Directed Data Management: A New Frontier in Database Usability

Mangesh Bendre, Sajjadur Rahman, Tana Wattanawaroon, Kelly Mack, Yu Lu
Kevin Chen-Chuan Chang, Karrie Karahalios, Aditya Parameswaran
University of Illinois at Urbana-Champaign (UIUC)
{bendre1 | srahman7 | wattana2 | knmack2 | yulu3 | kcchang | kkarahal | adityagp}@illinois.edu

ABSTRACT

Relational databases have limited usability for ad-hoc interactions with end-users. Inspired by the flexibility and ease-of-use of spreadsheets that stems from its direct manipulation capabilities, we propose a new research direction termed *directed data management*, to effectively bring the usability benefits of spreadsheets to databases, while not sacrificing the power and scalability of databases. We argue that on extremely large datasets the vanilla direct manipulation capabilities offered by spreadsheets are no longer effective, necessitating extensions to support multiple perspectives, accelerated actions, and progressive feedback. We describe the challenges underlying directed data management, as well as our progress towards this goal via our prototype system DATASPREAD, under development for the past two years.

1. INTRODUCTION

Despite the immense successes of relational databases in supporting enterprise applications, the usability of databases for ad-hoc interactions with end-users has long been deemed to be a severe issue, with every report on the state of the database research in the last decade acknowledging it as such [4, 7, 6]. Present-day interactions with data residing in databases happen via SQL queries—this declarative querying paradigm is central to the commercial success and adoption of databases. However, constructing SQL queries for every single interaction is often too cumbersome. Much research effort has therefore gone into making it easier to craft SQL queries, using gestures [28, 18], natural language [24], and autocompletion [5, 21]. While these efforts are notable, ad-hoc interactions with databases are still challenging, for two reasons: *operations* and *representation*. First, even if it is easier to issue correct SQL queries, SQL is still an indirect means for *operating* on data, requiring users to issue declarative queries in "batch mode" on relations at a time, waiting until the entire query is crafted and issued before seeing any results. Second, databases do not persist the state of the analysis, nor can users organize and *represent* their analysis results in a way that is easy to understand and share.

While the usability of databases for ad-hoc interactions is questionable, spreadsheet software is incredibly popular for such tasks, especially on small datasets, with Microsoft Excel having a user base of over 750 million people [2]. Spreadsheet users range from real-estate developers to financial analysts, from scientists to journalists, and from quantified self-enthusiasts to medical doctors [27]. Our anecdotal evidence suggests that even students who take advanced database classes at Illinois (via informal polls of 1000+ students in the past 15 years) do not use databases to manage their data, opting to use spreadsheets instead. Spreadsheets provide *direct manipulation capabilities*, as defined by Shneiderman [33], allowing users to interact with a continuous *representation* of the object of interest (data in a collection of cells) via incremental, reversible *operations* (updates, filters) by performing physical actions (clicks, scrolling)—addressing the same two aspects that were problematic for databases. Shneiderman attributes the success and usability of spreadsheets to its direct manipulation capabilities. Despite these substantial benefits and their immense popularity, spreadsheets suffer from severe limitations [27]: they do not support large datasets beyond main memory limits, and cannot encapsulate sophisticated operations, requiring complex workarounds (e.g., a collection of VLOOKUPs to implement a foreign-key join).

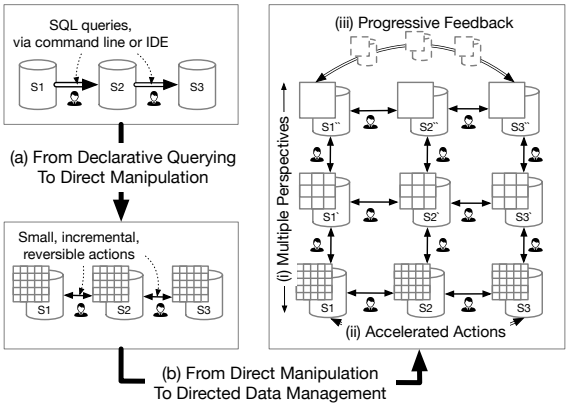

Figure 1: Steps towards Directed Data Management.

