Query-through-Drilldown
Data-Oriented Extensional Queries

Alan Dix
Computing Department, InfoLab21
Lancaster University
Lancaster, LA1 4WA
+44 1524 510 319
alan@hcibook.com

Damon Oram
Corporate Information Systems
Information Systems Services
Lancaster University
Lancaster, LA1 4WA
d.oram@lancaster.ac.uk

http://www.hcibook.com/alan/papers/avi2008-query-through-drilldown/

ABSTRACT
Traditional database query formulation is intensional: at the level of schemas, table and column names. Previous work has shown that filters can be created using a query paradigm focused on interaction with data tables. This paper presents a technique, Query-through-Drilldown, to enable join formulation in a data-oriented paradigm. Instead of formulating joins at the level of schemas, the user drills down through tables of data and the query is implicitly created based on the user's actions. Query-through-Drilldown has been applied to a large relational database, but similar techniques could be applied to semi-structured data or semantic web ontologies.

Categories and Subject Descriptors
H.2.3 [**Database Management**]: Languages – *query languages.* H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *query formulation.* H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User Interfaces – *graphical user interfaces, interaction styles.*

General Terms
Design, Human Factors

Keywords
database query, data-oriented interaction, SQL, tabular interface, extensional query, data structure mining, query-by-browsing

1. INTRODUCTION
Traditional database query formulation is intensional, users are forced to formulate their queries in terms of schemas, table and column names. This often involves users in very abstract thinking, Boolean logic for defining filters and trying to understand the way that tables are linked together in joins – especially challenging for well-normalised databases. While languages and tools for this may be a powerful for experts, less experienced users may find them unnatural. Indeed the most successful end-user interaction techniques: web browsing and spreadsheets both keep the user focused on the data itself not meta-level descriptions of the data.

This paper takes the position that for many users a more extensional paradigm based on interacting with data is more easily understood.

Previous work on Query-by-Browsing has shown that it is possible to create filters using a query paradigm focused on data; users interact with and extensional view of a query and a query is inferred through machine learning. This paper presents a technique, Query-through-Drilldown, to enable join formulation in a data-oriented paradigm. Instead of formulating joins at the level of schema, the user drills down through tables of data and the query is implicitly created based on the user's actions.

In the next section, the paper begins by discussing the concept of extensional/data-oriented access. As Query-by-Browsing [5] was the initial inspiration for this work, we describe this in detail and analyse some of the generic issues it highlights. In particular QbB enables the creation of filters by simply allowing the user to select desired rows from a table of data. However, QbB does not have any way for the user to create joins.

Section 3 presents Query-through-Drilldown (QtD), a tableau-based interaction that allows complex multi-table queries to be created without explicit joins. The technique depends on an entity-relationship structure, so we also describe techniques to automatically derive this. Section 4 presents our experiences in implementing and evaluating a prototype of QtD and section 5 compares QtD with other data-oriented forms of browsing including semantic web ontologies. Finally, we discuss planned future work and possible extensions to less structured data.

2. DATA-ORIENTED ACCESS
2.1 Intensional vs. extensional data access
In database semantics, following other areas such as logic, a distinction is drawn between intensional and extensional forms of description. The intensional form is the query in terms of the schema, in relational databases usually expressed in SQL whilst the extensional form is the collection of records.

AVI'08, 28-30 May , 2008, Napoli, Italy

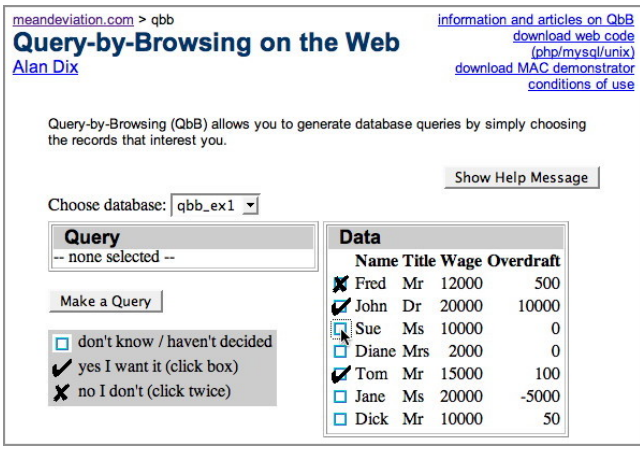


Figure 1. QbB (web interface) – user selects records.

