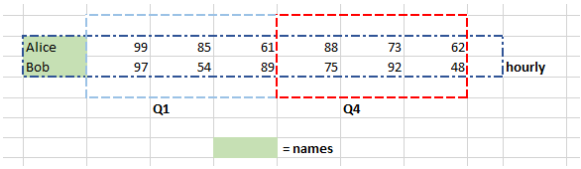

Figure 7: Output of the *name.hourly▷Q1.hourly▷Q4.hourly* query.

Finally, we want to name the spreadsheet returned to us in the last step. Hence, we query $\rho_{progress}Q$ and get the final result shown in Figure 8.

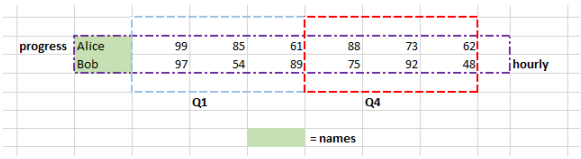

Figure 8: Output of the renaming query.

3 TSQL IMPLEMENTATION

This section covers technical challenges, existing software, and proposed solution, pertaining to the Tiled Spreadsheet Query Language.

3.1 Technical Challenges

Currently, there is no query tool that adequately allows users to compare different areas of a spreadsheet. The problem is nontrivial to solve because of the current limitations shown by existing software. While conducting research pertinent to existing applications, we noticed that a user is able to perform queries over a specified area in a spreadsheet but could not combine or construct several areas for further analysis. One of the challenges foreseen for TSQL is an efficient way of collecting tile information from the user that can scale to large spreadsheets. TSQL aims at giving the user the most attainable freedom over the structure and grouping of their tiles, creating a query suiting their specific vision. This problem is currently being addressed as prototype implementations of TSQL are tested.

TSQL should provide functionality, but at the same time be usable. A challenge encountered when designing TSQL was coming up with powerful syntax that is also easily understood. Originally, the syntax was very similar to SQL, but behaved differently, creating ambiguity for anyone who tried to use it. After many revisions, we decided on syntax similar to relational algebra, which better represents the functionality.

A challenge that came up during further research is the integration into Excel or a similar tool. Excel is a common platform for spreadsheet manipulation and would be a suitable platform for TSQL. Currently, we are unsure of how we will implement this functionality.

3.2 Existing Software for Spreadsheet Representation

This research is built upon the `Smash` library, which is used for spreadsheet representation. This library has been used in various research conducted by several programming languages and human-computer interaction groups. An example use case of this library is shown by Abraham and Erwig in [1]. The paper presents an automated system for error detection in spreadsheets.

This library contains several data constructors that encompass a variety of possible cell entries in a spreadsheet. A spreadsheet is represented by `Sheet Fml`, which is a collection of `Cell Fml`. The `Cell` type contains the index of the cell, as well as the formula which it holds. The `Fml` constructor represents the various possible cell entries such as formulas and numbers.

One of the key functionalities of the library is the translation of a CSV spreadsheet into a Haskell data type. Haskell's powerful computation tools allow for in depth analysis and transformation of spreadsheets. Figure 9 is a visual representation of a spreadsheet in Haskell. Unfortunately, after iterations of refactoring, this functionality is not working as intended; instead of each cell being parsed as the correct type (formula, input, etc.), they were parsed in as strings. Correct typing of cells is crucial when checking for errors in user-defined queries. This issue, along with its respective solution, is further discussed in Section 3.3.1.

While this library has many interesting and useful functionalities, it does not provide an efficient way for querying information from a spreadsheet. Research Question 2 focuses on this aspect of the research presented by this paper. The implementation of TSQL is shown in detail in Section 3.3.2.

3.3 Proposed Solution

Definition 3.1. A **tile** is a rectangular area that is annotated with a name. The rectangular area is represented by the coordinations of the leftmost top and rightmost bottom cells.

Definition 3.2. A **tiled spreadsheet** is a spreadsheet that includes tiles.

Our current proposed approach is to create a domain specific language that allows a user to create queries over a tiled spreadsheet. The initial design was naive and lacked a clear vision of the end goal. Currently, we are working with tiles defined by the user and a simple, yet powerful, language to create queries.

3.3.1 Software Maintenance. One of the immediate challenges when using the `Smash` library was the malfunctioning CSV spreadsheet parser. The lack of this functionality is detrimental to the ability to progress on implementation; it hinders the ability to check user-created queries for mistakes.