Our overall goal is to *bring the usability benefits of a spreadsheet-like interface—the direct manipulation capabilities and ease-of-use—to relational databases*, thereby providing a solution to database usability for ad-hoc interactions. We have been working towards this goal for the past two years. Our first finding is that adding spreadsheet-based direct manipulation capabilities to databases is not straightforward, in part due to fundamental incompatibilities between direct manipulation and the declarative querying paradigm. While direct manipulation has the potential to greatly increase the usability of databases, we discovered that direct manipulation capabilities, while necessary, are not sufficient to support ad-hoc interactions at scale. Our second finding is that we need to go beyond direct manipulation to a new paradigm, one we are calling *directed data management*. Next, we describe these findings, and the challenges they introduce.

From Declarative Queries to Direct Manipulation. To make databases more usable, we need to equip databases with direct manipulation capabilities via a spreadsheet-like frontend, while applications requiring SQL access to the database continue to have such unfettered access to the underlying data, as shown in Figure 1a. Unlike traditional data management where there are irreversible transitions between database states (depicted as drums) based on coarse-grained SQL queries on entire relations (depicted as unidirectional

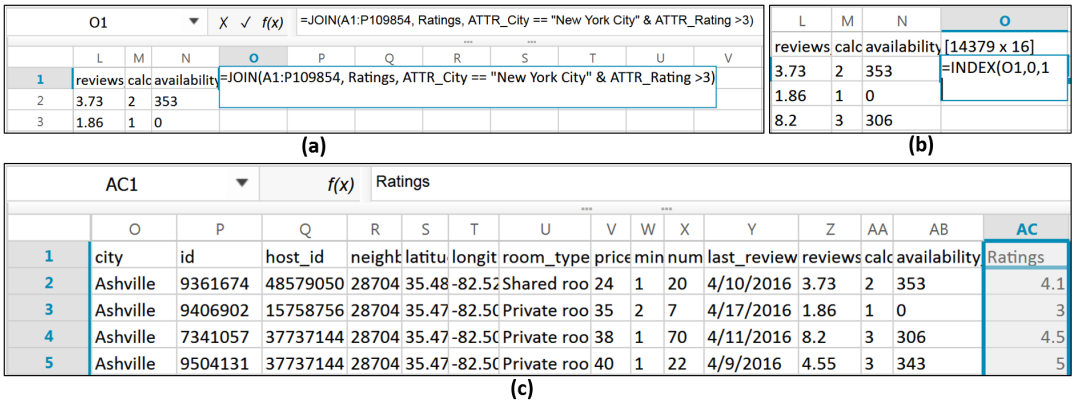

Figure 2: (a) user adds the JOIN formula, (b) user adds the INDEX formula to retrieve the JOIN results, and (c) results displayed.

thick arrows), our goal is to support a spreadsheet representation (depicted as a grid) as well as the database state at each juncture, and supports incremental, reversible, and fine-grained operations (depicted as bidirectional thin arrows) to change the representation along with data. However, we found that databases *do not natively support an ad-hoc spatial representation, with operations that impact one or more cells at a time, referred to by their position on the sheet.* From a representation standpoint, data on spreadsheets is organized in an ad-hoc, task-dependent manner, with tables interspersed with whitespace and formulae; databases instead organize data in uniform, unordered relations. Moreover, the organization heavily relies on position—but maintaining position within a database is hard: adding or deleting a row can lead to cascading updates to row numbers of subsequent rows, necessitating indexing structures that can maintain positional information efficiently. From an operation standpoint, direct manipulation encourages the embedding of formulae along with data, leading to new optimization challenges: we need to provide databases with the ability to execute a complex network of formulae, prioritizing for what the user is seeing. Overall, we need to equip databases with what we're calling *representation*-awareness and *operation*-awareness. We describe these challenges in detail in Section 3.

From Direct Manipulation to *Directed* Data Management. While direct manipulation greatly increases the usability of databases, we discovered that it is not as effective at scale. When dealing with data that is extremely large, direct manipulation is only adequate for certain simple actions, like editing a cell or deleting a row that is already visible, or adding a new row or column. On the other hand, it is hard to envision scrolling through a billion row spreadsheet manually, without some ability to navigate to the rows of interest. For example, our biology collaborators at Mayo Clinic, while thrilled at the ability to examine their large genomics dataset in a spreadsheet, acknowledged that they wouldn't be able to scroll through all of it, limiting its usefulness. Cockburn and others argue that mentally assimilating and manipulating a large information space can lead to cognitive and mechanical burdens on users [13, 22]. It is also hard to expect users to express computation on a billion row spreadsheet, such as dragging a formula through the entirety of a column, which may span all billion rows. In fact, a single relational operator may, in this case, be able to replace a number of formulae proportional to the size of the data. Finally, some formulae can take a long time to execute with no feedback to the user about the status of the computation.

