HHS Public Access

Author manuscript

Proceedings VLDB Endowment. Author manuscript; available in PMC 2016 February 17.

Published in final edited form as:

Proceedings VLDB Endowment. 2015 August ; 8(12): 2000–2003.

DataSpread: Unifying Databases and Spreadsheets

Mangesh Bendre, **Bofan Sun**, **Ding Zhang**, **Xinyan Zhou**, **Kevin ChenChuan Chang**, and **Aditya Parameswaran**

University of Illinois at Urbana-Champaign (UIUC)

Mangesh Bendre: bendre1@illinois.edu; Bofan Sun: bsun6@illinois.edu; Ding Zhang: dzhang13@illinois.edu; Xinyan Zhou: xzhou14@illinois.edu; Kevin ChenChuan Chang: kcchang@illinois.edu; Aditya Parameswaran: adityagp@illinois.edu

Abstract

Spreadsheet software is often the tool of choice for ad-hoc tabular data management, processing, and visualization, especially on tiny data sets. On the other hand, relational database systems offer significant power, expressivity, and efficiency over spreadsheet software for data management, while lacking in the ease of use and ad-hoc analysis capabilities. We demonstrate DataSpread, a data exploration tool that holistically unifies databases and spreadsheets. It continues to offer a Microsoft Excel-based spreadsheet front-end, while in parallel managing all the data in a back-end database, specifically, PostgreSQL. DataSpread retains all the advantages of spreadsheets, including ease of use, ad-hoc analysis and visualization capabilities, and a schema-free nature, while also adding the advantages of traditional relational databases, such as scalability and the ability to use arbitrary SQL to import, filter, or join external or internal tables and have the results appear in the spreadsheet. DataSpread needs to reason about and reconcile differences in the notions of schema, addressing of cells and tuples, and the current “pane” (which exists in spreadsheets but not in traditional databases), and support data modifications at both the front-end and the back-end. Our demonstration will center on our first and early prototype of the DataSpread, and will give the attendees a sense for the enormous data exploration capabilities offered by unifying spreadsheets and databases.

1. INTRODUCTION

Since the early days of computing, spreadsheet software, such as VisiCalc, Lotus 1-2-3, and more recently Microsoft Excel and Google Sheets, have found ubiquitous use in ad-hoc tabular data analysis, especially by non-programmers; including statisticians, finance professionals, consultants, and physical scientists. The main advantages of spreadsheets include the ability for direct manipulation of data, an intuitive user interface, and a flexible data model with the ability to add new rows, columns, or tuples, seamlessly.

However, spreadsheet software has many limitations, making it unsuitable for present-day big data analysis, primarily due to poor performance on large data sets, and the low

Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st September 4th 2015, Kohala Coast, Hawaii.

expressivity of the spreadsheet syntax [1, 2]. For the former issue, i.e., poor performance, for example, in Microsoft Excel, it is common knowledge that beyond a few 100s of thousands of rows, the software is no longer responsive [1]. For the latter issue, there are a number of common data analytics operations that are either very cumbersome or not easy to do in spreadsheet software. To illustrate this, consider a simple example, where a user is studying a spreadsheet containing course assignment scores and eventual grades for students from rows 1–100, columns 1–5 in one sheet, and demographic information for the students from rows 1–100, columns 1–10 in another sheet. Consider the following operations that the user may want to do to compute some intermediate tabular result (The user may then visualize or study this result in some way.) :

- Say the user wants to understand the impact of assignment grades on the course grade, for which they want to select the students having points higher than 90 in at least one assignment. There is no way for the user to sub-select a set of rows of this form for further analysis, except manually identifying these rows, and then copy-pasting each one into another area.
- Say the user wants to plot the average grade by demographic group (undergrad, MS, PhD). This requires a “join” of the two sheets of the spreadsheet to generate the desired result, also very cumbersome to do on current spreadsheet software.
- Say the course management software outputs actions performed by students into a relational database or a CSV file; there is no easy way for the user to study this data within the spreadsheet, as the data is continuously added.