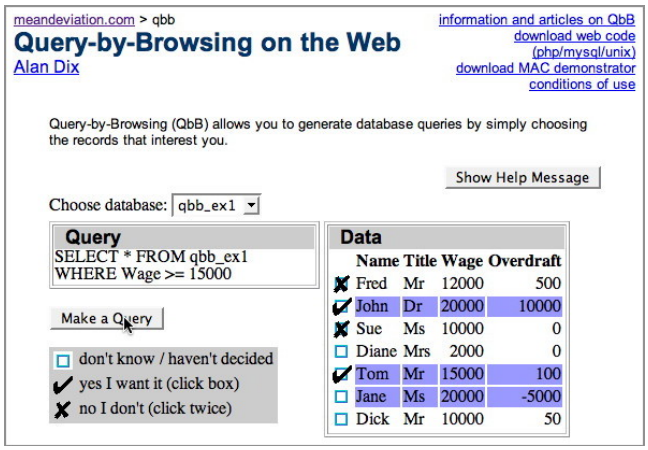


Figure 2. QbB generates SQL and highlights query results.

We see similar patterns in other forms of formally structured data, in particular for semantic web ontologies stored in RDF we have SPARQL queries (intension) and a corresponding graph/set of triples (extension) as an output [14]. The powerful thing about intensional descriptions is that they can be reapplied to new data to obtain precise results, however they are often only usable by experts. Indeed even early studies of database query mechanisms showed that those that incorporated some form of tabular interface outperformed purely textual interfaces such as SQL [7].

Even in information retrieval (IR) systems or web search there is often a Boolean query (intension) giving rise to a set of pages or documents (extension), although the distinction is less sharp. For simple search the distinctions become more problematic as the search terms used are themselves part of the content of the document, however search terms can be re-interpreted over a different collection, so have an intensional aspect – indeed part of the skill of a good web user is knowing which terms to use rather than which pages to visit. In web search and certain forms of bibliographic search, the focus is much more in skimming the data of the results to choose appropriate ones, rather than necessarily tuning the search terms to be precisely correct.

Web and hypertext browsing is perhaps more complex still as the 'schema', such that there is, is at best node + link. The user's focus here is almost solely on the content except in sophisticated systems with multiple link types. This is also true of many forms of graph or tree browsing, although in such case the content may be represented simply by a name or icon.

Similarity-based or recommender systems are also more data oriented, for example, Amazon recommendations are specific books, not specifications of interesting books. Similarly, the Scatter-Gather Browser [11] clusters documents, but presents the clusters in terms of generated summaries – while these are not instances, the summaries are focused on the data content.

In general intensional descriptions are more precise and generalisable, but correspondingly more complex and hard to understand. In contrast extensional descriptions are simpler and more comprehensible, but cannot be easily generalised and hard to be sure of unless checked exhaustively.

The challenge is to use both effectively where they are strong.

2.2 Query-by-Browsing

Query-by-Browsing precisely addresses this issue by effectively turning the traditional query processing pipe on its head, starting with an extensional description and generating an intensional description from it.

QbB was first described in a concept paper in 1992 and later implemented [4,5]. However it is also available as a web demo and the screenshots are taken from that[1].

Figure 1 shows the first stage of use. The user has selected a number of records that are either wanted (ticks ✓) or not wanted (crosses ✗). In this initial stage the user's focus is entirely on the list of records; that is extensional; all the user is doing is selecting positive and negative examples.

After a period the user clicks "Make a Query" and the system generates an SQL query (Figure 2):

```
SELECT * FROM qbb_ex1 WHERE Wage > = 1500
```

In the initial paper this step was described as occurring when the system had sufficient confidence in its inferred query, but in all the implemented systems this is at the user's request.

Looking in detail at Figure 2, we can see that there is both the SQL query in the left hand area and also highlighted items in the listing on the right. The highlighted items are those that would be retuned by the SQL query. That is, the result is *both* intensional (the SQL) and extensional (the highlighted items).

The QbB papers emphasise the importance of this dual representation. Whilst a user may find it hard to produce syntactically correct SQL they may be able to recognise whether it is correct. For more complex Boolean queries the dual representation may make it easier for a user to make sense of the connective – for example the confusing difference between 'and' as used in Boolean logic and its everyday use.

The highlighted records (extensional output) make it easy for the user to verify the query, selecting the appropriate records from

[1] "Query-by-Browsing on the Web". accessed 19 Dec 2007. http://www.meandeviation.com/qbb/qbb.php

those that can be seen. However, the SQL query itself (intensional output), allows the user to verify that the query will also apply correctly to unseen records. This would be important if, for example, the selected records were to be updated in some way … perhaps awarding a pay increase!