To address these limitations, we identified three extensions to direct manipulation that help us make a substantial leap towards our original research goal. We call this new paradigm *directed data management.*[1] Directed data management extends the direct manipulation principles in various ways (in *italics*)—see Figure 1b, allowing users *(i)* to interact with a *multi-perspective* continuous representation of the object of interest—that is, allowing operations at various organizations and granularities of the data *(ii)* via *accelerated* actions—allowing users to skip fine-grained steps if necessary, using coarse-grained operations (such as relational algebra and SQL), *(iii)* by performing operations whose impact is *progressively visible*—that is, the system constantly provides partial results to the user, even for expensive computation. We describe the challenges underlying directed data management in Section 4.

DATASPREAD**: A Progress Report.** We have been developing a system, DATASPREAD, addressing some of the challenges underlying directed data management. We report on our current prototype, as well as the lessons learned from this experience in Section 2.

Related Work. The only project with a similar goal of trying to improve database usability using spreadsheets was the CONTROL project from the late 90s at UC Berkeley [31]; however, their goal was to provide read-only access to spreadsheet data, as opposed to allowing edits, making the spreadsheet less powerful as an ad-hoc data manipulation or presentation tool—as such, their focus was on providing rapid feedback for certain expensive operations, such as aggregation [17], filters, and sorts [32], which we can build on.

There have been some efforts to support one-way import of data from databases or the web into spreadsheets, including Power Pivot and Query [39], Zoho [36], ExcelDB [38], Airtable [35], and Blockspring [37], or the import of database operators into spreadsheets [34, 8]; however, they do not address the scalability limitations of present-day spreadsheet tools. Similarly, there has been work on exporting spreadsheet functionality into a database, such as work by Oracle [41, 40]—without providing any frontend capabilities, and on extracting data from spreadsheets [12].

Other work has also targeted database usability [19] by making it easier to issue SQL queries, via query builders [5, 10], spreadsheet operations [25], and query sharing tools [20, 11, 15].

2. DATASPREAD: CURRENT PROGRESS

We have been building DATASPREAD—our attempt towards realizing the goals of directed data management. The DATASPREAD prototype has a scalable web-based frontend, based on the ZK framework [3], and a PostgreSQL backend—but both the frontend and backend have undergone substantial changes. The source code, documentation, and user guide of the latest version of DATASPREAD can be found at http://dataspread.github.io. As in traditional spreadsheets, users can use DATASPREAD to interactively manage data and computations (in the form of formulae). Our initial goal in developing DATASPREAD was to develop a direct manipulation in-

[1] Directed data management is not just reliant on direct manipulation—computation can be "directed" by the user and the system: users "direct" computation at a higher level if needed, aided by automation, so the system "directs" computation as well.

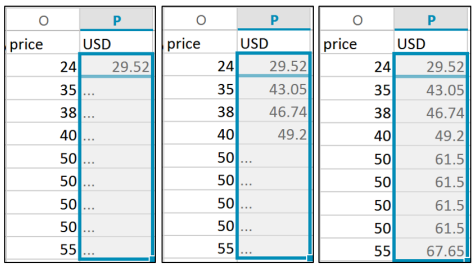

Figure 3: Asynchronous Formula Execution: (a) user writes a formula and makes copies of it by dragging the autofill handle, (b) display of partial results, and (c) formula execution completed.

terface for databases, for which we implemented the following features. All the examples discussed in this paper refer to a realtor working on a dataset of Airbnb listings [14].

Direct Manipulation for Databases. In order to bring direct manipulation capabilities to databases, we developed a spreadsheet-like interface, which can be used to manage and interact with the back-end database—users can organize, view, and manipulate data laid out on the spreadsheet. For example, the relational table in Figure 2c maintains a two-way correspondence between the interface and the underlying database, such that any operations on the interface are translated into operations on the corresponding database relations, and vice-versa. Thus, a user can use traditional spreadsheet operations such as updating a cell's value to update the corresponding relation. We describe the challenges involved in identifying an efficient backend organization of ad-hoc heterogeneous data on the spreadsheet-like interface, and in maintaining indexes by position in Section 3.

Scalability via Paging and Asynchronous Computation. An interesting consequence of providing database support for a spreadsheet-like interface is *scalable spreadsheets*. DATASPREAD inherits the scalability of databases, and can operate on arbitrarily large datasets beyond main-memory limits [9]. It achieves scalability by fetching data on-demand from persistent storage, using indexes based on position, when triggered by a user action (like scrolling) or from a system action (like formula calculation).

