Expressive Query Construction through Direct Manipulation of Nested Relational Results

Eirik Bakke
MIT CSAIL
ebakke@csail.mit.edu

David R. Karger
MIT CSAIL
karger@csail.mit.edu

ABSTRACT

Despite extensive research on visual query systems, the standard way to interact with relational databases remains to be through SQL queries and tailored form interfaces. We consider three requirements to be essential to a successful alternative: (1) query specification through direct manipulation of results, (2) the ability to view and modify any part of the current query without departing from the direct manipulation interface, and (3) SQL-like expressiveness. This paper presents the first visual query system to meet all three requirements in a single design. By directly manipulating nested relational results, and using spreadsheet idioms such as formulas and filters, the user can express a relationally complete set of query operators plus calculation, aggregation, outer joins, sorting, and nesting, while always remaining able to track and modify the state of the complete query. Our prototype gives the user an experience of responsive, incremental query building while pushing all actual query processing to the database layer. We evaluate our system with formative and controlled user studies on 28 spreadsheet users; the controlled study shows our system significantly outperforming Microsoft Access on the System Usability Scale.

This work was previously published at SIGMOD 2016.

1. INTRODUCTION

Four decades after Query by Example [51], the broad problem of Making Database Systems Usable [25] remains open. Technical users still interact with relational data through hand-coded SQL, while non-technical users rely on restrictive form- and report-based interfaces tailored, at great cost, for their specific database schema [32, 27, 4]. Queries that involve "complex aggregates, nesting, correlation, and several other features remain on a tall pedestal approachable only by the initiated" [23]. Simple report queries traversing one-to-many relationships in the database schema, such as retrieving "a list of parts, and for each part a list of suppliers and a list of open orders", are painful to define for programmers and largely inaccessible to end users.

Meanwhile, users from a wide range of backgrounds seem happy, indeed eager, to interact with their data if it is served to them in spreadsheet form. "Export to Excel", the joke goes, "is the third most common button in data and business intelligence apps... after OK and Cancel"[1]. Spreadsheets lack basic database functionality such as joins and views, but demonstrate the great value of usable, general-purpose data manipulation tools [4].

Shneiderman [43] attributes the usability of the spreadsheet to its nature as a *direct manipulation* interface. The properties of such an interface include "visibility of the object of interest", "rapid, reversible, incremental actions", and "replacement of complex command language syntax by direct manipulation of the object of interest". Shneiderman paraphrases Harold Thimbleby: "The display should indicate a complete image of what the current status is, what errors have occurred, and what actions are appropriate."

We agree with Liu and Jagadish [35] that a successful solution to the visual query language problem must come in the form of a spreadsheet-like direct manipulation interface. In particular, we consider three requirements that have yet to be met in a single user interface design:

R1. *Query specification through direct manipulation of results.* The user should build queries incrementally through a sequence of operations performed directly on the data in the database, as seen through the result of each intermediate query [35]. In Shneiderman's terms, the *object of interest* is not the query, but the data, as when working with a spreadsheet.

R2. *The ability to view and modify any part of the current query, including operations performed many steps earlier, without redoing subsequent steps or departing from the direct manipulation interface.* This is tricky in light of R1, because the user will be looking at and manipulating the *result* of a query rather than an actual query expression. The mapping between the two is not obvious. [35]

R3. *SQL-like expressiveness from within the direct manipulation interface.* R1 and R2 can be trivially met if only simple queries are allowed. For example, Excel's *filter* feature works by direct manipulation of results, and allows its complete state to be viewed and modified from within the same interface, but supports only basic selection queries. To compete with SQL, a visual query system should allow the user to express any query commonly supported by SQL implementations, including arbitrary (multi-block) combinations of operations such as joins, calculations, and aggregations.

In this paper, we present SIEUFERD (pronounced *soy-fird*), the first visual query system to meet all of the requirements above in

Short version prepared for the 2016 KDD IDEA Workshop.
To cite, please refer to our full SIGMOD paper [5].

KDD 2016 Workshop on Interactive Data Exploration and Analytics (IDEA'16). August 14th, 2016, San Francisco, CA, USA.

[1] http://www.powerpivotpro.com/2012/03/the-3rd-most-common-button-in-data-apps-is

Direct Manip.	*Query Representation*	*Year*	*System*		*R1*	*R2*	*R3*	*Unrestricted Nested Results*
Yes	Overlaid on Result	2014	GBXT	[2]	X	X		X
		2012	DataPlay	[1]	X	X		X
		2006	Tabulator	[8]	X	X		X
		2002	Polaris/Tableau	[46]	X	X		
	Spreadsheet Formulas	2016	Object Spreads.	[37]	X	X		X
		2010	Spreads. as DB	[48]	X	X		
		2005	A1	[30]	X	X		X
		1997	OOF Spreads.	[15]	X	X		X
		1994	Forms/3	[10]	X	X		
	Exposed Algebraic	2013	Mashroom	[20]	X		X	X
		2011	Wrangler	[29]	X		X	
		1991	TableTalk	[18]	X		X	X
	Hidden Algebraic	2016	Gneiss	[13]	X		X	
		2013	GestureDB	[38]	X		X	
		2010	CRIUS	[41]	X			X
		2009	SheetMusiq	[35]	X			
		2008	AppForge	[50]	X			X
		1989	R^2	[22]	X		X	X
No	Diagram-based	2014	VisualTPL	[14]				X
		2009	App2You	[31]				X
		2005	QBB	[40]				
		2002	QURSED	[39]				X
		1990	QBD	[3]				
	Form-based	2008	Form Cust.	[28]				
		1998	QBEN	[36]				X
		1997	ESCHER	[49]				X
		1989	PERPLEX	[44]				
		1977	QBE	[51]				

Table 1: Summary of related systems, evaluated as visual query interfaces. R1 is indicated where some class of queries can be initially specified by direct manipulation of results. R2 is indicated where all parts of such queries can subsequently be modified through similar means. R3 is indicated where the same class of queries is relationally complete and supports aggregation in arbitrary multi-block queries.

a single user interface design. The key insight is that given a suitable data model for results, the complete structure of a query can be encoded in the schema of the query's own result. This in turn allows the user interface to display the query and its result in a single visual representation, which can then be manipulated directly to modify any part of the query. Specifically, we allow queries to produce results from the nested relational data model [24, 33], and display results using a nested table layout [6]. In our visual representation, the header area of the result's nested table layout encodes the structure of the query, which can then be manipulated using spreadsheet idioms such as formulas and filters. The use of nested results affords a natural visualization of operations such as joins and aggregation, and allows the user to see, in context, intermediate tuples produced in any part of the query.

Using our system, the user can express a relationally complete [16] set of query operators plus calculation, aggregation, outer joins, sorting, and nesting [5, Appendix A]. This covers the full set of query operators generally considered as the minimum to model SQL [7, 21], and expresses, for example, all `SELECT` statements valid in SQL-92.

In an initial formative user study, 14 participants were able to solve complex query tasks with a minimal amount of training, with many expressing strong levels of satisfaction with the tool. In a second, controlled study, another 14 participants rated both SIEUFERD and the query designer found in Microsoft Access on the System Usability Scale (SUS) [9] after doing a series of tasks on each. Users rated SIEUFERD 18 points higher on average than Access. This corresponds to a 46 percentage point difference on a percentile scale of other studies in the Business Software category.

This work was previously published at SIGMOD 2016 [5].

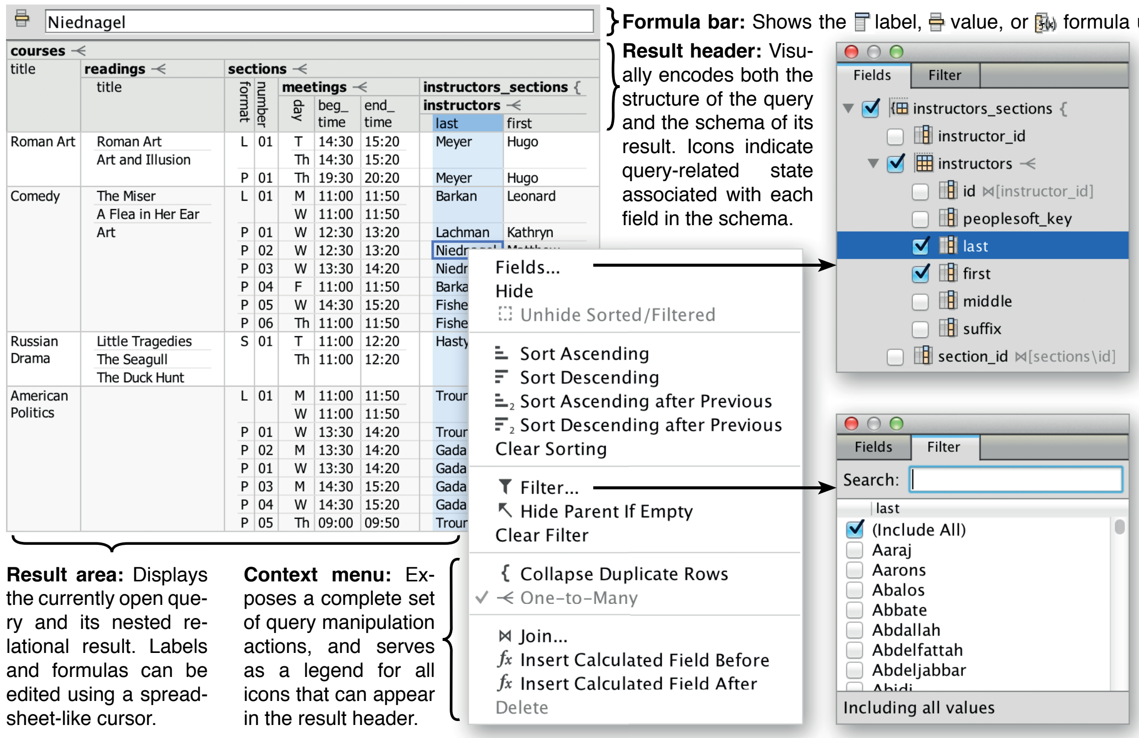
2. RELATED WORK

Visual query systems have been surveyed by Catarci et al. [11] and, recently, El-Mahgary and Soisalon-Soininen [17]. Systems discussed in this section include, in particular, those that employ direct manipulation, nested results, or optimizations for traversing relationships in the database. Table 1 categorizes systems by query representation style, and provides an assessment of each system against the requirements set forth in the introduction.