QbB uses machine-learning to create the query. The algorithm in the original implementation and the web interface is a variant of Quinlan's ID3 [15], but alternative algorithms are also described [6]. The algorithm used in the extant implementations is guaranteed to give a consistent result – that is the records selected by the query will include all the positive examples and none of the negative ones. However, there may be several queries that are consistent with a given set of positive and negative examples, so while the algorithm is consistent it may not accord with the intention of the user. The user may detect this either because the highlighted rows are not as expected or because the query does not seem right (e.g. the query says "`Wage>=15000`", but the user knows that the key value is really a tax threshold of 14250).

If the user is not satisfied with the inferred query, more positive and negative examples can be given. The highlighted records are again useful as any that are highlighted but not wanted are obvious candidates to be explicitly excluded and vice versa. The user then requests a fresh query and iterates until the returned query is satisfactory. The QbB papers also suggest that the user should be able to interact with the query – particularly easy if the query return format is a Relational Query by Example tableau [20]. That is, the user's input to the system could be a mixture of intensional and extensional elements. However, again this is not implemented in the extant systems.

2.3 Table-based interaction

As well as being data-oriented, QbB is table based. In fact, the basic principles of data-oriented querying could be applied to non-tabular interfaces, it is no accident that in a system designed to be easy for non-experts tables were chosen as a reference implementation. Early studies comparing end-user performance with several database query facilities (including SQL and QBE) found that those facilities that included a tabular interface outperformed those based on a purely textural SQL interface [7]. While there are many times when various forms of graphical or network representations can be useful, tables, however, mundane, are at the heart of many data-intensive interfaces not least the ubiquitous spreadsheet.

While tables are often the output format of choice, they are also used as a central part of almost any information rich interactive environment including lists of messages in email clients, files in a directory or classes in an IDE. They have also been used in various forms of interactive visualization, notably for exploring patterns, correlations and trends in Table Lens [16]. Even Scatter-Gather [11] can be seen as partially table/list focused. In all of these cases it is the records actually selected by the user that are of interest (the extension) rather than any inferred query of criteria.

An interesting exception to this is Query-by-Excel [19]. Here the user uses a spreadsheet that includes extracts from several tables in the full database. Standard spreadsheet functions and formulae are used to link the data in the different table extracts. When the user is satisfied that the Excel spreadsheet it is uploaded into the Query-by-Excel system and the formulae on the extracts are generalised into a full database query or procedure.

Arguably this use of Excel (and indeed much ordinary use) is intensional as the user manipulates formulae. Indeed the power of spreadsheet use is the rapid and incremental turnaround between intensional formulae-focused steps and extensional reflection on the values in the cells.

Query-by-Excel is also particularly interesting as the system can use the formulae to create linkage between tables as well as calculation/selection within them. That is Query-by-Excel can create *joins*, one of the weaknesses of Query-by-Browsing.

3. QUERY-THROUGH-DRILLDOWN

3.1 The concept

Query-by-Browsing demonstrates how data-oriented, table-based interaction can be used to create generic queries. However, a clear weakness is its lack of provision for joins. This raises the question as to whether a similar philosophy of extensional querying can be used to create inter-table joins.

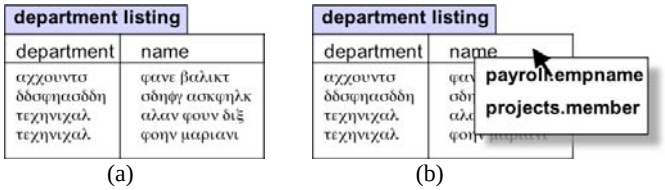
When tables are used in standard interactive applications they may be used to select multiple items for some operation (e.g. to which classifications an uploaded paper belongs) or to allow drilldown to further information. In the latter case this may result in the selected item being opened in its own window (as when an email or file opens) or some sort of hierarchical expansion in place or an adjoining frame.

We will effectively use a form of the last of these. However, typically when rows are 'expanded' the focus is on a *single row*, which already represents a single item. In contrast in a database table listing, it is some of the *columns* that represent foreign keys or shared values that form a point of potential connection to another table. We use columnar drilldown as a way for users to view particular information linked to a given set of records and in so doing implicitly create a join between those tables. For example, if there is a "City Name" column in a table, it could be connected either a table of tourist information about the city or local government. The user's *choice* of which of these to follow effectively creates a join.