There are many other data analysis operations that are similarly very cumbersome on current spreadsheet software.

Therefore, we propose to *bring the power of relational databases to bear on spreadsheets*. Relational databases are efficient and expressive, and are certainly capable of natively handling the operations described above via SQL. On the contrary, relational databases are not as easy-to-use or as amenable to direct manipulation as spreadsheet software, e.g., seamlessly adding a new column, copy-pasting data. Thus, we unify relational databases with spreadsheet software, in order to preserve the benefits of both.

We propose a system, DataSpread, that is a holistic unification of relational databases and spreadsheets. Here we use spreadsheet as an intuitive user interface and database as a back-end engine. However, designing DataSpread is not trivial since databases and spreadsheets adopt very different architectures and ideologies. In particular, we need to deal with the following challenges:

- Schema: databases have a strict schema-first data model, which is based on tables and tuples, while the spreadsheet data model is based on sheets with rows and columns, and no explicitly defined schema.
- Addressing: spreadsheets treat rows and columns as identical, while databases operate on sets of tuples.

- Window: spreadsheet have the notion of the current window, which is the portion of the spreadsheet that the user is currently looking at; there is no such notion in databases.
- Modifications: spreadsheets support updates at any level and granularity: rows or columns, while databases only support modifications that correspond to a SQL query.
- Computation: spreadsheets support value-at-a-time formulae to allow derived computation, while databases support arbitrary SQL queries operating on groups of tuples at once.

We have identified these as research issues and have build the first version of DataSpread to explore them, among others. Externally, DataSpread retains many of the front-end user interface aspects of spreadsheets that make it as easy to use, while at the same time enhanced and supported by a back-end relational database, providing efficiency and expressivity. In the front-end, in addition to all the traditional spreadsheet commands, DataSpread supports the use of arbitrary SQL via custom dbsql and dbtable commands, enabling the import, and constant updating of data from relational databases, as well as the computation of selections and joins of data contained in the spreadsheets. Conceptually, these commands, along with other spreadsheet commands, are stored as *interface views* in the underlying database. In the back-end, an optimizer, optimizes for keeping the user window up-to-date and in-sync with the underlying relational database. Even though the spreadsheet can only support a few rows, as the user pans through the spreadsheet, the burden of supplying or refreshing the current window is placed on the relational database, which is very efficient.

Demonstration

In our demonstration, we will allow conference attendees to interact with our prototype of DataSpread (built using Microsoft Excel and PostgreSQL), enabling them to interactively analyze a two-way synchronized view of relational data using more expressive dbsql, dbtable commands to filter, join, project, and export data residing in multiple sheets.

Related Work

With the goal to achieve the benefits of spreadsheets and relational databases while dealing with tabular data, our holistic unification strives to unify the notion of table in both systems. Recent works have proposed to enrich spreadsheets and relational databases with features from one another in three orthogonal directions: *a*) Use of spreadsheets to mimic the relational database functionalities [3]: Although this approach achieves expressivity of SQL, it is unable to leverage the scalability of databases. *b*) Use of databases to mimic spreadsheet functionalities [4, 5]: Although this approach achieves scalability of databases, it is does not support ad-hoc tabular management provided by spreadsheets. *c*) Use of spreadsheet interface for querying data [6]. This approach provides an intuitive interface to query data, but looses the expressivity of SQL as well as ad-hoc data management capabilities.

Rest of the Paper

In the next section we propose a desired design by developing a unification semantics. We then use the semantics to propose an architecture for DataSpread. Finally, we discuss demonstration scenarios for our DataSpread prototype.

2. DESIGN OF DataSpread

In this section, we describe the semantics for DataSpread. In particular, we discuss some important concepts and challenges that arise due to the unification of the two disparate ideologies: spreadsheets and databases.