Besides our core requirements, Table 1 also indicates which systems support nested results, i.e. a graphical equivalent of a hierarchical data model such as XML, JSON, or nested relations. This handles report-style queries that encode multiple parallel one-to-many relationships in a single result, as when retrieving "a list of parts, and for each part a list of suppliers and a list of open orders" [6]. Systems that base their result representation on a single flat table of primitive values, such as Tableau [46], are unable to express such queries. The same tends to hold for any system that takes its input from a single joined SQL query, since multivalued dependencies [19] in the flattened result (PARTS$\twoheadrightarrow$SUPPLIERS and PARTS$\twoheadrightarrow$ORDERS in the preceding example) would interact to produce a pathological number of tuples for even small inputs. Some systems, like Tableau and Gneiss [13], support a restricted form of nesting, where an otherwise flat result table can be grouped into a single-branch hierarchy, or a finite set of such (a *dashboard* in Tableau, or a set of *hierarchical tables* in Gneiss). This still does not handle PARTS$\twoheadrightarrow$SUPPLIERS/ORDERS-type queries from the example above. Tableau, as well as other systems based on the pivot table concept, produce cross-tabulated rather than nested results; these concepts are orthogonal.

We first discuss visual query systems that do not fall in the direct manipulation category. *Form-based* systems originated with Query by Example (QBE) [51], where the user populates a set of empty *skeleton tables* with conditions, variables (*examples*), and output indications. ESCHER [49] and QBEN [36] extend QBE to support nested results, while PERPLEX [44] supports general-purpose logic programming. The ubiquitous search forms of commercial database applications can be seen as restricted versions of QBE tailored for a specific schema; Form Customization [28] generalizes such forms by considering the form designer as part of the query system. In *diagram-based* systems, the user manipulates queries for example through a schema tree or schema diagram, as in Query by Diagram (QBD) [3], Query by Browsing (QBB) [40], QURSED [39], and App2You [31], or through a diagrammatic query plan, as in VisualTPL [14]. The diagram-based query building style is common in commercial tools–Microsoft Access, Navicat, pgAdmin, dbForge, Alteryx etc. The general problem with both form-based and diagram-based interfaces is that users must manipulate queries through an abstract query representation that is divorced from the actual data that is being retrieved. To construct and understand queries, the user must look back and forth between the query representation on one side of the screen and a separate result representation on the other. Thus we do not consider these systems to be direct manipulation interfaces (requirement R1).

In the direct manipulation category, we now consider *algebraic* user interfaces. In such systems, the user builds queries by selecting, one step at a time, a series of operations to be applied to the currently displayed result. Each operation is applied to the result of all previous operations. Formal expressiveness is easy to achieve in algebraic interfaces, since the relevant relational operators can simply be exposed to the user directly. The main problem with algebraic interfaces is that the user has no direct way to, in the words of Liu and Jagadish, "modify an operation specified many steps ear-



courses ≺								
title	readings ≺	sections ≺						
				meetings ≺			instructors_sections {	
							instructors ≺	
	title	format	number	day	beg_time	end_time	last	first
Roman Art	Roman Art	L	01	T	14:30	15:20	Meyer	Hugo
	Art and Illusion			Th	14:30	15:20		
		P	01	Th	19:30	20:20	Meyer	Hugo
Comedy	The Miser	L	01	M	11:00	11:50	Barkan	Leonard
	A Flea in Her Ear			W	11:00	11:50		
	Art	P	01	W	12:30	13:20	Lachman	Kathryn
		P	02	W	12:30	13:20	Niedr	
		P	03	W	13:30	14:20	Niedr	
		P	04	F	11:00	11:50	Barka	
		P	05	W	14:30	15:20	Fishe	
		P	06	Th	11:00	11:50	Fishe	
Russian Drama	Little Tragedies	S	01	T	11:00	12:20	Hasty	
	The Seagull			Th	11:00	12:20		
	The Duck Hunt							
American Politics		L	01	M	11:00	11:50	Trour	
				W	11:00	11:50		
		P	01	W	13:30	14:20	Trour	
		P	02	M	13:30	14:20	Gada	
		P	01	W	13:30	14:20	Gada	
		P	03	M	14:30	15:20	Gada	
		P	04	W	14:30	15:20	Gada	
		P	05	Th	09:00	09:50	Trour	

Formula bar: Shows the label, value, or formula under the selected cell.

Result header: Visually encodes both the structure of the query and the schema of its result. Icons indicate query-related state associated with each field in the schema.

Field selector: Pop-up displaying a tree representation of the query structure, including exact join conditions, centered around the selected field. The selector includes previously hidden fields as well as fields that can be reached through joins over known foreign key relationships.

Filter popup: Allows the user to associate a filter with the currently selected field. The list of values available to filter on is generated automatically using a separate database query. Filters may be associated with either primitive fields or relation fields.

Result area: Displays the currently open query and its nested relational result. Labels and formulas can be edited using a spreadsheet-like cursor.

Context menu: Exposes a complete set of query manipulation actions, and serves as a legend for all icons that can appear in the result header.

Figure 1: The SIEUFERD query interface. To create queries, users start from a simple tabular view of a table in the database and add filters, formulas, and nested relations. The integrated result and query representation is displayed continuously as the user interacts with the data. The particular query above instantiates six database tables (one per nested relation), contains five joins (each child relation against its parent), and is evaluated using five generated SQL queries (one for each one-to-many relationship ≺). This query was constructed purely by checking off the appropriate fields and foreign key relationships in the field selector.

lier without redoing the steps afterwards" [35] (requirement R2). For example, in GestureDB [38], the user has no way to modify a filter on a column that was subsequently used in an aggregation or removed with a projection. Similar problems exist in R^2 [22], AppForge [50], CRIUS [41], and Gneiss [13]. SheetMusiq [35] provides a partial solution by using an algebra where certain operators can commute out of a complex expression for subsequent modification; however, the technique breaks down for expressions enclosed in binary operators such as joins, set union, or set difference. In other systems, the underlying algebraic expression is exposed directly, as in the procedural *data manipulation scripts* of Wrangler [29], the XQuery-like *mashup scripts* of Mashroom [20], or the diagram-based representation in TableTalk [18]. Thus, only the initial query specification can be done through direct manipulation; tweaking and examination of existing queries must be done with a separate, indirect interface.

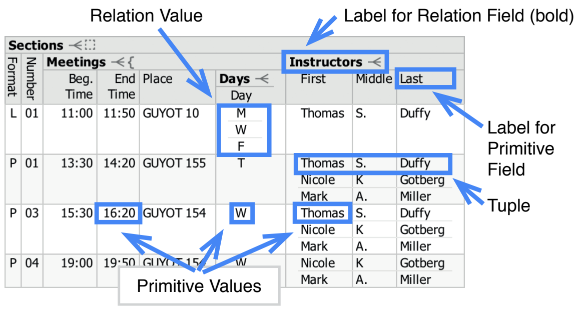
With clever use of formulas, Tyszkiewicz [48] shows that existing spreadsheet products can be considered expressive enough to formulate arbitrary SQL queries. If we consider Excel as a query system, however, only a subset of such queries could be said to be constructible by direct manipulation. Heavy reliance on set-based formula functions such as INDEX, MATCH, and SUMPRODUCT means that spreadsheet formulas soon take the role of a text-based query language, with a vocabulary far removed from that of typical query tasks. This would also be the case for spreadsheet programming systems such as Forms/3 [10], Object Oriented Functional Spreadsheets [15], A1 [30], and Object Spreadsheets [37].

Last, we consider direct manipulation systems that *overlay* their query representation on the result of the same query, with the structure of the query reflecting the visual structure of the result. This solves the mapping problem of requirement R2. The problem is that current such representations are not expressive enough to support arbitrary queries (requirement R3). For example, the direct manipulation interfaces of Tabulator [8] and GBXT [2] support filters and joins over schema relationships, but are unable to express calculation, aggregation, general-purpose joins, or other binary operators. In DataPlay [1], direct manipulation is used only to choose between universal and existential qualifiers. Tableau [46] allows a large class of two-dimensional visualizations to be created and manipulated through direct manipulation of table headers and corresponding axis *shelves*; however, queries involving calculations or binary operators must be configured using a separate interface rather than through direct manipulation. Our own system is the first to achieve SQL-like expressiveness from within a direct manipulation interface based on an overlaid query/result representation.

3. SYSTEM DESCRIPTION

3.1 Overview

Our core query building interface is shown in Figure 1. All user interactions are initiated from the *result area*, which shows the current query's nested relational result, formatted using a nested table layout. In a nested table layout, the table's *header* area visually encodes the schema of the nested result, including which fields are nested under others in the hierarchical schema. Because our system maps all query-related state to specific fields in the result schema, the result's table header simultaneously becomes a visual representation of the query that generated it. A set of icons, carefully designed to allow every aspect of the query state to be represented in the header, is used to augment the information that can be derived from the names and positions of fields.



Sections		Meetings			Days	Instructors		
Format	Number	Beg. Time	End Time	Place	Day	First	Middle	Last
L	01	11:00	11:50	GUYOT 10	M	Thomas	S.	Duffy
					W			
					F			
P	01	13:30	14:20	GUYOT 155	T	Thomas	S.	Duffy
						Nicole	K	Gotberg
						Mark	A.	Miller
P	03	15:30	16:20	GUYOT 154	W	Thomas	S.	Duffy
						Nicole	K	Gotberg
						Mark	A.	Miller
P	04	19:00	19:50	GUYOT 15[illegible]	W	Nicole	K	Gotberg
						Mark	A.	Miller

Figure 2: Terminology of the nested relational data model, illustrated on a nested table layout.