3.2 Scenario

To see how this works we will work through a simple scenario.

We assume as a start point that a form of entity-relation structure already exists for the database. That is we know which columns in any table connect to which others. This might have been created by hand, or may be mined automatically. In section 3.4 we will describe methods to achieve the latter, but for now will simply assume it exists.



(a) (b)

Figure 3. Selecting a column to drilldown through

Figure 3.a shows a listing a single table of department staff. In Figure 3.b the user selects the `name` column in order to find out more about the people. The system offers two options as there are two tables that have columns that are linked to the `name` field in `department listing`. In the figure these are named by the table name and column they are linked to, but part of a hand-crafted entity-relation structure might include more meaningful names for the relationships.

When the user selects one of the links from the `name` column the columns from the selected table are appended to the table. Figure 4 shows this in the case where the user has selected to expand the `payroll` record. In this case we have assumed there is a unique payroll item for each person so the table simply gets wider.

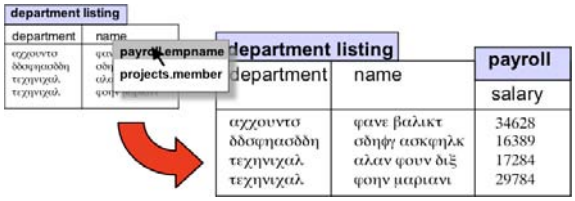


Figure 4. Selected column expands

Only one column is shown corresponding to the case if the `payroll` table had only two columns. In practice tables tend to have many columns and so the user may need to hide unwanted columns. To make this easier the system could default to show the most common columns from the table first (determined by handcrafted meta-data, automated analysis, or personal profile).

Figure 5 shows the SQL generated by this drill down. Unlike Query-by-Browsing we are not currently displaying this to the user in parallel to the tabular interface, but for experts this may be useful in order to generate the query, perhaps alongside a client or end user, and then copy the SQL for later use.

```
SELECT  d.department,
        d.name,
        p.salary
FROM    department d
INNER JOIN payroll p
ON d.empname = p.empname
```

Figure 5. Generated SQL

Several linked tables may be opened and Figure 6 shows the results if the user drills through the name to the `projects` table. Note it is shown at the same level as `payroll` to make clear it is a child (drilled from) the original `department listing` table. In this case, the projects are assumed to be in an m-n relationship with the names from the `department listing`. So in some cases there are several projects listed for each individual and in some cases none.

department listing		payroll	projects
department	name	salary	project
αγχουντσ	φανε βαλικτ	34628	φοβ το δο
δδσφηασδδη	σδηφγ ασκφηλκ	16389	προφεχτ
			φοβ το δο
τεχνιναλ.	αλαν φουν διξ	17284	φοβ το δο
τεχνιναλ.	φοην μαριανι	29784	– NONE –

Figure 6. Additional column for m–n relation.

Note that in the case of m–n relationships a `LEFT JOIN` is generated; that is all rows are retained in the `department listing` table even if there is no corresponding name in the `projects` table (people who are not members of any projects). This is because the user has started with the list of staff members in departments and so it makes sense not to lose these during drilldown. However, it would be equally odd to find extra names appear as it would with a `RIGHT JOIN`. Note that choosing the right kind of join is often confusing even for semi-experienced database users. However, the way in which the user constructs a query makes it obvious which kind of join is required.

Similar techniques can be used to drill down further through the linked tables, to add computed columns, filter and sort [2]. Figure 7 shows the end point of a series of interaction following on from Figure 6. Three computed columns have been added two connected with the `department listing` table and one with the `projects` (indicated by heights of the tabs). The overall table has also been reordered by project name. The relative heights of the table names help the user keep track of the relationship between the tables – the `payroll` table is only indirectly linked to the `projects` through the `department listing`. This is similar to the effect that would have happened if the user had started with the `projects`, drilled through to `department listing` and then to `payroll`.

projects	department listing		payroll			total cost
project	department	name	salary	nos proj = ...	proj cost = ...	= ...
φοβ το δο	αγχουντσ	φανε βαλικτ	34628	1	34628	51464
	δδσφηασδδη	σδηφγ ασκφηλκ	16389	2	8194	
	τεχνιναλ.	αλαν φουν διξ	17284	1	8642	
προφεχτ	δδσφηασδδη	σδηφγ ασκφηλκ	16389	2	8194	8194
– NONE –	τεχνιναλ.	φοην μαριανι	29784	0	#####	#####

Figure 7. Complex query: added columns and reordered (also see colour plate)

3.3 Relationship Model

The relational structure of a database can be thought of as a labelled graph where the vertices are tables and the labels on edges are relationships between foreign keys or shared values:

Schema = < Tables, Reln >

Reln $\subseteq$ Table × Table × SharedColumnFormula

The SharedColumnFormula will typically be a set of equalities between fields, but may be more complex as in an SQL `JOIN` clause. As noted earlier, for hand-crafted structures the relationships could be given meaningful names in each direction.