2.1 DataSpread Overview

With a goal of unifying databases and spreadsheets, we now propose a framework for DataSpread based on two key ideas. First, to leverage the intuitiveness and the richness of a spreadsheet interface, rather than changing it significantly, we enhance it with concepts borrowed from databases. Underneath the interface, we propose to have a relational database that is enhanced to support the spreadsheet interface. Second, to improve the expressivity of the interface, we expose some database features, for example, declarative querying, from the underling database to the interface. Using these two key ideas, we enable users to leverage the strengths of both spreadsheets and databases for dealing with tabular data.

2.2 Semantics and Syntax

Although spreadsheets and databases have both been designed to manage data in form of tables, their treatment of this data is vastly different. Spreadsheets have been developed primarily with presentation of data in mind and hence their design focuses primarily on simplicity, intuitiveness and a rich user interface. On the other hand databases have been designed with powerful data management capabilities to work with large tables. Hence, certain data manipulation operations, e.g., queries, joins, summarization, are very naturally expressed as SQL statements in databases.

We propose semantics for DataSpread such that we are able to naturally leverage the strengths of both systems. Since we plan to enrich databases to effectively support interfaces, we use the strong points of spreadsheets to motivate our semantics.

Support for Dynamic Schema—Spreadsheets enable users to effortlessly create tables and update their schema. A user typically structures data on a spreadsheet as tables, with columns and rows, where columns generally correspond to attributes and rows to tuples. Here, adding an attribute, which is essentially a change to schema, is as natural and convenient as adding a tuple. This is due to the fact that spreadsheets do not treat columns and rows differently when we consider the operations possible on each. On the other hand, relational databases have a schema-first data model. Relational tables, which belong to a database's schema, need a pre-defined structure in terms of attributes. Since changing the structure of a table in a database requires an update to all its tuples, it is not efficient as adding, deleting or updating the tuples of the table.

To make relational table creation as effortless as table creation on a spreadsheet, we propose the ability for a user to select an arbitrary range on the spreadsheet and use it to define the structure and the data for a table within the database. Once created it should behave like a regular table within the database, and the user should be able to refer to it and use it in queries.

To streamline the concept of a dynamic schema, we propose that a user is able to update a table's schema and tuples that are displayed on a spreadsheet, which in turn updates the schema and tuples of the underlying table in the database. Further, the database should be able to handle this schema change with an efficiency similar to tuple updates. This makes table updates within a database as natural as updating them on a spreadsheet.

Make Databases Interface Aware—Since spreadsheets have been designed with an interface in mind, they very naturally lay out data that is both consumed and manipulated by users. This interface has a very strong influence on functionality offered to the user. Features like laying out a table in a desired format and obtaining the totals of some attributes beneath the table (using a spreadsheet formula) feel natural. Thus, the interface provides a *context* to the operations performed on a spreadsheet.

Positional addressing, which enable users to address data based on its position on a spreadsheet, is an intuitive and effective way to refer to presented data. By laying out data on a spreadsheet, a position gets implicitly assigned to the displayed data, due to which a spreadsheet is able to use positional referencing, e.g., a cell reference of $_{A2}$ from cell $_{C2}$ implies a cell that is two columns left and in the same row. The positional referencing is a commonly used feature while building expressions as it enables us to copy expressions across cells while still maintaining the relative references.

Conversely, databases completely lack interface aspects. Once a query result is output, the database is no longer cognizant of how that result is consumed. This disconnect is a key weakness due to which a database cannot be used as-is to effectively support a spreadsheet interface. For instance, when a user wants to update a specific attribute of a displayed table, the database is unable to help because it is not aware of the tuple or attribute being modified.

We propose to make databases aware of the interface's data layout. This enables them to understand interactions on the presented data, e.g., for a join using displayed tuples, the database is able to identify the tuples just based on their implicit context. This further enables the databases to optimize the query execution by prioritizing the displayed tuples over the ones that are not displayed.