Starting from any selection of fields (columns) in the result area, the user may open a *context menu* of query-related actions, which also serves as a legend for icons that may appear in the result header. Query actions modify the query state, not the data in the database. Whenever a visual query is modified, the system generates and executes one or more corresponding SQL queries to evaluate it, merges the returned flat results into a single nested result, and displays the latter to the user. At the same time, the fields and iconography in the new result's header reflect the updated state of the modified query.

To keep the result layout compact, several aspects of the query state are indicated with icons in the header but are not displayed in full until the user requests it. In these cases we leverage well-established spreadsheet idioms to expose the underlying state. A filter icon (▼) next to a field label indicates the presence of a filter on that field, which can be manipulated by opening the *filter popup* from the context menu. A formula icon (fx) indicates that the primitive field in question is a calculated field with an associated spreadsheet-style formula. The actual formula can be edited using the *formula bar* above the result area, or directly in any non-header cell belonging to the field's column. Finally, as in a spreadsheet, our system allows fields (columns) to be hidden from view and later recalled for inspection. If the hidden field was used for filtering or sorting, or is referenced from a formula, a dashed cell icon (⬚) is shown for the relevant dependent field to indicate that the visible result depends on a hidden portion of the query. Hidden fields can be recalled using the *field selector* popup, which shows an expandable list of available fields, centered around the field it was opened for. The field selector also serves to suggest new joins over known foreign key relationships, modeled as pre-existing hidden fields, and to display exact join conditions.

For the remainder of this paper, we will use the following terminology when referring to concepts in the nested relational data model: A *value* is either a *primitive* or a *relation*, where a relation is defined as a set of *tuples*, each containing a set of *fields* identified by *labels*, each containing a value, recursively. The *schema* of a value either defines the value to be a primitive, or defines the value to be a relation, with schemas further specified for each of the latter's fields, recursively. See Figure 2.

3.2 Query Model

We now discuss the specific structure of queries in our system. A visual query is modeled as a nested relational schema that has been annotated with query- and presentation-related properties on each field. We refer to the annotated schema as the SIEUFERD *query model*. When SQL queries are generated from a visual query and flat result sets have been assembled into a nested relational result, the schema of the nested result is identical to the schema in the query model. This correspondence makes it straightforward to translate high-level user interactions on the visualized query result to concrete modifications on the underlying query model, and conversely, to indicate the state of the query model in the table header of the visualized result.

Table instantiation. As a basic rule, each relation in the query model gets to retrieve data from one concrete table in the underlying database; that relation is said to *instantiate* the database table. The following is a simple query that instantiates the table called COURSES and displays a selection of its fields:

courses					
id	area_id	title	may_pdf	may_audit	exam_type
56	2	Roman Art	N	Y	Other
177	2	Comedy	Y	Y	Final
845	2	Russian Drama	N	N	Other
1795	4	American Politics	Y	Y	Final
2566		Junior Seminars	N	N	Other
3921	4	Judicial Politics	Y	Y	Final

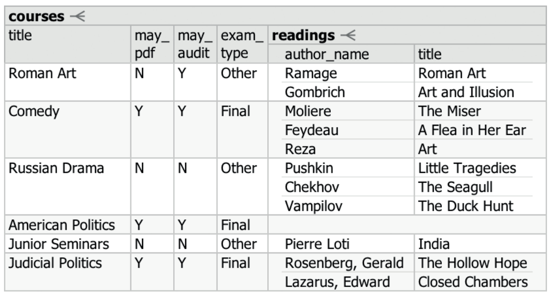
Nesting and joins. Queries need to be able to incorporate data from multiple tables. Commonly, tables need to be equijoined together, for example when the user wishes to examine data spread across foreign key relationships in a normalized database schema. In the SIEUFERD query model, the introduction of a new table instance can be done by defining a *nested relation*, optionally constrained by an equijoin condition against its parent relation:

courses						readings			
id	area_id	title	may_pdf	may_audit	exam_type	id	⋈ course_id	author_name	title
56	2	Roman Art	N	Y	Other	44	56	Ramage	Roman Art
						8,838	56	Gombrich	Art and Illusion
177	2	Comedy	Y	Y	Final	4,998	177	Moliere	The Miser
						12,138	177	Feydeau	A Flea in Her Ear
						16,878	177	Reza	Art
845	2	Russian Drama	N	N	Other	603	845	Pushkin	Little Tragedies
						9,207	845	Chekhov	The Seagull
						12,366	845	Vampilov	The Duck Hunt
1795	4	American Politics	Y	Y	Final				
2566		Junior Seminars	N	N	Other	9,935	2566	Pierre Loti	India
3921	4	Judicial Politics	Y	Y	Final	2,570	3921	Rosenberg, Gerald	The Hollow Hope
						17,629	3921	Lazarus, Edward	Closed Chambers

In the query above, the nested relation READINGS instantiates the database table with the same name, and equijoins itself against its parent relation COURSES on the COURSE_ID field, as indicated by the join icon (⋈) on the latter. The other side of the equijoin condition is the ID field in the COURSES relation. The latter information is omitted from the result layout to save space, but is displayed in the field selector (Figure 1). The one-to-many icon (≺) on the READINGS relation indicates that our system decided the latter may contain more than one tuple for each corresponding tuple in COURSES, the parent relation.

The joins described here have different semantics than the traditional flat joins encountered in SQL and most other visual query tools. Rather than duplicating tuples on one side of the operator for each occurrence of a matching tuple on the other, each tuple from the parent side of the join has a nested relation added to it holding zero or more matching tuples from the child side. This operator is known formally as a *nest equijoin* [45], though we will simply use the term *join* when unambiguous. One convenient property of nest equijoins is that tuples on the left-hand side of the operator do not disappear when the join fails to find matching tuples on the right; this can be seen in the query above for the course AMERICAN POLITICS, which has no books in its reading list.

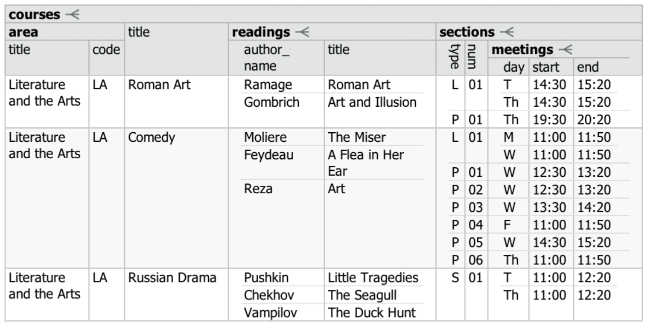
It is often desirable to hide technical primary key fields, fields made redundant by equijoin conditions (e.g. COURSE_ID), or otherwise uninteresting fields, for presentation purposes. Continuing the example above, our query model allows us to hide several fields without altering the query semantics:



courses ≺					
title	may_ pdf	may_ audit	exam_ type	readings ≺	
				author_name	title
Roman Art	N	Y	Other	Ramage	Roman Art
				Gombrich	Art and Illusion
Comedy	Y	Y	Final	Moliere	The Miser
				Feydeau	A Flea in Her Ear
				Reza	Art
Russian Drama	N	N	Other	Pushkin	Little Tragedies
				Chekhov	The Seagull
				Vampilov	The Duck Hunt
American Politics	Y	Y	Final		
Junior Seminars	N	N	Other	Pierre Loti	India
Judicial Politics	Y	Y	Final	Rosenberg, Gerald	The Hollow Hope
				Lazarus, Edward	Closed Chambers

The hidden fields could be recalled at any time using the field selector. As before, the field selector can also be used to see the exact join conditions between READINGS and COURSES.

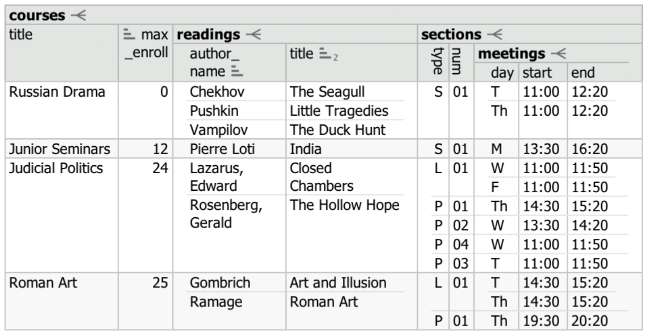
Nested relations can be used very effectively to display data spread over many tables in a database schema. In the following example, we pull data from five database tables to see more information about each university course:



courses ≺									
area		title	readings ≺		sections ≺				
title	code		author_ name	title	type	num	meetings ≺		
							day	start	end
Literature and the Arts	LA	Roman Art	Ramage	Roman Art	L	01	T	14:30	15:20
			Gombrich	Art and Illusion			Th	14:30	15:20
					P	01	Th	19:30	20:20
Literature and the Arts	LA	Comedy	Moliere	The Miser	L	01	M	11:00	11:50
			Feydeau	A Flea in Her Ear			W	11:00	11:50
					P	01	W	12:30	13:20
			Reza	Art	P	02	W	12:30	13:20
					P	03	W	13:30	14:20
					P	04	F	11:00	11:50
					P	05	W	14:30	15:20
					P	06	Th	11:00	11:50
Literature and the Arts	LA	Russian Drama	Pushkin	Little Tragedies	S	01	T	11:00	12:20
			Chekhov	The Seagull			Th	11:00	12:20
			Vampilov	The Duck Hunt					

Notice that tuples in the READINGS relation occur independently of tuples in the SECTIONS relation; this kind of visualization can not be constructed in tools based on flat tabular results (see Related Work). Also notice the absence of the one-to-many icon (≺) on the AREA relation: because the latter relation was joined on its instantiated table's primary key, our system deduced that at most one tuple can exist in AREA for each parent tuple in COURSES.