In order to tackle this problem, we had to assess all functions related to the CSV parsing. The task of following information flow through several files turned out to be a bigger task than anticipated. Haskell's conciseness in code is one of its strengths, but also a great weakness - readability is sacrificed. The original

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2	Alice	99	85	61	63	53	74	76	93	95	88	74	62
3	Bob	97	54	89	90	47	67	63	93	78	75	92	48
4	Carl	95	45	31	29	89	88	90	77	83	49	90	78
5	Dan	38	30	44	57	90	82	56	95	78	64	84	94
6	Ed	81	46	61	45	25	37	36	43	49	76	54	82

Figure 9: Haskell Representation of the Productivity Spreadsheet

authors of the parser also user function names such as `allRun` which is not entirely indicative of its purpose.

The bug was introduced in the function `readMsg`, which was responsible for reading the string and parsing it into a `Cell` data type. The function correctly parsed the index of the cell, but saved all inputs as strings. To alleviate this issue, we created a function `inp2fml` (Input to Formula) to further process the string and produce the correct type.

A maintenance challenge arose from the design of the spreadsheet representation. Thus far, research that has worked with the `Smash` library has not had a need for representing empty cells. The user-defined tiles create a partition of the original spreadsheet; that is, the union of the cells in all tiles must create the spreadsheet. In order for this to be true, empty cells need be accounted for as well. The `Fml` data type was modified to reflect this change as shown in Figure 10. The ellipses denote the rest of the data type definition which is not relevant to this section. Note that the Empty constructor takes no arguments, meaning that the cell cannot hold any information.

```
data Fml = ...
         | Empty   (empty cell)
```

Figure 10: Fml data type modification

3.3.2 Software Evolution. The `Smash` library provides a way to represent and manipulate spreadsheets, but does not have any support for tiles. We defined a new data type, `TSheet` to represent tiled spreadsheets, shown in Figure 11.

```
type Name = String

type Col = Int
type Row = Int
data Indx = Indx Col Row

type Area = (Indx, Indx)
type Tile = (Name, Area)

type Cell a = (Indx, a)
newtype Sheet a = Sheet unSheet::[Cell a]

type TSheet a = (Sheet a, [Tile])
```

Figure 11: Basic TSQL Objects

The `Name` type represents the name of a tile. The `Indx` data type is an integer representation of a row and a column within a spreadsheet. The `Area` type is a pair of indexes representing the top left most and bottom right most cells of a tile, respectively. From these cells, we can determine the area occupied by a tile. The `Tile` type associates a tile name with its area. The interesting type to note in Figure 11 is `TSheet a` data type; this type associates a spreadsheet with its corresponding tiles.

It is important to note that, as of now, we have not discovered a visual representation fit for a tiled spreadsheet. This representation will be further developed and investigated after the functionality of TSQL is finalized.

Query Language. The query language currently consists of six operations; region, intersection, horizontal composition, vertical composition, renaming, and difference. The region selection operation returns a tile based on the input name. Calling "Sheet" will return the full spreadsheet as the result. The intersection returns a tiled spreadsheet that consists of the intersecting cells in the input queries. Horizontal and vertical composition allows for the combination of tiled spreadsheets resulting from the input queries. The renaming operation simply renames the result of the input query. This allows for further customization of tiles, as shown by the motivating example. Newly added, the difference operation returns all cells from tile A that are not also part of B. These constructors are continuously updated as further research is conducted.

$$
\begin{array}{lcll}
q & ::= & n & \textit{Region selection} \\
 & | & q \bullet q & \textit{Intersection} \\
 & | & q \triangleright q & \textit{Horizontal composition} \\
 & | & q \triangledown q & \textit{Vertical composition} \\
 & | & \rho_n q & \textit{Naming operation} \\
 & | & q - q & \textit{Difference}
\end{array}
$$

Figure 12: TSQL Syntax

3.4 Implementation Progress

Currently, our research is solely focusing on the functionality of TSQL before any effort is put towards visualizing the tiled spreadsheets. Our software is capable of performing region selection, vertical composition, naming, and difference. Figure 13 shows the Haskell representation of Figure 5.

```
*TSQL.TQuery> ex_q1_hourly
(     | B  | C  | D
------------------
   2 | 99 | 85 | 61
   3 | 97 | 54 | 89
,[("Sheet",(B2,D3)),("Q1",(B2,D3)),("Hourly",(B2,D3))])
```

Figure 13: Output of the *Q4.hourly* query.

As discussed previously, we have yet to finalize a visualization for the tiles. They are currently displayed as a list of pairs, consisting of the tile name and Area.

We have also began drafting the type checker for this query language. A type checker is necessary to prevent any errors that may occur from errors in user-defined queries.