[2] A longer scenario with more of these features can be found at http://www.hcibook.com/alan/teaching/projects/workspace-drill-down.pdf

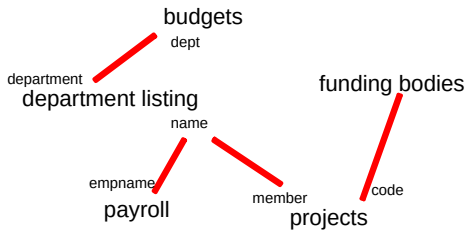


Figure 8. Relationship graph for database

Figure 8 shows an example database relationship structure corresponding to the example in Figures 37.

The query generated by Query-through-Drilldown is effectively a tree where the nodes are tables and the edges relations:

QbBquery = < Tree(Nodes,Edges), NodeMap, EdgeMap >

root ∈ Nodes
parent, child: Edges → Nodes
NodeMap: Nodes → Tables
EdgeMap: Edges → Reln

∀ e ∈ Edges: <t1,t2,c}> = EdgeMap(e)
p = NodeMap(parent(e)) ∧ c = NodeMap(child(e))
⇒ (t1 = p ∧ t2 = c) ∨ (t1 = c ∧ t2 = p)

Note that the mapping between Edges and Tables need not be injective as a table may be returned to during drill down. For example, from the configuration in figure 6 it would be possible to drill down through `projects` back to the `department listing`. This would give for each person in the department a list of the people who are in a project with them. Figure 9 shows a query tree for figure 6 (solid arrows) with the dashed arrow representing the additional drilldown back from `projects` to `department listing`.

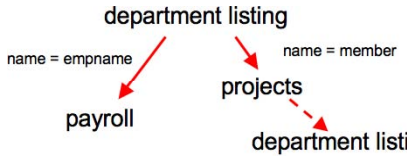


Figure 9. Query tree

3.4 Mining the Model

As noted the relationship model may be constructed by hand in which case meaningful names may be added for many of the relationships. However, for large databases or informal sources (such as a .csv file downloaded from the web) such hand annotation may be infeasible or impossible. Indeed even integrity constraints such as foreign keys are often only maintained implicitly in code and not in the database schema, so it seems likely that some form of automatic structure is needed.

Foreign keys are an obvious first step as they clearly establish a semantic connection between tables. These are most important (and happily most likely to be present) where the keys are simple ids as these are hardest to match implicitly.

Where there is no semantic information available or it is incomplete, the data itself can be used by matching the values in columns across different tables. If there is a high level of overlap between values in two columns then we can infer a relationship. However, this needs to take into account the density of values in their respective domains and especially for integer values. It is common to find id columns in tables consisting mainly of the initial N integers. Without a density check there would be many false positives as columns of ids and similar numbers of elements would overlap even where there is no real relationship. However, ignoring such accidental number range matches does mean that foreign id keys tend to be missed.

The example database that we have been using had a large number of such id fields and so techniques with more semantic information were required. Happily the database in question had large numbers of stored procedures. These procedures can be accessed via a straightforward SQL query (Figure 10).

```
SELECT text
FROM syscomments sc
INNER JOIN sysobjects so
ON sc.id = so.id
WHERE so.xtype = 'P'
```

Figure 10. SQL to access stored procedures

The queries in these formed a rich source to analyse (see figure 11). Wherever a JOIN is found (explicit or implicit in the form of "SELECT …WHERE table1, table2 …") we use the list of fields connecting the two to establish a relationship.

```
SELECT ss.student_id, sname = ss.surname +
', ' + ss.forename, … more fields …
   FROM std s
   INNER JOIN std_snapshot ss
       ON ss.student_id = s.student_id
   INNER JOIN std_address sa
       ON  ss.student_id = sa.student_id
       AND sa.address_type_lid = '000763'
   … 10 more lines  containing 3 more INNER JOINS …
   INNER JOIN org o
       ON ss.org_id = o.org_id
```

Figure 11. Typical SQL in stored procedures

This technique is not guaranteed to find every relationship; indeed in the database we were using with 300 tables it is likely that some potential relationships have never been traversed in previous use of the database. However, where stored procedures are heavily used, they are likely to find the most typical and useful relationships, including most of the important foreign keys.