Scalable spreadsheets warrant new computational techniques to maintain interactivity. Traditional spreadsheet software uses computation that is *synchronous*, i.e., the system is unresponsive if any computation is ongoing, and users can only interact with the system after the computation is complete. Instead, DATASPREAD prioritizes interactivity by adopting *asynchronous computation*, where control of the spreadsheet is returned back to the user while computation happens in the background. For example, in Figure 3, when the realtor drags the formula autofill handle from top to bottom, the cells that are involved in the computation are marked using an ellipsis ("...") indicating ongoing computation. The user can interact with the other cells in the interface, as usual, while the computation is performed in the background.

While using scalable spreadsheets to work with large datasets, we quickly realized the limitations of direct manipulation techniques, as discussed in Section 1. We now discuss our efforts towards addressing these limitations via directed data management.

Navigation Between Perspectives. On very large datasets, scrolling within the spreadsheet interface to find desired data can be tedious. To enable users to quickly navigate to an area of interest, we introduce a hierarchical navigation panel. This panel provides a high-level overview of the data, and supports zooming in and out of data at various granularities—to access the desired data, users can pan across the overview and skip over irrelevant regions, without having to scroll endlessly. The panel also simplifies formula specification by allowing users to select formulae operands directly using the data overview, without having to manually select the relevant data. For example, in Figure 4, the realtor zooms into a specific price bracket (61-85) within the hierarchical navigation panel to get a fine-grained view and then specifies a COUNTIF formula to compute the number of listings per price bracket with more than two reviews. DATASPREAD presents the results of the formula for each bracket in the form of charts.

Figure 4: DATASPREAD Navigation Panel. (A) multi-perspective representation, (B) accelerated action through formula (chart) computation, (C) spreadsheet-like interface, and (D) navigation context.

Accelerated Actions. The cell-oriented formula model of spreadsheets provides limited expressiveness, especially while working with tabular data. To address this, we introduced relational operators and SQL, via the following functions: union, difference, intersection, crossproduct, join, filter, project, rename, and sql. These newly introduced functions work along with the traditional spreadsheet formulae and refer to relations in the underlying database as well as tabular regions of data on the interface. For example, our realtor wants to see all the listings in NYC with an average rating above some threshold along with their ratings—this corresponds to a *natural join* of a range with the ratings table, implemented using two formulae: JOIN(A1:P100000, Ratings, City=="NYC" & Rating > 3) and multiple instances of INDEX(O1,0,*), one for each cell in the result of the join (see Figure 2).

3. TOWARDS DIRECT MANIPULATION

As we argued in Section 1, direct manipulation on a spreadsheet-like interface hinges on human-interpretable *representations* of the state of the data and intuitive *operations* that allow transitions between states: users *"view a concrete, visible representation of data as a tabular layout of cells, upon which to perform incremental operations"* [29]. As it turns out, adding support for direct manipulation to a database requires every layer of the database stack to be more aware of human-centered needs.

On the one hand, to allow users to interact with a "concrete, visible representation of data in a tabular layout of cells", this requires the data to be *ordered* in either dimension, and *positioned* on a grid. Such a task-specific representation is necessarily *irregular*, with multiple modularized regions of data, interspersed with whitespace, all on a single sheet, along with *embedded* computation in the form of formulae, many of which are similar to each other. Unfortunately, these notions of ordered, positional, irregular, and

query-embedded data representations are in sharp contrast to the relational model in a database, which is set-oriented, not organized in a grid, regular (i.e., each tuple has the same set of attributes), and often normalized and thus disallows redundancy (no derived values). Thus, in order to support direct manipulation, databases must be equipped with *representation-awareness*—a new notion.

On the other hand, to support rapid, incremental, and reversible operations, we should provide *small operations* with *guaranteed responsiveness* (e.g., sub-second) and *undo* capabilities, to cater to the fact that users are impatient, and inadvertently end up making errors. Once again, we find that these notions are very different from the declarative querying paradigm in a database, where a query is a "batch" command, with a non-guaranteed response time and irreversible operations. To support direct manipulation, databases must also be equipped with *operation-awareness*.

3.1 Supporting Representation Awareness

We now consider some of the key challenges in bringing representation-awareness to databases.