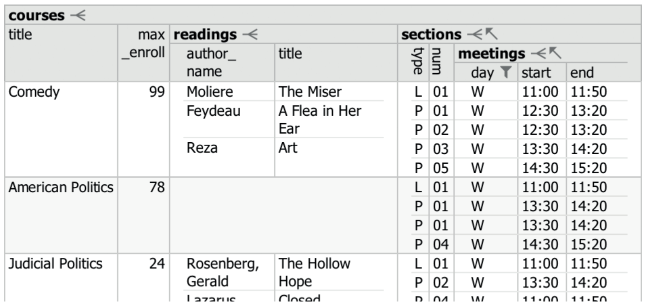
Sorting. Each nested relation can be sorted on a sequence of its direct child fields, indicated by subscripted sort icons ($\equiv_{123}$) on the latter. In the following example, the root-level COURSES relation is sorted ascending on the MAX_ENROLL field, while individual sets of READINGS are sorted by AUTHOR_NAME, then by TITLE:



courses ≺								
title	≡ max _enroll	readings ≺		sections ≺				
		author_ name ≡	title $\equiv_2$	type	num	meetings ≺		
						day	start	end
Russian Drama	0	Chekhov	The Seagull	S	01	T	11:00	12:20
		Pushkin	Little Tragedies			Th	11:00	12:20
		Vampilov	The Duck Hunt					
Junior Seminars	12	Pierre Loti	India	S	01	M	13:30	16:20
Judicial Politics	24	Lazarus, Edward	Closed Chambers	L	01	W	11:00	11:50
						F	11:00	11:50
		Rosenberg, Gerald	The Hollow Hope	P	01	Th	14:30	15:20
				P	02	W	13:30	14:20
				P	04	W	11:00	11:50
				P	03	T	11:00	11:50
Roman Art	25	Gombrich	Art and Illusion	L	01	T	14:30	15:20
		Ramage	Roman Art			Th	14:30	15:20
				P	01	Th	19:30	20:20

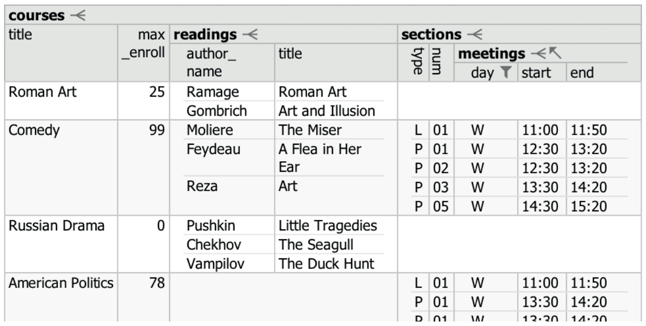
It is possible to sort on both primitive and relation fields, though we omit the exact semantics of the latter case here. Following any explicit sort terms, our system automatically sorts every relation on a tuple-identifying subset of its retrieved fields. This ensures that all query results are retrieved in a deterministic order. The automatic sort is usually on an indexed primary key; see *set projection* below.

Filter. Using the filter popup (Figure 1), a filter can be defined on any field, indicated by the filter icon (▼). Filters on relation fields restrict the set of tuples retrieved in that relation, while filters on primitive fields restrict the tuples of the parent relation. In the following example, the MEETINGS relation is filtered to show only tuples for which the DAY is W:



courses ≺								
title	max _enroll	readings ≺		sections ≺↖				
		author_ name	title	type	num	meetings ≺↖		
						day ▼	start	end
Comedy	99	Moliere	The Miser	L	01	W	11:00	11:50
		Feydeau	A Flea in Her Ear	P	01	W	12:30	13:20
				P	02	W	12:30	13:20
		Reza	Art	P	03	W	13:30	14:20
				P	05	W	14:30	15:20
American Politics	78			L	01	W	11:00	11:50
				P	01	W	13:30	14:20
				P	01	W	13:30	14:20
				P	04	W	14:30	15:20
Judicial Politics	24	Rosenberg, Gerald	The Hollow Hope	L	01	W	11:00	11:50
				P	02	W	13:30	14:20

By default, the effect of a filter in a nested relation is propagated all the way to the root of the query by means of a HIDE PARENT IF EMPTY setting on each intermediate relation, indicated by the arrow-towards-root icon (↖) on the SECTIONS and MEETINGS relations above. In the example, the courses ROMAN ART and RUSSIAN DRAMA have disappeared because they do not have any Wednesday sections. If, rather than retrieving “a list of courses with at least one Wednesday section”, we wanted to retrieve “a list of all courses, showing sections on Wednesday only”, we could deactivate HIDE PARENT IF EMPTY on the SECTIONS relation:



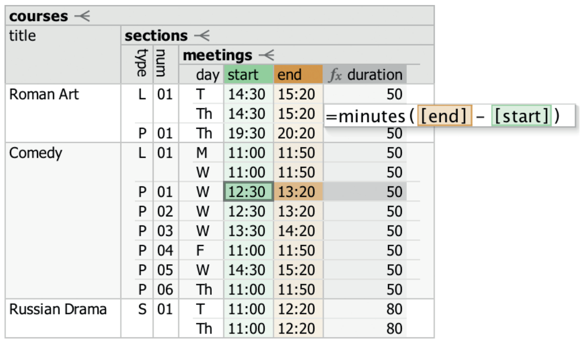
courses ≺								
title	max _enroll	readings ≺		sections ≺				
		author_ name	title	type	num	meetings ≺↖		
						day ▼	start	end
Roman Art	25	Ramage	Roman Art					
		Gombrich	Art and Illusion					
Comedy	99	Moliere	The Miser	L	01	W	11:00	11:50
		Feydeau	A Flea in Her Ear	P	01	W	12:30	13:20
				P	02	W	12:30	13:20
		Reza	Art	P	03	W	13:30	14:20
				P	05	W	14:30	15:20
Russian Drama	0	Pushkin	Little Tragedies					
		Chekhov	The Seagull					
		Vampilov	The Duck Hunt					
American Politics	78			L	01	W	11:00	11:50
				P	01	W	13:30	14:20

Formulas. An important part of the expressiveness offered by SQL is the ability to include scalar and aggregate computations over primitive values in any part of the query. In the SIEUFERD query model, both kinds of calculations are supported by means of *calculated fields*. A calculated field is a primitive field, added to any relation by the user, that takes its value from a *formula* rather than from a particular column in an instantiated database table. Like other fields, calculated fields can be sorted or filtered on.

SIEUFERD formulas are syntactically similar to spreadsheet formulas, but belong to and reference entire columns of field values rather than hard-coded ranges of cells. This allows SIEUFERD queries, like SQL queries, to be defined independently of the exact data that might reside in a database at any given time. Without this design, the user might have to rewrite formulas if the data in the underlying data source changes, or if other parts of the query are changed in such a way as to add or remove tuples in the result. Forgetting to update formulas when input data is changed is a common kind of error in spreadsheets [26, 12], which we avoid.

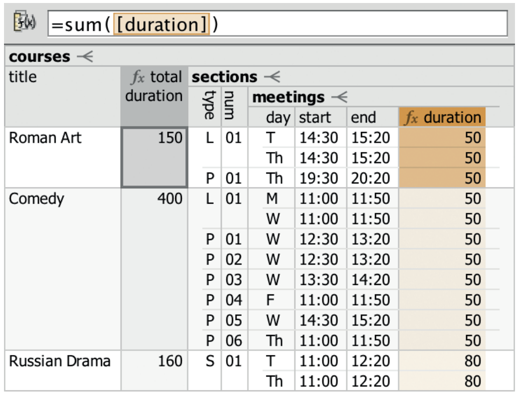
The restriction that calculated fields always be primitive fields is an important one; we do not wish formulas to take the role of a textual query language embedded within the visual one. Formulas do not provide a relational algebra, but rather allow simple computations over primitive values.

Continuing the course catalog example, we can calculate the duration of each meeting of a course section:



courses	sections		meetings			
title	type	num	day	start	end	duration
Roman Art	L	01	T	14:30	15:20	50
			Th	14:30	15:20	=minutes([end] – [start])
	P	01	Th	19:30	20:20	50
Comedy	L	01	M	11:00	11:50	50
			W	11:00	11:50	50
	P	01	W	12:30	13:20	50
	P	02	W	12:30	13:20	50
	P	03	W	13:30	14:20	50
	P	04	F	11:00	11:50	50
	P	05	W	14:30	15:20	50
	P	06	Th	11:00	11:50	50
Russian Drama	S	01	T	11:00	12:20	80
			Th	11:00	12:20	80

The calculated field DURATION, marked with the formula icon (*fx*), is evaluated once for each tuple in MEETINGS, its containing relation. Using another calculated field, we can add up the durations as well, at the level of each course:

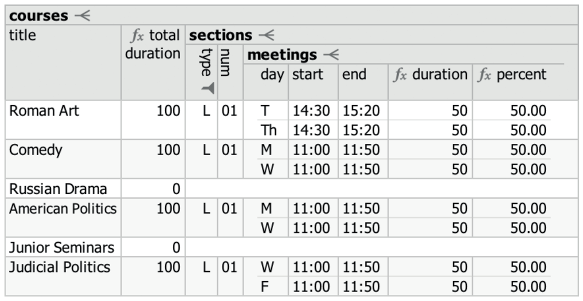


=sum([duration])

courses		sections		meetings			
title	total duration	type	num	day	start	end	duration
Roman Art	150	L	01	T	14:30	15:20	50
				Th	14:30	15:20	50
		P	01	Th	19:30	20:20	50
Comedy	400	L	01	M	11:00	11:50	50
				W	11:00	11:50	50
		P	01	W	12:30	13:20	50
		P	02	W	12:30	13:20	50
		P	03	W	13:30	14:20	50
		P	04	F	11:00	11:50	50
		P	05	W	14:30	15:20	50
		P	06	Th	11:00	11:50	50
Russian Drama	160	S	01	T	11:00	12:20	80
				Th	11:00	12:20	80

When using aggregate functions such as SUM or COUNT, the relation in which the calculated field is defined determines the level at which aggregate values are grouped. In the example above, because the TOTAL DURATION field is a child of the COURSES relation, a total is calculated for each course rather than, say, for each section. Each course includes in its total only tuples from the MEETINGS relation that are descendants of that course's tuple in the COURSES relation.