After making the database interface aware, we propose to leverage this to enable positional addressing in databases. This implies that the user should be able to refer to a value by its location on the spreadsheet and use it in any arbitrary query.

Novel Spreadsheet Constructs—We now describe how the positional addressing is leveraged in the front-end spreadsheet, enabling users to pose rich SQL queries while referring to data in the spreadsheet as well as the underlying relational table.

We encapsulate SQL references within the spreadsheet using one of two formulae: DBSQL and DBTABLE. DBSQL enables users to pose arbitrary queries combining data present on the spreadsheet, and data stored in the relational database. DBTABLE enables users to declare a portion of the spreadsheet as being either exported to or imported from the relational database, i.e., that portion of the spreadsheet directly reflects the contents of a relational database table.

In order to support arbitrary positional addressing or referencing of data on the spreadsheet for DBSQL, we add two new constructs: RANGEVALUE and RANGETABLE. This enables users to refer to a cell and a table on a sheet respectively relative to the cell where the query is entered. RANGEVALUE enables a user to refer to scalar values contained in a cell, e.g., SELECT FROM ACTORS WHERE ACTORID = RANGEVALUE(A1), referring to the value in cell A1. RANGETABLE on the other hand enables a user to refer to a range, and perform operations on it assuming it is a regular database table. This enables any range on a spreadsheet to be potentially a table, and all the operations, e.g., join, that the database allows on a table can be performed, e.g., SELECT FROM ACTORS NATURAL JOIN RANGETABLE(A1:D100).

Other Semantic Issues—Although we have discussed two important concepts, there are still many semantics that require attention if we want to realize a complete unification. Due to the space restriction, rather than discussing them in detail we have listed a few of them below: *a*) SQL support on spreadsheets: To leverage the expressiveness of SQL and the simplicity of formulae we propose to support both, and give flexibility to the user to interchangeably use either. *b*) Real-time sync: Using spreadsheets users are accustomed to having an always updated copy with them. For this we propose a real time two way synchronization of the displayed on the spreadsheet with the underlying database. *c*) Data typing: Spreadsheets dynamically type the data stored as cells. To make this work with databases, we propose the idea of automatically assigning data types within the databases based on the tuples. *d*) Computation optimization: By scaling up the amount of data, which can be presented on a spreadsheet, efficient computation become a necessity. We propose to leverage the presentation information for prioritizing computations for the data that is displayed. *e*) Lazy Computation: To maintain interactivity, we propose that the calculations of the visible cells should be prioritized and the remaining long running computations should be performed in background.

Challenge—Realizing the unified semantics is not a trivial task, since it stretches the capabilities of today's relational databases beyond what is available. For example, consider the semantics of schema, for today's databases a table's schema change requires an update to all the tuples of the table. Further, the activity is considered as "data definition language" and generally cannot participate in transactions. This requires us to propose the architecture of DATASPREAD by radically rethinking the databases' architecture.

3. PROPOSED ARCHITECTURE

Since relational databases are not designed to be interface-aware, when we unify the presentation layer of spreadsheets with databases, we need to redesign the underlying architecture of the database, as well as the interaction with the front-end interface.

To enable databases to support the semantics described earlier, we propose a redesigned database architecture as shown in Figure 1, where the shaded blocks represent new or enhanced components. The *interface manager* is tasked with the goal of making databases interface-aware. The *query processor* is enhanced to support and optimize the execution for positional addressing, a natural way to locate data presented on the interface. The *compute engine* leverages interface aspects, e.g., windowing, to optimize execution. We introduce a new type of index, *positional*, which makes interface-oriented operations, e.g., ordered presentation, efficient. The *interface storage manager* stores data that is presented on the interface but not designated as a relational table. The *relational storage manager* is enhanced to effectively support interface related operations such as schema changes.

While we have identified the extent of modifications needed for databases to effectively support an interface, our current implementation and discussion focuses on enhancing some core components. Naturally, there are other components that require modification, such as the transaction manager, and we leave them for future work.