Irregularity and Embedded Queries. Direct manipulation of data requires the flexibility to develop a concrete and visual representation of data, for a specific activity. For example, the realtor may want to organize the Airbnb listings by region to prepare a report. The database should persist this layout, along with the data, enabling the realtor to recall where specific data or analysis is located in order to complete a task, and create multiple such layouts for different tasks. This imposes a new challenge: *How do we support irregular layouts of data using the rigid relational model*?

Our insight here is that we may use a hybrid representation by carving out dense tabular regions of the spreadsheet, directly stored as database tables, and the remaining sparse cells, stored as key-value pairs, with the position as the key. We can also take into account access patterns, e.g., via formulae. Unfortunately, identifying the optimal hybrid representation is NP-HARD; however, we developed near-optimal approaches that yield substantial reductions in storage (up to 50%), and formula evaluation time (up to 50%) [9].

Since users embed computation (formulae and queries) in these irregular layouts, we can take advantage of that to further optimize the representation. For example, our realtor may add a derived column which calculates the *percentile* of each listing by its rental price. At first glance, we may consider these queries as *views*—however, we often have as many views (formulae) as there is data—in this case, one per listing. From a representation standpoint, we may be able to automatically identify that many of the formulae are very similar to each other and use that to compress them and evaluate them together as a unit. This introduces an additional challenge: *for these formulae, do we store the results (which speeds up retrieval) or not (which saves storage)*? We may want to prioritize the formulae that are frequently viewed for materialization.

Order and Position. In a spreadsheet-like interface, users can organize the data in very specific ways, tailored to their task. For example, our realtor can manipulate the order of the listings (e.g., by sorting the listings by district), and then drag the rows corresponding to "Queens" to be next to "Brooklyn" to compare those listings on other attributes. Users expect these changes to be persistent and expect to be able to able to refer to the data by position, e.g., by scrolling to or selecting a region of interest. We call these orders as *implicit*, as they are user-defined and do not correspond to any natural ordering based on attribute values. Thus, *how do we support the notion of order (both implicit and explicit) and position within a database, which is unordered in both rows and columns*? One simple approach is to simply support this by adding an attribute that captures the position, e.g., the row, for each row in the interface. However, even a small change, such as the addition of a row, can cause the cascading updates of the row numbers of all subsequent rows.

To address this, we can encode the position attribute via monotonically increasing proxies such that the insertion or deletion does not impact subsequent tuples. Then, we can use a hierarchical B+tree-like indexing structure that appends each node with the counts of nodes in the subtree below, allowing us to identify the tuple in the k-th position in $O(\log n)$. In practice, our encoding and indexing scheme ensures interactivity ($\approx$a few ms) for billions of cells [9].

While our hierarchical indexing structure allows for efficient access and updates for small numbers of rows, columns, or cells, it does not admit the natural manipulation of implicit orders. For example, consider the example earlier where the listings of "Queens" and "Brooklyn" were moved next to each other—here, the time taken is proportional to the number of listings moved. *Can we develop an indexing structure that can maintain such orders in time that is instead dependent on the number of operations performed?* One preliminary insight is to support *move* and *swap* operations by manipulating the corresponding sub-trees; however, this requires new balancing techniques for our hierarchical indexing structure.

Additionally, users can choose to order the same data in different implicit or explicit orders. For example, our realtor can lay out the listings in two different orders, one where the rows of "Queens" are next to "Brooklyn" and one where they are next to "Manhattan". *How do we efficiently support these orders that are only slightly different from each other*? To support such orders, we not only need to persist them in some way, but also issue updates as the underlying data changes. For example, row deletions need to be propagated to all orders—which is challenging for implicit orders since they do not admit any attribute-value based ordering. For this, our approach is to enhance our indexing structure to enable reverse lookups, using tuple identifiers to maintain the implicit order.

3.2 Supporting Operation Awareness

We now discuss how to enable *operation awareness* in a database, via *small operations* with *guaranteed responsiveness*. Moreover, these operations can often be done *collaboratively* by multiple users manipulating the data in-sync. With errors creeping in, there is often a need to *undo* operations, or revert to old versions.

Guaranteed Responsiveness on Small Operations. Direct manipulation presents new challenges for query optimization and execution. Rather than writing complex SQL queries, users perform computations in small, incremental steps using formulae, embedded along with data—oftentimes as many formulae as items of data. Formulae can refer to the results of other formulae, introducing dependencies. The *scale of queries and the notion of dependencies* introduce new challenges, not found in traditional query optimization. Even on present-day spreadsheets that have many formulae, formula computation ends up leading to system unresponsiveness spanning from minutes to even hours [27].