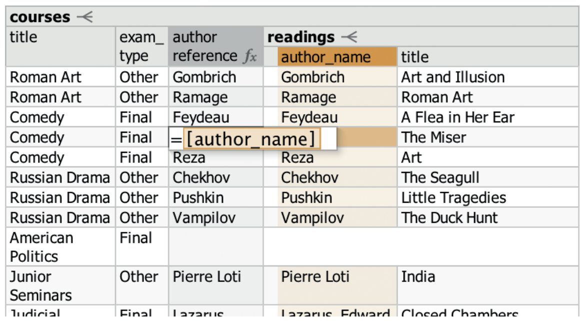
Filters and aggregate functions. When an aggregate function references a relation with a filter applied to it, the filter is evaluated before the aggregate. In the following example, the SECTIONS relation is filtered to only include lecture-type sections. The TOTAL DURATION for each course changes accordingly:



courses		sections		meetings				
title	total duration	type	num	day	start	end	duration	percent
Roman Art	100	L	01	T	14:30	15:20	50	50.00
				Th	14:30	15:20	50	50.00
Comedy	100	L	01	M	11:00	11:50	50	50.00
				W	11:00	11:50	50	50.00
Russian Drama	0							
American Politics	100	L	01	M	11:00	11:50	50	50.00
				W	11:00	11:50	50	50.00
Junior Seminars	0							
Judicial Politics	100	L	01	W	11:00	11:50	50	50.00
				F	11:00	11:50	50	50.00

It is equally valid to define a filter on the output side of an aggregate, e.g. on TITLE or TOTAL DURATION in the example above.

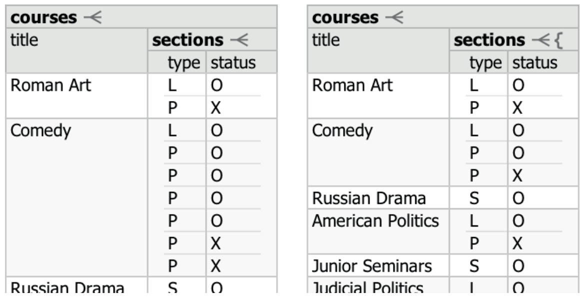
Flat joins. Traditional flat joins can be expressed by referencing a descendant relation from a formula without enclosing the reference in an aggregate function. In the following example, each course title is repeated once for each distinct author name in the reading list, because the AUTHOR REFERENCE field in the COURSES relation references the READINGS relation without the use of an aggregate function:



courses			readings	
title	exam_type	author reference	author_name	title
Roman Art	Other	Gombrich	Gombrich	Art and Illusion
Roman Art	Other	Ramage	Ramage	Roman Art
Comedy	Final	Feydeau	Feydeau	A Flea in Her Ear
Comedy	Final	=[author_name]		The Miser
Comedy	Final	Reza	Reza	Art
Russian Drama	Other	Chekhov	Chekhov	The Seagull
Russian Drama	Other	Pushkin	Pushkin	Little Tragedies
Russian Drama	Other	Vampilov	Vampilov	The Duck Hunt
American Politics	Final			
Junior Seminars	Other	Pierre Loti	Pierre Loti	India
Judicial	Final	Lazarus	Lazarus, Edward	Closed Chambers

The actual behavior is that of a left join, with a null value being returned for the course AMERICAN POLITICS, which has no readings in its reading list. To express an inner join instead, the HIDE PARENT IF EMPTY setting could be enabled on the READINGS relation. The left join semantics of these *inward* formula references help our visual query language maintain some desirable properties. In particular, the mere introduction of a new calculated field (e.g. AUTHOR REFERENCE) will never cause tuples to disappear from said field's containing relation (COURSES).

Set projection. By default, tuples retrieved for a relation always include the primary key fields of the relation's instantiated table, even if the user has hidden those fields from view. This allows our system to keep result tuples in a stable order as the user hides or shows fields, and to keep a one-to-one relationship between tuples on the screen and tuples in instantiated database tables. It also allows us to generate more efficient SQL queries, for example by avoiding expensive `SELECT DISTINCT` statements. The automatic inclusion of primary key fields in the projection of a particular relation can be avoided by means of the HIDE DUPLICATE ROWS option, indicated by the bracket icon ({):



courses	sections	
title	type	status
Roman Art	L	O
	P	X
Comedy	L	O
	P	O
	P	O
	P	O
	P	O
	P	X
	P	X
Russian Drama	S	O

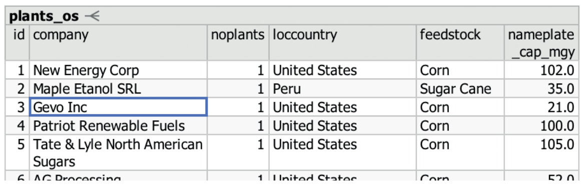
courses	sections	
title	type	status
Roman Art	L	O
	P	X
Comedy	L	O
	P	O
	P	X
Russian Drama	S	O
American Politics	L	O
	P	X
Junior Seminars	S	O
Judicial Politics	L	O

3.3 Query Building

Having explained the query model, we now show how the user would actually build queries using our direct manipulation interface. We do this by means of an example query building session. The user is an investigative journalist who is writing a story about ethanol biofuel lobbying[2]. She has compiled, in the table PLANTS_OS, a list of major ethanol producers[3], and would like to find the total lobbying expenditures of each. Another table, LOBBYING, contains quarterly lobbying reports from US corporations in the years 1998 through 2012 (727,927 tuples)[4].

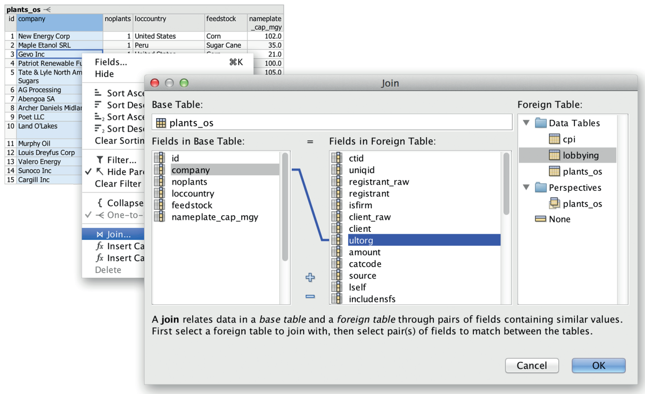
[2]E. Díaz-Struck (2013). Ethanol Industry Battles to Keep Incentives. http://eye.necir.org/2013/05/26/ethanol-industry-battles-to-keep-incentives
[3]Renewable Fuels Association/Maple Etanol SRL (2012)
[4]The Center for Responsive Politics (2012) https://www.opensecrets.org

Base table. The user starts by opening the table of ethanol producers as a template for the new query:



plants_os					
id	company	noplants	loccountry	feedstock	nameplate_cap_mgy
1	New Energy Corp	1	United States	Corn	102.0
2	Maple Etanol SRL	1	Peru	Sugar Cane	35.0
3	Gevo Inc	1	United States	Corn	21.0
4	Patriot Renewable Fuels	1	United States	Corn	100.0
5	Tate & Lyle North American Sugars	1	United States	Corn	105.0

Join. To add another table to the query, the user selects the column or columns to join on and invokes the JOIN action from the context menu. This opens a dialog box for selecting the table to join with, in this case LOBBYING, and for selecting the corresponding columns from the latter to be matched in an equijoin constraint. The user joins the PLANTS_OS and LOBBYING tables on the COMPANY and ULTORG fields, respectively:



plants_os					
id	company	noplants	loccountry	feedstock	nameplate_cap_mgy
1	New Energy Corp	1	United States	Corn	102.0
2	Maple Etanol SRL	1	Peru	Sugar Cane	35.0
3	Gevo Inc				21.0
4	Patriot Renewable Fu				100.0
5	Tate & Lyle North Am Sugars				105.0
6	AG Processing				
7	Abengoa SA				
8	Archer Daniels Midlar				
9	Poet LLC				
10	Land O'Lakes				
11	Murphy Oil				
12	Louis Dreyfus Corp				
13	Valero Energy				
14	Sunoco Inc				
15	Cargill Inc				

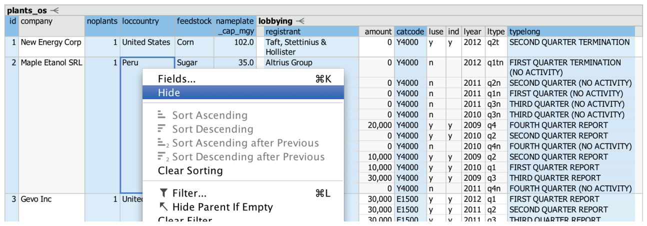
Fields in Base Table: id, company, noplants, loccountry, feedstock, nameplate_cap_mgy

Fields in Foreign Table: ctid, uniqid, registrant_raw, registrant, isfirm, client_raw, client, ultorg, amount, catcode, source, lself, includensfs

A **join** relates data in a *base table* and a *foreign table* through pairs of fields containing similar values. First select a foreign table to join with, then select pair(s) of fields to match between the tables.

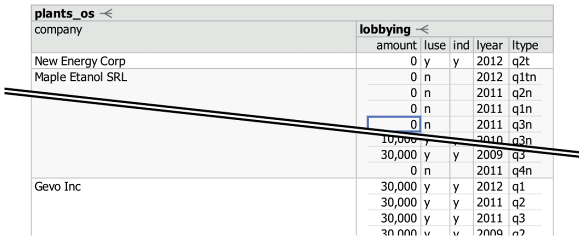
In cases where the database defines explicit foreign key relationships between tables, use of the above JOIN dialog is unnecessary; instead, all available joins will be available as hidden relations in the field selector. The effect is a schema navigation capability analogous to that of QBB [40], AppForge [50], and App2You [31].