A full SQL parser could be used for this extraction, but in fact a few regular expressions were sufficient to extract the majority of JOINS and their linkage columns. The exceptions were where aliases were used for table names (which could be captured by more complex regular expressions) and places where the JOIN includes database functions such as SUBSTRING() or INT().

Where stored procedures are not heavily used, the SQL for the queries may be scattered in the source code of many programs. However, databases often have some form of query logging, for example MySQL has a general query log where every query received is recorded [10]. However, compared to the use of stored procedures this is more computationally intensive as there will be many instances of essentially the same query with different parameterisations.

4. PROTOTYPE AND EXPERIENCE

4.1 Implementation

A prototype of Query-through-Drilldown has been created as a web-based interface using .NET framework on the server-side. It was originally hoped that the DataGrid control supplied in Visual Studio.NET web server could be extended. However, it was not possible to modify this to allow the step-down headings and so a custom solution was created using CSS and JavaScript.

The prototype has been developed and tested on our university student information database, which includes over 300 tables demonstrating scalability. However, because of obvious issues of privacy and security, the full and partial screenshots below are all taken from the Northwind, the example database, which forms part of the Microsoft SQL Server 2000.

Figure 12 shows a four table join constructed using the prototype. Note that even in the example database there are a substantial number of rows unlike the simulated screen shots shown earlier.

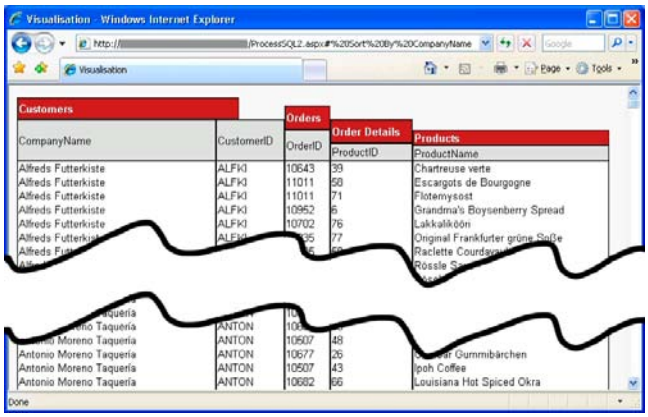


Figure 12. Prototype with four tables joined (also see colour plate)

While there are still many features we would like to add the prototype includes most of the key elements envisioned. For example, Figure 13 shows the query in figure 12 after further interaction adding a computed column and filtering the column based on the `CustomerID` column.

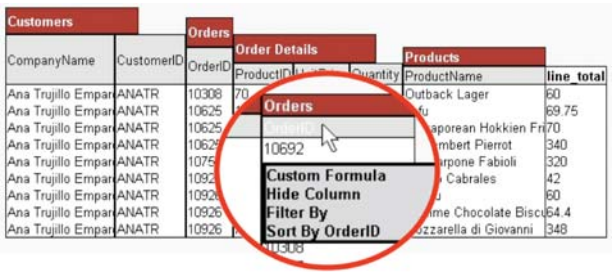


Figure 13. Prototype after filtering and computed column (also see colour plate)

4.2 Evaluation

Formative evaluation has been carried out with two groups of users one non-technical group and one technical group.

4.2.1 Non-technical users

Six non-technical users from an office environment took part in a more formal evaluation. They were initially contacted through their line-manager and then given some information ahead of the session by email describing the purpose of the study, duration and expectations on them. The experiment itself took place in their own premises, but with software installed by one of the authors. Due to security restrictions on the student database and to maintain privacy the Northwind database was used in these experiments. During the evaluation session itself the participants completed a pre-questionnaire to establish prior knowledge and then followed a number of tasks using a written think-aloud protocol (that is, rather than a verbal think-aloud, they were asked to keep notes while working and perform post-task reporting).

None of the non-technical user group had more than passing knowledge of SQL or SQL Server, although they had varying, but not deep, knowledge of desktop databases systems (particularly Access) and, once the term was explained, recognised Query by Example from its use in Access. All had extensive experience in use of spreadsheets.