To facilitate formula computation, we need to capture formula dependencies in a dependency graph, which could be arbitrarily deep and wide. One open question here is: *how and where do we maintain this graph as users add or delete formulae*? Given such a representation, we can make formula computation more responsive if we incorporate two techniques: *asynchronous computation*—as described in Section 2, allowing computation to happen in the background while users can still interact with the sheet, and *lazy computation*—since users are only viewing a window at a time, prioritizing for what they are seeing over what they are not. While enabling these techniques, we need to ensure that users see a *consistent view* of the analysis—no stale values should be shown.

Enabling both of these techniques leads to a number of challenges: *(a) How do we identify and optimize for the dependencies*? To prioritize for the user window, we need to quickly traverse the large dependency graph to find the formulae that are impacted. One approach is to store a compressed version of this dependency graph that can be traversed efficiently to identify dependent formulae, with minimizing false positives (identifying a formula as a dependent, when it is not), but no false negatives. *(b) How do we identify and optimize for the redundancies*? When executing these formulae, since many of them have a similar structure and refer to the same data, we can share access and computation. This challenge, like the previous one, is reminiscent of multi-query optimization. However, the new structural characteristics, in terms of the dependencies and redundancies, make the challenges distinct and novel, necessitating new techniques.

Collaboration and Undo. The ACID properties of databases are defined with correctness in mind, as opposed to their usability. For example, concurrency needs to be revisited in a direct manipulation interface—*how can users "lock" a portion of data for editing in a way that does not admit interference from others, and how can they "commit"*? This requires new interpretable visual signifiers to convey the notion of locks. Since we are dealing with humans, *how do we deal with transactions that take a long time or never complete*? Moreover, since errors are inevitable—errors on spreadsheets have led to huge financial losses in several financial organizations [1]—we need to support some form of versioning and/or checkpointing; undo would be a valuable first step, but often it is necessary to capture the entire lineage of data on the interface.

4. TOWARDS DIRECTED DATA MGMT.

Since a direct manipulation interface is unable to support certain types of operations at scale, we propose a new paradigm called *directed data management*, which extends direct manipulation by including support for both user-directed and system-directed analysis of very large datasets. We now discuss the motivation and the underlying challenges.

Enabling Direct Manipulation at Scale. Even though spreadsheets provide direct manipulation capabilities (see Section 1), given the scale of data, such an interface introduces a discontinuity between the information displayed in its limited window, creating a cognitive burden for the users in understanding the overall structure of the data, and navigating through it [22]. We aim to instead allow users to make sense of data using different *perspectives*, reducing the discontinuity, and allowing users to rapidly jump between perspectives; for example, between a bird's eye view of the entire interface, and a close-up, as in online maps.

Then, to manipulate data at scale, small, incremental actions to accomplish a given task (e.g., manually selecting a range of a million cells, or filling out a column of a million formulae) can be tedious and error-prone. Simplifying or *accelerating such actions* can greatly improve the usability of the system.

Finally, even with actions that are easier to express, sometimes the computation time of certain operations (e.g., a formula that aggregates a billion cells, or a sort operation) makes it impossible to receive feedback immediately. Studies in HCI have shown that delays of even half a second can severely affect the user experience, and may lead to users abandoning the intended task [26]. Thus, there is a need to provide an immediate response to the user through *progressively visible* feedback.

Multi-Perspective Representation. To reduce the impact of information overload with data at scale, we propose to enable data to be presented using multiple perspectives—where each perspective refers to both a way of organizing the data, and a specific granularity (or "zoom" level) for that organization. For example, the realtor may choose to view the listings by price or reviews, or both, at various granularities, and may choose to view an aggregate corresponding to the total number of rentals per month. Each perspective offers a unique vantage point. Supporting multiple perspectives can be challenging since it impacts the interface design, as well as the entire database stack.

From the interface standpoint, the primary challenge is to create an interpretable representation that can be used in conjunction with the traditional spreadsheet-like interface, allowing users to interact with both interfaces, enabling rapid exploration and drill-down. Our *navigation panel* (see Figure 4) is a step towards this idea by enabling users to work with data at different granularities on a hierarchical interface. Users can view data at different levels by "zooming in and out". *How can we automatically construct this hierarchical interface that adapts to various data types and user needs*? Presently, we use a simple histogram-based equi-depth binning strategy. Automated techniques can be augmented by learning user preferences through examples, as in Excel's flash-fill [16].