Hide fields. After the join, a lot of columns are shown, so the user selects a few of them and invokes the HIDE action:



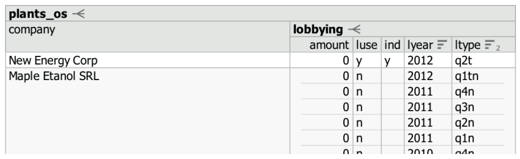
plants_os						lobbying							
id	company	noplants	loccountry	feedstock	nameplate_cap_mgy	registrant	amount	catcode	luse	ind	lyear	ltype	typelong
1	New Energy Corp	1	United States	Corn	102.0	Taft, Stettinius & Hollister	0	Y4000	y	y	2012	q2t	SECOND QUARTER TERMINATION
2	Maple Etanol SRL	1	Peru	Sugar	35.0	Altrius Group	0	Y4000	n		2012	q1tn	FIRST QUARTER TERMINATION (NO ACTIVITY)
							0	Y4000	n		2011	q2n	SECOND QUARTER (NO ACTIVITY)
							0	Y4000	n		2011	q1n	FIRST QUARTER (NO ACTIVITY)
							0	Y4000	n		2011	q3n	THIRD QUARTER (NO ACTIVITY)
							0	Y4000	n		2010	q3n	THIRD QUARTER (NO ACTIVITY)
							20,000	Y4000	y	y	2009	q4	FOURTH QUARTER REPORT
							0	Y4000	y	y	2010	q2	SECOND QUARTER REPORT
							0	Y4000	n		2010	q4n	FOURTH QUARTER (NO ACTIVITY)
							10,000	Y4000	y	y	2009	q2	SECOND QUARTER REPORT
							10,000	Y4000	y	y	2010	q1	FIRST QUARTER REPORT
							30,000	Y4000	y	y	2009	q3	THIRD QUARTER REPORT
							0	Y4000	n		2011	q4n	FOURTH QUARTER (NO ACTIVITY)
3	Gevo Inc	1	Unite				30,000	E1500	y	y	2012	q1	FIRST QUARTER REPORT
							30,000	E1500	y	y	2011	q2	SECOND QUARTER REPORT

It is now easier to get a sense of the data. We have a new child relation field, called LOBBYING, containing the lobbying reports for each company:



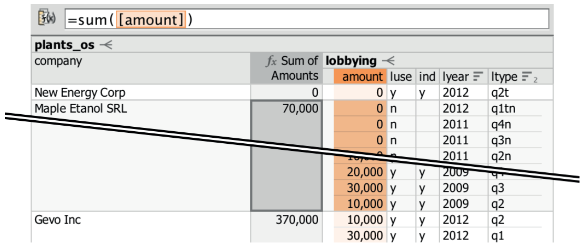
plants_os	lobbying				
company	amount	luse	ind	lyear	ltype
New Energy Corp	0	y	y	2012	q2t
Maple Etanol SRL	0	n		2012	q1tn
	0	n		2011	q2n
	0	n		2011	q1n
	0	n		2011	q3n
	10,000			2010	q3n
	30,000	y	y	2009	q3
	0	n		2011	q4n
Gevo Inc	30,000	y	y	2012	q1
	30,000	y	y	2011	q2
	30,000	y	y	2011	q3

Sort. The user decides to sort the lobbying reports for each company most-recent-first, invoking the SORT DESCENDING action on the LYEAR field and then invoking the SORT DESCENDING AFTER PREVIOUS action on the LTYPE field. This sorts individual LOBBYING relations by year (⩵) and then by quarter (⩵$_2$):



plants_os	lobbying				
company	amount	luse	ind	lyear	ltype
New Energy Corp	0	y	y	2012	q2t
Maple Etanol SRL	0	n		2012	q1tn
	0	n		2011	q4n
	0	n		2011	q3n
	0	n		2011	q2n
	0	n		2011	q1n

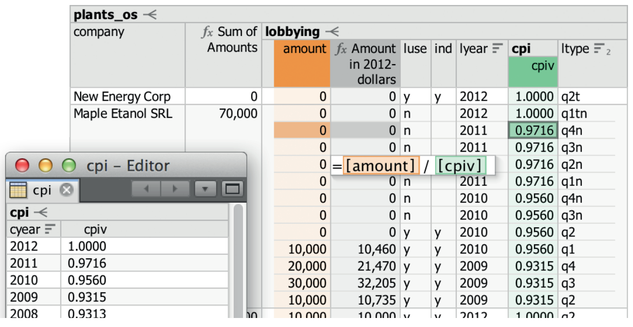
Aggregate formula. The user would now like to calculate a total lobbying amount for each company. She invokes the INSERT CALCULATED FIELD AFTER action to insert a calculated field (*fx*) next to the COMPANY field, and enters the name SUM OF AMOUNTS in the new column's label cell. She then moves the cursor to one of the column's value cells, and enters a sum formula, clicking the AMOUNT column to insert the column reference:



=sum([amount])

plants_os		lobbying				
company	*fx* Sum of Amounts	amount	luse	ind	lyear	ltype
New Energy Corp	0	0	y	y	2012	q2t
Maple Etanol SRL	70,000	0	n		2012	q1tn
		0	n		2011	q4n
		0	n		2011	q3n
			n		2011	q2n
		20,000	y	y	2009	
		30,000	y	y	2009	q3
		10,000	y	y	2009	q2
Gevo Inc	370,000	10,000	y	y	2012	q2
		30,000	y	y	2012	q1

Scalar formula. Reported lobbying amounts come from different years, some going back to 1998. The user would like to calculate inflation-corrected totals. A separate table CPI contains yearly Consumer Price Index values normalized for 2012. The user performs another JOIN, this time between LOBBYING and CPI, on the LYEAR and CYEAR fields, respectively. This brings the CPIV value for each lobbying report's year into the nested result. The user then adds another calculated field, this time under the same relation as the existing AMOUNT field, and enters a formula that calculates the inflation-adjusted amount for each report. We here have a useful example of an inward formula reference (to CPIV) that is not enclosed in an aggregate function:

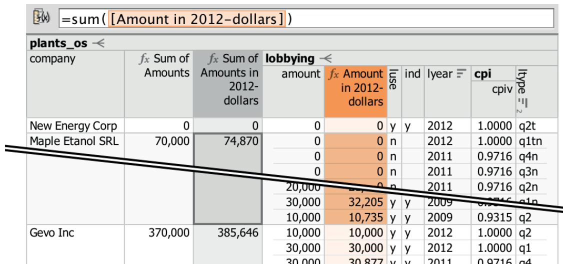


plants_os		lobbying					cpi	
company	*fx* Sum of Amounts	amount	*fx* Amount in 2012-dollars	luse	ind	lyear	cpiv	ltype
New Energy Corp	0	0	0	y	y	2012	1.0000	q2t
Maple Etanol SRL	70,000	0	0	n		2012	1.0000	q1tn
		0	0	n		2011	0.9716	q4n
		0	0	n		2011	0.9716	q3n
		0	=[amount] / [cpiv]				0.9716	q2n
		0	0	n		2011	0.9716	q1n
		0	0	n		2010	0.9560	q4n
		0	0	n		2010	0.9560	q3n
		0	0	y	y	2010	0.9560	q2
		10,000	10,460	y	y	2010	0.9560	q1
		20,000	21,470	y	y	2009	0.9315	q4
		30,000	32,205	y	y	2009	0.9315	q3
		10,000	10,735	y	y	2009	0.9315	q2

cpi – Editor

cpi	
cyear	cpiv
2012	1.0000
2011	0.9716
2010	0.9560
2009	0.9315
2008	0.9313

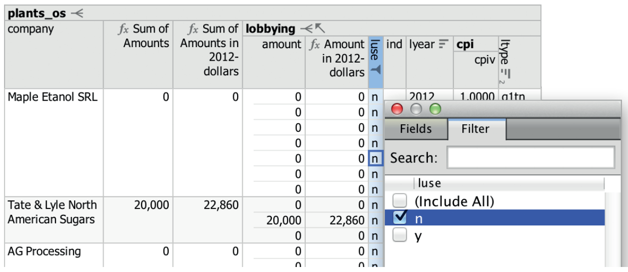
A new inflation-adjusted total can now be added as a calculated field at the PLANTS_OS level, shown adjacent to the existing non-adjusted sum:



=sum([Amount in 2012-dollars])

plants_os			lobbying					cpi	
company	*fx* Sum of Amounts	*fx* Sum of Amounts in 2012-dollars	amount	*fx* Amount in 2012-dollars	luse	ind	lyear	cpiv	ltype
New Energy Corp	0	0	0	0	y	y	2012	1.0000	q2t
Maple Etanol SRL	70,000	74,870	0	0	n		2012	1.0000	q1tn
			0	0	n		2011	0.9716	q4n
			0	0	n		2011	0.9716	q3n
			20,000		n		2011	0.9716	q2n
			30,000	32,205	y	y	2009		
			10,000	10,735	y	y	2009	0.9315	q2
Gevo Inc	370,000	385,646	10,000	10,000	y	y	2012	1.0000	q2
			30,000	30,000	y	y	2012	1.0000	q1

Filter. Lobbying reports may sometimes be amended, in which case the superseded reports should be excluded from totals to avoid double counting. The user can look for superseded reports by invoking the FILTER action on the LUSE field and selecting the value N:



plants_os

company	ƒx Sum of Amounts	ƒx Sum of Amounts in 2012-dollars	lobbying amount	ƒx Amount in 2012-dollars	luse	ind	lyear	cpi cpiv	ltype
Maple Etanol SRL	0	0	0	0	n		2012	1.0000	g1tn
			0	0	n				
			0	0	n				
			0	0	n				
			0	0	n				
			0	0	n				
			0	0	n				
Tate & Lyle North American Sugars	20,000	22,860	0	0	n				
			20,000	22,860	n				
			0	0	n				
AG Processing	0	0	0	0	n				

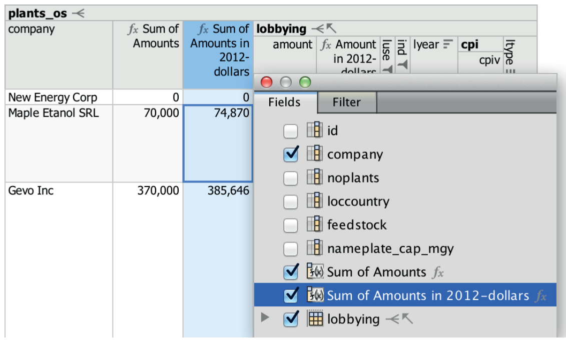
Fields | Filter

Search:

luse
- [ ] (Include All)
- [x] n
- [ ] y

The user sees that there are superseded reports in the database with non-zero dollar amounts, and inverts the filter to exclude them.