Interface Storage Manager

In this unified framework, a spreadsheet not only has tabular data, corresponding to relational tables in the underlying database, but also has other interface data, e.g., formulae or data entered by the user. This interface data requires special treatment as it does not have a schema. The interface storage component stores this data as a collection of cells. To enable efficient retrieval for a given range, the component groups the cells together by proximity and splits the groups into data blocks as required by the underlying storage. To enable efficient access, the blocks are further indexed by a two-dimensional indexing method.

Relational Storage Manager

Our unification semantics demand that the schema changes to the tabular data, which we persist in the database as relational tables, should be very efficient, almost as efficient as changes to tuples. With an insight to reduce the disk blocks to update during a schema change, the relational storage manager uses a hybrid of column-store and row-store to physically store the table. Here, data is structured along a collection of attribute groups, thereby radically reducing the disk blocks that need an update during a schema change.

Interface Manager

The interface manager keeps close tabs on the data presented to the user. For every data item, e.g., the output of a query, a table imported from the database, that is displayed on the interface, the presentation manager assigns a *context*; a context comprises a positional address along with a reference to the sheet. This context can then be utilized to enable functionalities such as two-way sync and relative addressing.

Along with positional addressing, the interface manager allows a two-way synchronization for the tables displayed on the interface. Since primary keys are a natural way to identify tuples in a relational database, the interface manager maintains a mapping between a tuple’s key attribute and its corresponding location. This enables translation of an update on the

interface, having a locational context, to the underlying relational database, which requires a key to uniquely identify a tuple.

Compute Engine

To optimally support interface interactions and data updates, we introduce a new component termed as “compute engine”. By using ideas like shared computation, the compute engine enables efficient handling of formulae and queries with positional referencing, e.g., DBSQL. It performs computations asynchronously, free from a user’s context, as updates are made to either the interface or the database. It further improves the interface’s interactivity by prioritizing the computation for visible cells.

4. DEMONSTRATION DESCRIPTION

Our DataSpread prototype is implemented using Microsoft Excel (that presumably most conference attendees as well as eventual users are already familiar with) as the front-end spreadsheet application, backed by PostgreSQL as the relational database backend. All the screenshots we depict are from our current prototype. A video demonstrating the features of DataSpread can be found at http://dataspread.cs.illinois.edu.

We demonstrate the following features of the DataSpread prototype: *a*) analytic queries that reference data on the spreadsheet, as well as data in other database relations. *b*) importing or exporting data from the relational database. *c*) demonstrating that DataSpread keeps data in the front-end and back-end in-sync during modifications at either end.

Feature 1: Querying

Consider Figure 2a. Here, expressed using the DBSQL spreadsheet function, the SQL query in B3 uses data from three relations in the database (MOVIES, MOVIES2ACTORS, ACTORS), and references the two cells above (B1 and B2), via special relative referencing commands (RANGEVALUE(B1) and RANGEVALUE(B2)). The output of the query is not limited to a single cell, but spans the range B3:B10. This enables the collection of cells to be computed collectively in a single pass (as opposed to traditional spreadsheet formulae that are one-per-cell). This will demonstrate how DataSpread provides the ability to naturally query the underlying database, and other data in the spreadsheet.

Feature 2: Import/Export

Consider Figure 2b. Here, on selecting a range in the sheet and selecting the create table command from the add-ins menu, we provide the ability to users to transform it into a relational database table. The schema of this table is automatically inferred using the column heading and the data. Optionally, users will be allowed to specify constraints on the table, such as primary keys. On completion, the table is created in the underlying database. The data on the sheet is replaced by DBTABLE, which is a spreadsheet function that selects data from the database and displays it on the spreadsheet. DBTABLE could also be used to directly import data already present in the relational database into the spreadsheet. This will demonstrate how DataSpread allows us to import or export data to and from the relational database.