From the backend standpoint, *when and how do we construct the hierarchical interface*? One option is to do so *lazily*, when the user explores the data organized in a certain way and granularity. However, this may take a long time—in which case we may need new storage models and indexing schemes for data at various granularities. For this, pre-materialized data cubes, as well as incremental approaches such as database cracking seem promising.

Accelerated Actions. The goal of accelerated actions is to allow users to move to the desired state by skipping a cumbersome sequence of small actions. Such accelerated actions can be either user or system-directed, and can be issued within a given perspective, as well as across perspectives. As a first step, as discussed in Section 2, we support SQL queries within a given perspective to allow users to go beyond basic spreadsheet-like direct manipulation to operate on collections of data at a time. For example, to filter out records that occur between two dates, our realtor can simply use a relational operation, instead of either manually selecting the rows of interest, or using the advanced spreadsheets filtering capabilities that requires multiple interactions. However, to display the result of the relational operation, users still have to refer to the results via an INDEX function that needs to be dragged across a rectangular region (as in Figure 2) which can be tedious.

The challenge, therefore, is: *how do we design intuitive and interpretable interactions within a given perspective, complementary to declarative querying*? Our navigation panel, discussed in Section 2, introduces new actions to aggregate, format, and organize data within a given perspective. For example, in Figure 4, computation of the charts for each price bucket is an accelerated action that is directed by the user that would otherwise require manual selection of the region in each price bucket, followed by formulae that operate on those regions. We can further enrich the interactions by allowing the charts to be used as visual query interfaces in order to support ad-hoc formula specification. For example, the realtor can use a slider to move along the histogram in Figure 4 to select different prices within a price bucket, without typing out the corresponding formula.

The next challenge is to support accelerated actions across perspectives. For example, if the user zooms out, DATASPREAD recomputes the charts from the new perspective. Multi-perspective navigation can be overwhelming and users can lose context due to abrupt changes in view. *How do we design interface cues to provide users context for these transitions*? For example, by displaying a navigation history we can provide users a context of where they were and where they are now. Then, to enable efficient query

execution for actions across perspectives, we can adopt view materialization schemes, so that repeated actions are not recomputed, while staying within a storage budget.

There are other preliminary avenues to support accelerated actions. For example, having queries or formulae embedded along with the data can be used by the system to identify errors or recommend reorganization/cleaning actions. Example-driven interactions are another rich avenue whose potential has only been explored within spreadsheets for data extraction [16, 23]. Our investigation of pain points in spreadsheets [27] reveals a lack of knowledge of typical workflows as a primary contributor to errors in formulae: auto-suggestions of next steps, or providing explanations for formulae can greatly increase understanding and efficiency.

Progressively Visible Feedback. In addition to the interactive computation ideas discussed in Section 3.2, our goal is to quickly provide users with approximate results that improve over time [17, 30]. For example, the realtor may want to sort millions of listings by price, or compute an average price of the listings. As the computation happens, the system can display the progress to the user incrementally. However, it is not clear *how to display this information in a way that maximizes user understanding and allows users to make decisions early*. Progressively updating the interface while presenting a stable and consistent view can be challenging. A natural approach would be to draw samples on the fly and update the interface periodically, with probabilistic error guarantees. One approach for formula computation is to display a progress bar as it is being computed (as the cell itself), with the current best estimate being overlaid on top.

From a query optimization standpoint, *how do we identify optimal sampling strategies for progressive feedback, optimizing for the many computations currently underway*? We need to trade-off the benefit of a sample across these computations. Moreover, traditional progressive approximation approaches [17, 30] only provide error guarantees for summary statistics. However, for operations that impact the position of data, such as *sort*, *how do we capture and display the positional uncertainty of the data*? For example, for a sort operation, we can leverage the cardinality statistics of the sort column to model the positional uncertainty. To be able to access data by position as it is sorted, we need to construct the positional index progressively—one strategy can be to bulk load data in increments, and reconstruct the indexes at each increment.

5. CONCLUSION

We introduced *directed data management* as a promising solution to the problem of database usability by building on and extending the principles of direct manipulation, as embodied by spreadsheets. We described a number of research challenges in directed data management, and our DATASPREAD prototype. We hope that database and HCI researchers alike will contribute to addressing the challenges underlying this rich and exciting research direction.

6. REFERENCES

[1] EuSpRIG Horror Stories. http://www.eusprig.org/horror-stories.htm.

[2] How finance leaders can drive performance. https://enterprise.microsoft.com/en-gb/articles/roles/finance-leader/how-finance-leaders-can-drive-performance/.

[3] ZK Spreadsheet. https://www.zkoss.org/product/zkspreadsheet.