Select fields. The user now decides to hide the individual reports altogether and instead reintroduce some of the fields that were hidden from the PLANTS_OS relation before, using the field selector:



plants_os

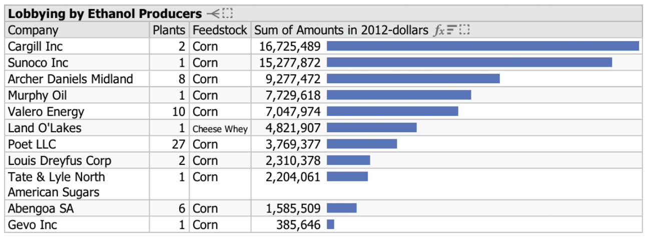
company	ƒx Sum of Amounts	ƒx Sum of Amounts in 2012-dollars
New Energy Corp	0	0
Maple Etanol SRL	70,000	74,870
Gevo Inc	370,000	385,646

lobbying: amount, ƒx Amount in 2012-dollars, luse, ind, lyear, cpi cpiv, ltype

Fields | Filter

- [ ] id
- [x] company
- [ ] noplants
- [ ] loccountry
- [ ] feedstock
- [ ] nameplate_cap_mgy
- [x] Sum of Amounts ƒx
- [x] Sum of Amounts in 2012-dollars ƒx
- [x] lobbying

Final touches. The user edits the field labels to make them a bit more readable, and sorts the companies by their lobbying totals. The underlying SQL column names can still be seen in the field selector. The user also enables a formatting option on the last column to produce a bar chart visualization. The result now looks presentable:



Lobbying by Ethanol Producers

Company	Plants	Feedstock	Sum of Amounts in 2012-dollars ƒx
Cargill Inc	2	Corn	16,725,489
Sunoco Inc	1	Corn	15,277,872
Archer Daniels Midland	8	Corn	9,277,472
Murphy Oil	1	Corn	7,729,618
Valero Energy	10	Corn	7,047,974
Land O'Lakes	1	Cheese Whey	4,821,907
Poet LLC	27	Corn	3,769,377
Louis Dreyfus Corp	2	Corn	2,310,378
Tate & Lyle North American Sugars	1	Corn	2,204,061
Abengoa SA	6	Corn	1,585,509
Gevo Inc	1	Corn	385,646

While the LOBBYING relation that feeds into the aggregate formula is now hidden, the user could easily make it visible again from the field selector, like she did for the previously hidden PLANTS and FEEDSTOCK fields. There are also shortcuts for unhiding hidden fields referenced from the formula, or the hidden filter, indicated by the dashed cell icons (⬚).

4. FORMATIVE USER STUDY

We conducted a formative user study with 14 participants (denoted A through N, 5 male, median age 42). 7 had experience with SQL, 11 used Excel daily. In the first part of the study, done by users A-I, users were given standardized tasks aimed at assessing the initial learnability of our tool. No prior training was given; instead, initial tasks were designed to act as training tasks for subsequent ones. In the second part of the study, and as time permitted during earlier sessions, users were given a chance to do more open-ended tasks on datasets we provided. Here, we gave participants demos and instructions for operating our tool, in order to gather higher-level observations than would be possible during pure learning tasks. In this section we discuss a selection of observations from our study; see our full paper [5] for more details.

Manual joins. Performing the lobbying query from Section 3.3, most users moved through the manual join dialog quickly and correctly on their first attempt. Still, users preferred automatic joins once introduced to them, see below. Users attempting the inflation correction portion of the query had no problems with the join against the CPI table; only users DG required a hint that they would need to use the JOIN feature again.

Formulas. When first attempting to perform a sum aggregation, users BCDE started by looking for an explicit sum action, as would be found in Excel's toolbar. Users CGK looked for an Excel-style formula builder. Having eventually realized that they needed to insert a calculated field and enter a formula themselves, users DEFK had initial trouble learning how to physically enter the formula, trying for example to enter the formula in an already-existing column, or in the column header.

In Excel, sums can be produced either using formulas or pivot tables. The two interfaces are largely separate, with users often preferring one or the other. Our system follows the formula approach. Users CH commented that they thought of pivot tables when first trying to compute a sum, while users BEI thought of pivot tables during other tasks.

A significant difference between spreadsheet formulas and SIEUFERD formulas is that the latter, like SQL queries, reference entire columns of values rather than an explicit range of cells. Users ABCFH expected this on their first attempts to insert a reference in a sum formula. Users DEGN expected the spreadsheet model, initially attempting to select a range of cells. A related challenge was to understand the level at which a calculated field should be inserted in order for sums to be grouped in the right way. The fact that the position of a formula in the relation hierarchy determines the grouping of aggregate functions is a further deviation from the spreadsheet model, while the lack of an explicit `GROUP BY` clause may be confusing to SQL users. User H tried to specify the set of columns to group by in the aggregate function itself, as in the formula =SUM([NAME],[AMOUNT]), while user F tried to hide every field other than the one to be summed. User G attempted to invoke the HIDE DUPLICATE ROWS action. Users CFGH also tried placing the calculated field next to the value to be summed rather than at the parent level. User G thought aloud:

"Wouldn't it be fantastic if there was a way simply to operate at that group level rather than these individual entries? [After creating a new formula at the correct level:] Is it doing it that way? Oh, that's perfect. ... That is meeting my heart's desire. But I wouldn't have the cue for that."

Despite initial difficulty with formulas in a training task, users applied them quickly and accurately in a follow-up task, despite the follow-up task requiring more steps. This suggests users are able to apply formulas effectively after first learning them, but that there is significant potential for improved learnability. We agree with users AM, who suggested adding an explicit sum action like that of Excel. This feature would automatically generate a sum formula above the nearest one-to-many relationship, which would then serve as an example to the user to learn from.

After initial learning, users appreciated the behavior of formulas. Users CEGK noted explicitly that the behavior of aggregate functions, including grouping and subtotaling behavior, made sense. Users ILK also commented that the all-column nature of formula references made sense and was an advantage over Excel's range-style references. User K noted:

"I just feel like I have a truer sense of what I'm adding up, or

what's being considered in this format vs. the traditional Excel. Because [in Excel] you could be pulling from the wrong places, you can be getting weird numbers, you could accidentally hit a field that now ends up in your calculation."

Field selection; automatic joins. Working on the course catalog dataset that was seen in Section 3.2, users were generally able to use the automatic foreign key join feature without trouble. The exception was user N, who had a hard time because of the lack of visible indications in the result area that more fields could be shown. User G also noted this issue. Users IKN specifically looked for an action named "Unhide", like in Excel. This suggests that our user interface needs a more visible affordance for accessing hidden fields. We expect hidden fields to be far more common in SIEUFERD than in Excel, since a typical database query projects only a small subset of columns available from instantiated database tables. The design of an improved unhide affordance should take this into account.

Users EGHJKL reacted particularly enthusiastically to the automatic join feature, using words such as "fantastic", "wow", "damn", and "amazing". User E noted:

"Yes, the manual join made sense, but that was a very simple situation. I wouldn't want to have done the joins on this [more complicated database]. The fact that I was just able to double-click and expand it out, that meant, it dumbed the task down to the level that I was happy performing it."

5. CONTROLLED USER STUDY

In a second user study, we aimed to get a more precise idea of how users might rate our system compared to an existing industry tool. We chose the "Query Design" facility of Microsoft Access 2016 as a control. Being part of the Office Professional suite, it is one of the most common visual query tools available. It is also a good example of a query builder that uses a diagram-based approach rather than direct manipulation of results (see Related Work).

The controlled study was a within-subjects counterbalanced design, measuring usability using the System Usability Scale (SUS) [9]. Tullis and Stetson [47] recommend sample sizes of 12-14 users to get reasonably representative results from within-subjects studies based on the SUS survey; we collected data from 14 users (5 male, median age 36). 2 had prior experience with the Access query designer, 6 had significant exposure to SQL. 2 used Excel daily, the rest weekly or monthly. We met with each user for a single study session, structured as follows:

1. Complete demographic/background survey.
2. Briefly discuss the sample database that will be used for tasks, consulting a schema diagram on paper. The paper diagram remains available to the user during the tasks that follow.
3. Work through some standardized tasks to evaluate Tool 1. Stop after about 20 minutes. The first tool is SIEUFERD for half of the users and Microsoft Access for the other half, randomized.
4. Complete SUS survey for Tool 1.
5. Work through the same tasks in Tool 2, under otherwise identical conditions. Stop after about 20 minutes.
6. Complete SUS survey for Tool 2.
7. Discussion and feedback.

The standardized tasks [5], all done on the 7-table "Northwind" example database that shipped with older versions of Microsoft Access, are intended to be realistic examples of queries that a user might want to run on such a database. They incorporate joins, filters, sorting, scalar calculations and aggregates, but are limited to queries that can be expressed in Microsoft Access' visual query designer; this excludes queries requiring nested results as well as multi-block queries (e.g. aggregates used as inputs to other aggregates). In both tools, we configured foreign key relationships upfront so that the user would not have to manually specify exact join constraints between tables. The first five tasks are guided training tasks, intended to expose the user to all features, in both tools, that are needed to complete the subsequent unguided tasks. The guided tasks tended to take about half of the 20 minutes that users had available to try each tool. After the guided tasks, users were asked to try solving four unguided tasks without help. Since the main purpose of tasks was to give the user enough of an impression of each system to complete the subsequent SUS survey, we gave hints during unguided tasks whenever users reported being stuck.

Table 2: Mean SUS survey results for the controlled study, using various standard scales. Higher scores are better. Error bars show the standard error of the mean.

Scale	*Tool*	*Score (0-100)*
Raw SUS	Access	50
	Sieuferd	68
Learnability	Access	49
	Sieuferd	64
Usability	Access	50
	Sieuferd	69
Percentile	Access	6
	Sieuferd	52

The results of the study are shown in Table 2. The raw SUS score is reported along with separate Learnability and Usability scores as defined by Lewis and Sauro [34], as well as a percentile rating among 30 other studies in the B2B (Business Software) category as detailed by Sauro [42]. The difference in raw SUS scores between Access and SIEUFERD is statistically significant ($p = 0.0019$ with two-tailed paired t-test).