Feature 3: Modifications

Consider Figure 2c. Here, after a table is displayed on the spreadsheet using DBTABLE, and formatted in cells A3 to B5, as modifications are made to the table on the front-end the data in the relational database is updated, and the data displayed in cells from A10 to B12 (corresponding to a DBSQL command referencing that data) is immediately updated. This will demonstrate how DATASPREAD provides the ability to keep data in-sync during modifications at both the front-end and back-end

Overall, the aforementioned demonstration scenarios will convince attendees that our DATASPREAD system offers a valuable hybrid between spreadsheets and databases, retaining the ease-of-use of spreadsheets, and the power of databases.

REFERENCES

1. Clemens S. 5 Ways To Tell You Have Outgrown Excel. http://www.insightsquared.com/2011/06/5-ways-to-tell-you-have-outgrown-excel/.
2. Collie R. Big Data is Just Data, Why Excel “Sucks”, and 1,000 Miles of Data. http://www.powerpivotpro.com/2012/10/big-data-is-just-data-why-excel-sucks-and-1000-miles-of-data/.
3. Tyszkiewicz, J. SIGMOD. ACM; 2010. Spreadsheet as a relational database engine; p. 195-206.
4. Witkowski, A.; Bellamkonda, S.; Bozkaya, T.; Dorman, G.; Folkert, N.; Gupta, A.; Shen, L.; Subramanian, S. Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, SIGMOD ’03. New York, NY, USA: ACM; 2003. Spreadsheets in rdbms for olap; p. 52-63.
5. Witkowski, A.; Bellamkonda, S.; Bozkaya, T.; Naimat, A.; Sheng, L.; Subramanian, S.; Waingold, A. Proceedings of the 31st International Conference on Very Large Data Bases, VLDB ’05. VLDB Endowment; 2005. Query by excel; p. 1204-1215.
6. Liu, B.; Jagadish, H. Data Engineering, 2009. ICDE’09. IEEE 25th International Conference on. IEEE; 2009. A spreadsheet algebra for a direct data manipulation query interface; p. 417-428.