[4] D. Abadi et al. The beckman report on database research. *Communications of the ACM*, 59(2):92–99, 2016.

[5] A. Abouzied et al. Dataplay: interactive tweaking and example-driven correction of graphical database queries. In *UIST*, 2012.

[6] D. Agrawal et al. Challenges and opportunities with big data. 2011.

[7] R. Agrawal et al. The claremont report on database research. *ACM Sigmod Record*, 37(3):9–19, 2008.

[8] E. Bakke and D. R. Karger. Expressive query construction through direct manipulation of nested relational results. In *SIGMOD*, 2016.

[9] M. Bendre et al. Towards a holistic integration of spreadsheets with databases: A scalable storage engine for presentational data management. In *ICDE*, April 2018.

[10] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: A survey. *Journal of Visual Languages & Computing*, 8(2):215–260, 1997.

[11] U. Cetintemel et al. Query Steering for Interactive Data Exploration. In *CIDR*, 2013.

[12] Z. Chen et al. Senbazuru: A prototype spreadsheet database management system. *PVLDB*, 2013.

[13] A. Cockburn, A. Karlson, and B. B. Bederson. A review of overview+ detail, zooming, and focus+ context interfaces. *ACM Computing Surveys (CSUR)*, 41(1):2, 2009.

[14] M. Cox. Inside Airbnb. http://insideairbnb.com/get-the-data.html.

[15] H. Gonzalez et al. Google fusion tables: web-centered data management and collaboration. In *SIGMOD*, 2010.

[16] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN Notices*, volume 46, pages 317–330. ACM, 2011.

[17] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Acm Sigmod Record*, volume 26, pages 171–182. ACM, 1997.

[18] S. Idreos et al. dbtouch: Analytics at your fingertips. In *CIDR*, 2013.

[19] H. V. Jagadish et al. Making database systems usable. In *SIGMOD*, pages 13–24. ACM, 2007.

[20] N. Khoussainova et al. A Case for A Collaborative Query Management System. In *CIDR*. www.cidrdb.org, 2009.

[21] N. Khoussainova et al. SnipSuggest: Context-aware autocompletion for SQL. *VLDB Endowment*, 4(1):22–33, 2010.

[22] S. Kruck, J. J. Maher, and R. Barkhi. Framework for cognitive skill acquisition and spreadsheet training. *Journal of Organizational and End User Computing (JOEUC)*, 15(1):20–37, 2003.

[23] V. Le and S. Gulwani. Flashextract: a framework for data extraction by examples. In *ACM SIGPLAN Notices*, volume 49, 2014.

[24] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. *VLDB Endowment*, 8(1), 2014.

[25] B. Liu and H. Jagadish. A spreadsheet algebra for a direct data manipulation query interface. In *ICDE*, pages 417–428. IEEE, 2009.

[26] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE TVCG*, (1):1–1.

[27] K. Mack et al. Characterizing scalability issues in spreadsheet software using online forums. In *SIGCHI*, 2018.

[28] A. Nandi, L. Jiang, and M. Mandel. Gestural Query Specification. *VLDB Endowment*, 7(4), 2013.

[29] B. A. Nardi and J. R. Miller. *The spreadsheet interface: A basis for end user programming*. Hewlett-Packard Laboratories, 1990.

[30] S. Rahman et al. I've seen enough: incrementally improving visualizations to support rapid decision making. *PVLDB*, 2017.

[31] V. Raman, A. Chou, and J. M. Hellerstein. Scalable spreadsheets for interactive data analysis. In *1999 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1999.

[32] V. Raman et al. Online dynamic reordering for interactive data processing. In *VLDB*, volume 99, pages 709–720, 1999.

[33] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16(8):57–69, 1983.

[34] J. Tyszkiewicz. Spreadsheet as a relational database engine. In *SIGMOD*, pages 195–206. ACM, 2010.

[35] https://www.airtable.com/. Airtable.

[36] https://www.zoho.com/. Zoho Reports.

[37] http://www.blockspring.com/. Blockspring.

[38] http://www.excel-db.net/. Excel-DB (retrieved March 10, 2015).

[39] C. Webb. *Power Query for Power BI and Excel*. Apress, 2014.

[40] A. Witkowski et al. Advanced SQL modeling in RDBMS. *ACM Transactions on Database Systems (TODS)*, 30(1):83–121, 2005.

[41] A. Witkowski et al. Query by excel. In *VLDB*, pages 1204–1215, 2005.