Interpreting the results, with the caveat that these observations are based on only 20-minute interactions with each tool, we see that SIEUFERD significantly outperformed Microsoft Access in terms of usability. Most of the difference can be attributed to the poor performance of Microsoft Access, considering its low ranking on the percentile scale; SIEUFERD simply achieved an average rating compared to other business software. This supports the original hypothesis of our paper: database querying is hard, but can be made significantly easier using a direct manipulation interface. SIEUFERD still has significant potential for improved usability. In conversations with users, the main requests for future design improvements were (1) the ability to get an overview of the complete database schema from within the query interface and (2) reduced dependency on formulas during query building. This is consistent with observations from the formative study.

6. CONCLUSION

SIEUFERD is a visual query system that achieves SQL-like expressiveness from a pure direct manipulation interface. Whereas previous direct manipulation systems either sacrifice expressiveness or hide the actual query from the user, SIEUFERD integrates the query and its result into a single interactive visualization, using spreadsheet concepts like filters and formulas to expose the complete state of the current query. Compared with the diagram-based query designer of Microsoft Access 2016, users greatly preferred our direct manipulation interface, with the latter scoring 46 percentiles higher on a SUS-based percentile scale. In future work, we hope to incorporate *editing* of data in our system; this will allow SIEUFERD to act as a complete schema-independent end user front-end for relational databases.

7. REFERENCES

[1] A. Abouzied, J. Hellerstein, and A. Silberschatz. DataPlay: Interactive tweaking and example-driven correction of graphical database queries. In *Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST '12)*, pages 207–218, New York, NY, USA, 2012. ACM.

[2] S. Achler. GBXT: A gesture-based data exploration tool for your favorite database system. In *Model and Data Engineering*, pages 224–237. Springer International Publishing, Cham, Switzerland, 2014.

[3] M. Angelaccio, T. Catarci, and G. Santucci. Query by Diagram: A fully visual query system. *Journal of Visual Languages & Computing*, 1(3):255–273, 1990.

[4] E. Bakke and E. Benson. The schema-independent database UI: A proposed holy grail and some suggestions. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, 2011.

[5] E. Bakke and D. R. Karger. Expressive query construction through direct manipulation of nested relational results. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*, pages 1377–1392, New York, NY, USA, 2016. ACM.

[6] E. Bakke, D. R. Karger, and R. C. Miller. Automatic layout of structured hierarchical reports. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2586–2595, December 2013.

[7] E. Baralis and J. Widom. An algebraic approach to static analysis of active database rules. *ACM Transactions on Database Systems (TODS)*, 25(3):269–332, September 2000.

[8] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop (SWUI '06)*, 2006.

[9] J. Brooke. SUS: A quick and dirty usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, editors, *Usability evaluation in industry*, pages 189–194. Tailor & Francis, London, UK, 1996.

[10] M. Burnett, J. Atwood, R. Walpole Djang, J. Reichwein, H. Gottfried, and S. Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of Functional Programming*, 11:155–206, March 2001.

[11] 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.

[12] J. P. Caulkins, E. L. Morrison, and T. Weidemann. Spreadsheet errors and decision making: Evidence from field interviews. *Journal of Organizational and End User Computing*, 19(3):1, 2007.

[13] K. S.-P. Chang and B. A. Myers. Using and exploring hierarchical data in spreadsheets. In *Proceedings of the 34th Annual ACM Conference on Human Factors in Computing Systems (CHI '16)*, New York, NY, USA, 2016. ACM.

[14] W.-K. Chen and P.-Y. Tu. VisualTPL: A visual dataflow language for report data transformation. *Journal of Visual Languages & Computing*, 25(3):210–226, 2014.

[15] C. Clack and L. Braine. Object-oriented functional spreadsheets. In *Proceedings of the 10th Glasgow Workshop on Functional Programming (GlaFP '97)*, 1997.

[16] E. F. Codd. Relational completeness of data base sublanguages. In *Database Systems*, pages 65–98. Prentice Hall, 1972.

[17] S. El-Mahgary and E. Soisalon-Soininen. A form-based query interface for complex queries. *Journal of Visual Languages & Computing*, 29:15–53, 2015.

[18] R. G. Epstein. The TableTalk query language. *Journal of Visual Languages & Computing*, 2(2):115–141, 1991.

[19] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems (TODS)*, 2(3):262–278, 1977.

[20] Y. Han, G. Wang, G. Ji, and P. Zhang. Situational data integration with data services and nested table. *Service Oriented Computing and Applications*, 7(2):129–150, 2013.

[21] L. Hella, L. Libkin, J. Nurmonen, and L. Wong. Logics with aggregate operators. *Journal of the ACM (JACM)*, 48(4):880–907, July 2001.

[22] G.-J. Houben and J. Paredaens. A graphical interface formalism: Specifying nested relational databases. In *Proceedings of the IFIP TC2 Working Conference on Visual Database Systems*, pages 257–276, 1989.

[23] Y. E. Ioannidis. Visual user interfaces for database systems. *ACM Computing Surveys (CSUR)*, 28(4es), 1996.

[24] G. Jaeschke and H. J. Schek. Remarks on the algebra of non first normal form relations. In *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS '82)*, pages 124–138, New York, NY, USA, 1982. ACM.

[25] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 13–24, New York, NY, USA, 2007. ACM.

[26] D. Janvrin and J. Morrison. Using a structured design approach to reduce risks in end user spreadsheet development. *Information & management*, 37(1):1–12, 2000.

[27] M. Jayapandian and H. V. Jagadish. Automated creation of a forms-based database query interface. *Proceedings of the VLDB Endowment*, 1:695–709, August 2008.

[28] M. Jayapandian and H. V. Jagadish. Expressive query specification through form customization. In *Proceedings of the 11th International Conference on Extending Database Technology (EDBT '08)*, pages 416–427, New York, NY, USA, 2008. ACM.

[29] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the 2011 annual conference on Human Factors in Computing Systems (CHI '11)*, pages 3363–3372, New York, NY, USA, 2011. ACM.

[30] E. Kandogan, E. Haber, R. Barrett, A. Cypher, P. Maglio, and H. Zhao. A1: End-user programming for web-based system administration. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST '05)*, pages 211–220, New York, NY, USA, 2005. ACM.

[31] K. Kowalzcykowski, A. Deutsch, K. W. Ong, Y. Papakonstantinou, K. K. Zhao, and M. Petropoulos. Do-It-Yourself database-driven web applications. In *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research (CIDR '09)*, 2009.

[32] D. Król, J. Oleksy, M. Podyma, and B. Trawiński. The analysis of reporting tools for a cadastre information system. In *Proceedings of the 9th International Conference on Business Information Systems (BIS '06)*, pages 150–163, 2006.

[33] M. Levene. *The Nested Universal Relation Database Model*, volume 595 of *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, 1992.

[34] J. R. Lewis and J. Sauro. The factor structure of the system usability scale. In *Proceedings of the 1st International Conference on Human Centered Design (HCD '09)/HCI International 2009*, pages 94–103, Berlin, Heidelberg, 2009. Springer-Verlag.

[35] B. Liu and H. V. Jagadish. A spreadsheet algebra for a direct data manipulation query interface. In *Proceedings of the IEEE 25th International Conference on Data Engineering (ICDE '09)*, pages 417–428, April 2009.

[36] N. Lorentzos and K. Dondis. Query by Example for Nested Tables. In *Database and Expert Systems Applications*, pages 716–725. Springer, 1998.

[37] R. M. McCutchen, S. Itzhaky, and D. Jackson. Initial report on Object Spreadsheets. Technical Report MIT-CSAIL-TR-2016-001, MIT Computer Science and Artificial Intelligence Laboratory, January 2016.

[38] A. Nandi, L. Jiang, and M. Mandel. Gestural query specification. *Proceedings of the VLDB Endowment*, 7(4):289–300, 2013.

[39] Y. Papakonstantinou, M. Petropoulos, and V. Vassalos. QURSED: Querying and reporting semistructured data. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 192–203, New York, NY, USA, 2002. ACM.

[40] S. Polyviou, G. Samaras, and P. Evripidou. A relationally complete visual query language for heterogeneous data sources and pervasive querying. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*, pages 471–482, Washington, DC, USA, 2005. IEEE Computer Society.

[41] L. Qian, K. LeFevre, and H. V. Jagadish. CRIUS: User-friendly database design. *Proceedings of the VLDB Endowment*, 4(2):81–92, 2010.

[42] J. Sauro. *A practical guide to the System Usability Scale: Background, benchmarks & best practices*. Measuring Usability LLC, 2011.

[43] B. Shneiderman. Direct Manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69, 1983.

[44] M. Spenke and C. Beilken. A spreadsheet interface for logic programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '89)*, pages 75–80, New York, NY, USA, 1989. ACM.

[45] H. J. Steenhagen, P. M. G. Apers, and H. M. Blanken. Optimization of nested queries in a complex object model. In *Proceedings of the 4th International Conference on Extending Database Technology (EDBT '94)*, pages 337–350, New York, NY, USA, 1994. Springer New York.

[46] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional databases. *Communications of the ACM*, 51(11):75–84, November 2008.

[47] T. S. Tullis and J. N. Stetson. A comparison of questionnaires for assessing website usability, 2004. Usability Professionals Association (UPA) 2004 Conference.

[48] J. Tyszkiewicz. Spreadsheet as a relational database engine. In *Proceedings of the 2010 International Conference on Management of Data (SIGMOD '10)*, pages 195–206, New York, NY, USA, 2010. ACM.

[49] L. Wegner, S. Thelemann, J. Thamm, D. Wilke, and S. Wilke. Navigational exploration and declarative queries in a prototype for visual information systems. In C. Leung, editor, *Visual Information Systems*, volume 1306 of *Lecture Notes in Computer Science*, pages 199–218. Springer Berlin/Heidelberg, 1997.

[50] F. Yang, N. Gupta, C. Botev, E. F. Churchill, G. Levchenko, and J. Shanmugasundaram. WYSIWYG development of data driven web applications. *Proceedings of the VLDB Endowment*, 1(1):163–175, 2008.

[51] M. M. Zloof. Query-by-Example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.