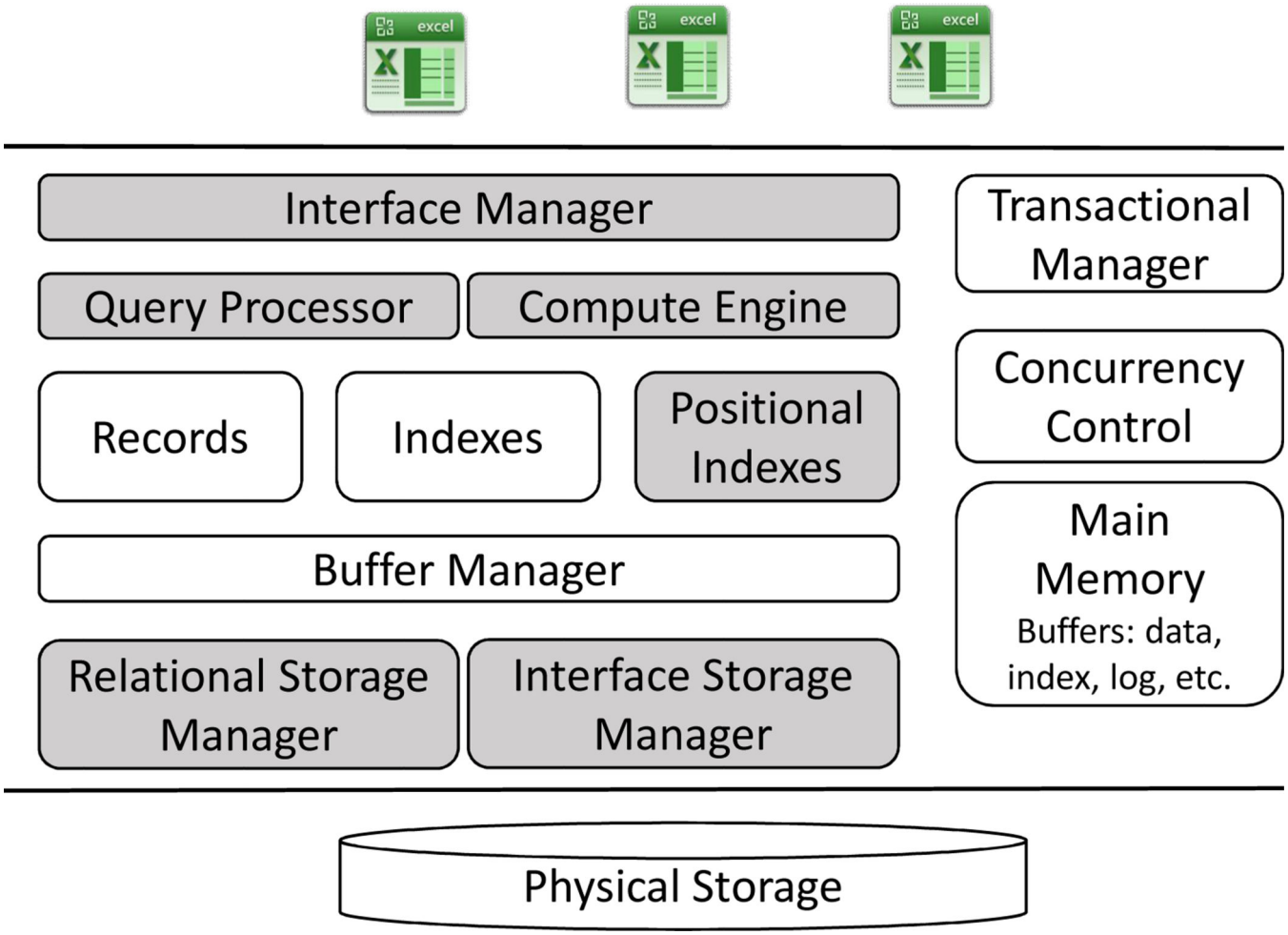

Figure 1.
DataSpread Architecture.

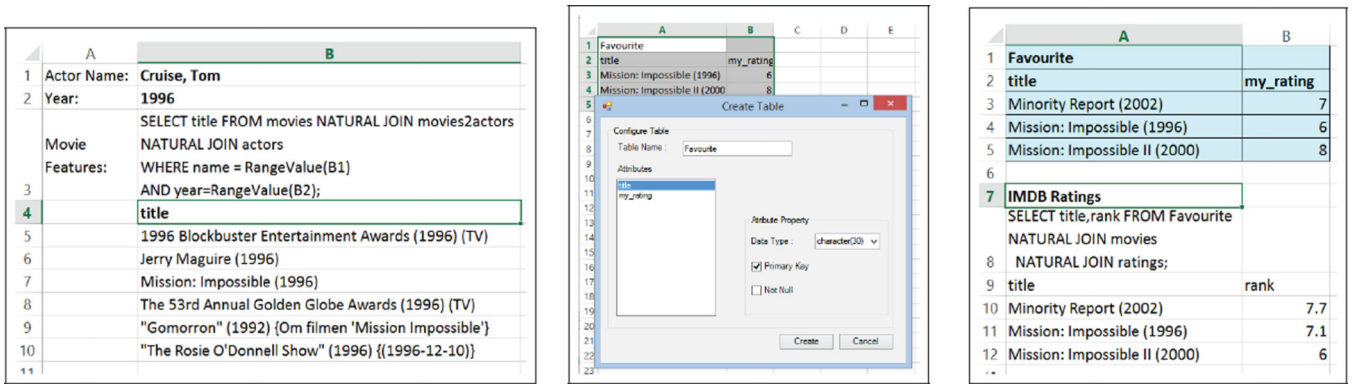

Figure 2.

(a) Executing SQL with relative referencing. (b) Table creation. (c) Two-way table sync.