Many of the user comments referred to fine details of the interface or requests for additional features such as the lack of short-cut keys, sorting on several columns, difficulty of finding certain menus, and confusing error messages. It is always a problem of such evaluations that many user comments relate to superficial interface 'bugs' rather than specific issues relating to the novel aspects. While the former are useful to improve a production system, it is the latter we really need at this formative stage. Happily, some of the comments were indicative of deeper issues.

One such issue was that column names were not regarded as 'user friendly' – they were simply the names of the columns in the database. In desktop databases there is usually provision for having column titles that are more meaningful to users than the column names found in the database schema. In a large commercial database such information is more often embedded in programs or reports. Where a report or UI generator has been used it may be possible to extract the column titles automatically, rather like the JOINS were mined from stored SQL queries. However, even if such column titles were found there may be several such names as the same database row may be presented differently to different kinds of user. This is not just an issue for Query-through-Drilldown, but any system that provides a universal user-interface to databases. In practice this requires semi-automatic user profiling or hand annotation, although this could be inferred if users are allowed to edit the column headings.

Another class of issues were due to the fact that even in the experiment we were using realistic data with substantial numbers of columns and records. When discussing figure 4, we we assumed that unwanted columns had been hidden from the projects table when it was added to the tableaux. However, even when the user only opens essential columns, the tableau grows in width and users complained about horizontal scrolling. This is a problem in any tabular layout, and certainly in more complex spreadsheets. Potentially the focus+context techniques of Table Lens would be useful here [16] or the grouping of columns as used in HyperGrid [8]. Vertical scrolling was also mentioned as a problem, which again might be helped by elision techniques. The

shear number of options created by the database size can also be daunting and may require more structured menus (see Fig 14).

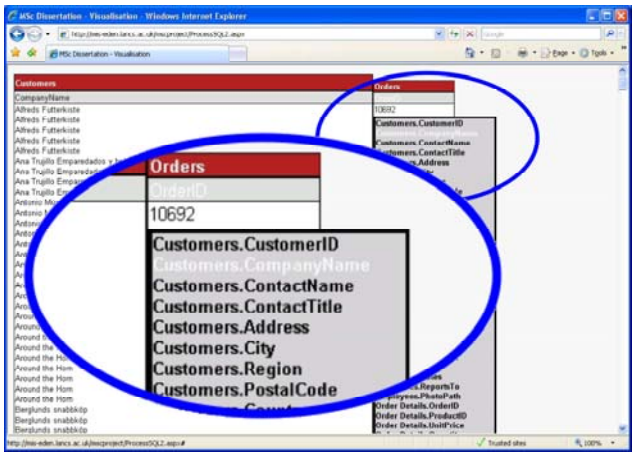


Figure 14. Long menus!

As noted the prototype has been developed using a very large database and, somewhat surprisingly, it has scaled without undue problems. However, users did note some delay on more complex refinements. This is because with a few interactions users were able to create complex queries with multiple joins and very large result sets. If this were submitted as an SQL query a delay a few seconds would seem reasonable, but in an interactive setting second or sub-second responses are expected. The current prototype is completely transaction based and stateless. However, if the interface were delivered as a stand-alone application or if the web interface used AJAX, then it would be possible to know which records the user was currently viewing. This would enable queries to be executed against the sub-selection of visible records substantially increasing the speed and in most cases making query processing proportional to the number of viewed records rather than the total table size. Again these response issues are ones that affect any highly interactive visualisation or query technique.

4.2.2 Technical users

Four technical users, two from an academic support environment and two from a commercial company were recruited for a form of focus group. These users all had high levels of database knowledge and of SQL in particular.

These users were treated very much in a co-designer/participative role. They were given access to the complete source code of the system before the session (in order to allow comments at a system architecture level) and were given a short presentation as to the purposes and vision of the system followed by hands-on time during the discussion session.

As with the non-technical users some of the discussion related to issues that, while important for practical deployment, were not directly related to the fundamental nature of the new technique; for example where configuration options should be stored, better use of the status line and window title, and browser-specific features. However, as these expert users had more knowledge of the purposes of the system they were also able to give more specific remarks about the system concept including the ER mining techniques. In particular they highlighted some of the limitations noted in section 3.4 regarding the regular expressions used to analyse stored procedures.

A major problem they noted, again common to most data visualisation systems, was how to connect to a server, choose a database and an initial table. Once started it becomes easier to navigate based on context, but how does one get started?