4 RELATED WORK

This section includes research similar in nature to TSQL, mainly focusing on spreadsheet transformation. Each research paper is briefly explained and compared to various aspects of TSQL.

4.1 FlashExtract: A Framework for Data Extraction by Examples

FlashExtract[3] is a general framework to extract relevant data from semi-structured documents, such as a spreadsheet. The framework relies on the user input of nested hierarchical definition of the data, as well as the relationships amongst the various data fields. The user is then free to create queries over specific fields based on values or constraints. This research is similar to TSQL in the sense that it queries over structured documents, but there are many ways in which they differ. FlashExtract constraints the user to a hierarchical initial composition, meaning several groups of interest cannot be expressed if they share data. The TSQL tiles are unconstrained, which can lead to interesting ways of looking at structured documents such as spreadsheets.

4.2 Spreadsheet Table Transformations from Examples

The contribution of this research[4] is a language of programs that represents large set of practical transformations over spreadsheets, namely TableProg. Along with TableProg, the authors present ProgFromEx, an algorithm that infers a TableProg program from example inputs and outputs. The aim of their research was to alleviate end user programmers of struggling through scripting languages provided by programs such as Excel. TableProg provides a larger set of transformations to encompass any example input-output relationship that may occur. One of the strengths of TSQL is the simplicity of the language, yet still preserving spreadsheet transformation power. ProgFromEx requires an example input and output spreadsheet to create a TableProg explanation. TSQL demonstrates more freedom in the output, allowing the user to build the output with the query.

4.3 Potter's Wheel: An Interactive Data Cleaning System

Much like TSQL, Potter's Wheel[6] is a spreadsheet transformation system. While Potter's Wheel is not a query language, it shares many similarities with TSQL. Potter's Wheel is an interactive data cleaning system that performs data transformation, as well as discrepancy detection. Users are able to compose, as well as debug, transformations to their spreadsheet. If any discrepancy is found in the data, the user is alerted. However, Potter's Wheel does not support any sort of user annotating of the spreadsheet; it iterates over rows and columns until the values satisfy the constraint. TSQL allows the user to think about the spreadsheet as groups of data represented by tiles. This benefit can lead to the user to view the spreadsheet from a different perspective, inspiring queries that match their vision.

4.4 Automating String Processing in Spreadsheets Using Input-Output Examples

This research[2] presents an expression language that is expressive enough to represent a large set of string manipulation tasks that end user programmers struggle with. The language was designed around an extensive study of Excel online help forums. This research aims at shortening the user-expert conversation by providing a language that expresses transformation automatically. The interesting thing to note about this research language is the definition of the language. The expressions are focused around strings and string manipulation, which may be restrictive if one tries to perform any other form of transformation. This could be applicable to less fine-grained operations and applied to the whole spreadsheet.

4.5 A Spreadsheet Algebra for a Direct Data Manipulation Query Interface

The authors present an SQL-like spreadsheet algebra [5] capable of performing queries over spreadsheets. The design is inspired by the queries performed over relational databases, but heavily influenced by the structure of a spreadsheet. The research presents several challenges faced by the data structure, such as grouping data, query composition, operator ordering. The original idea for TSQL was similar to this; injecting SQL-like queries throughout a spreadsheet. While this view is functional, it is limited as to what kind of tables can be produced. The tiled spreadsheet approach allows for analysis and comparison of larger groups of data, abstracting from the constraints of considering individual cells.

REFERENCES

[1] Robin Abraham and Martin Erwig. 2007. UCheck: A spreadsheet type checker for end users. *Journal of Visual Languages And Computing* 18, 1 (2007), 71 – 95. https://doi.org/10.1016/j.jvlc.2006.06.001

[2] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-output Examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '11)*. ACM, New York, NY, USA, 317–330. https://doi.org/10.1145/1926385.1926423

[3] Sumit Gulwani. 2014. FlashExtract: A Framework for Data Extraction by Examples. https://www.microsoft.com/en-us/research/publication/flashextract-framework-data-extraction-examples/

[4] William R. Harris and Sumit Gulwani. 2011. Spreadsheet Table Transformations from Examples. *SIGPLAN Not.* 46, 6 (June 2011), 317–328. https://doi.org/10.1145/1993316.1993536

[5] Bin Liu and H. V. Jagadish. 2009. A Spreadsheet Algebra for a Direct Data Manipulation Query Interface. In *Proceedings of the 2009 IEEE International Conference on Data Engineering (ICDE '09)*. IEEE Computer Society, Washington, DC, USA, 417–428. https://doi.org/10.1109/ICDE.2009.34

[6] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 381–390. http://dl.acm.org/citation.cfm?id=645927.672045