The group also noted that it would be useful for users to be able to bookmark states of the system. The ease of interaction meant it was easy to try out something, make a mistake and lose track of where one had been. Even implementing undo when very large SQL statements are being executed 'under the hood' is problematic, certainly requiring either caching or localisation techniques similar to those discussed to improve interactive performance in the previous section. However, explicit bookmarks would be useful too, not just during a single interactive session, but also to return to later. Sharing such useful queries would be one way to alleviate the 'blank screen' problem.

The knowledge of the technical users meant they could question the detailed semantics of Query-by-Browsing. In particular they were interested in the semantics of aggregation when columns were hidden. Interestingly the most intuitive semantics for a user interacting with the system is not the most 'obvious' SQL. Indeed some forms of sorting may require embedded `SELECT`s to create the 'right' answers for a user. The danger of this is that it may end up being confusing for the expert users.

The group suggested adding (the option of) an SQL window to show the actual query being constructed, as is found in Query-by-Browsing. This would fit more closely to QbB's paradigm of optimally combining intensional and extensional representations and also clarify expert users' questions about the semantics of more complex queries.

5. RELATED TECHNIQUES

We have already discussed several table-based interaction techniques in section 2.3. In addition, forms of drilldown or click-through have been used extensively for navigating data, from file browsers to the web, and in some places to aid query constructions.

Some uses of drill down operate at the level of instances of data, such as with links in web pages. Often, like web pages, these replace the current view so that the user 'moves' through the information space. However, rather as we have done with tables, this act of movement can be used to derive more generic queries. In the PESTO [2] system an OO database is browsed by drilling through properties of individual objects instances and each object (or object collection) is opened in a separate window connected to its parent in a graph. However, the path taken effectively forms a generic query and so if the parent object is changed then all the ancestors change accordingly. While Query-by-Browsing shows all the data in a tableau, PESTO focuses on the equivalent of a single row. Each has advantages and there would be arguments for being able to move back and forth between such representations.

Drilldown techniques are an obvious way to interact with hierarchical classifications and have been used extensively in mundane interfaces such as file browsers and also ones involving multi-faceted data or polyarchies [12, 3,17]. Drill-down has also

been used for database queries; one system, also called Query by Browsing [13], uses a file-system-like folder representation where each folder is effectively a table or class, and drilling down through a folder reveals not the rows of the table (instances), but other folders that are linked to the chosen one through the relational structure. While in some ways similar to our system this operates entirely at the schema (intensional) level.

There has been a long tradition of visual query languages [1], but most focus on schema-level constructions. An interesting example is a recent US patent which describes a table-oriented query formulation technique [9] using the relative positioning of tables to represent different forms of relationship, so, whilst displaying data, this is still schema focused.

Query-through-Drilldown uses the relational structure of a database and could easily be used on similar structures, notably semantic web ontologies. In that area m-Spaces [18] are perhaps most closely related as they also tabular layout of instances of classes to perform multi-faceted selections in related classes. However, in m-Spaces the equivalent of the JOIN, that is the specification of relationships between classes, is performed in a configuration step that requires more expertise than the selection interactions.

6. CONCLUSIONS

We have demonstrated how a data-oriented interaction paradigm can be used to create complex queries including joins. Whilst most comparable methods focus on the schema, that is extensional definitions of the query, the focus in Query-through-Drilldown is on the data, that is intensional.

While Query-through-Drilldown was envisaged as an end-user technique, from the evaluation it emerged that it would also be of value to experts in helping them rapidly create complex queries, but to do this would require a more explicit representation of the query, as in QbB.

Query-through-Drilldown has been described here and prototyped as a database interface. However, it was originally envisaged some years ago as a method to operate over other forms of tabular data as found ubiquitously in spreadsheets, word-processor documents and web pages, allowing integration of semi-structured data with fully structured databases. In the future we would like to create some form of adaptors to link such data with more structured databases and semantic web sources.

Given the inspiration for the extensional paradigm is Query-by-Browsing we also intend to integrate QbB filtering with Query-through-Drilldown giving an end-to-end data-oriented query platform.

7. ACKNOWLEDGMENTS

We are grateful to the subjects who gave their time during this project and a special mention for the baby who arrived in the middle. Also thanks to Andrew and Russell for rich discussions way back in 1998 when the first germs of this concept began.

8. REFERENCES

[1] Batini, C. Catarci, T. Costabile, M.F. Levialdi, S. 1991. Visual strategies for querying databases In Proc. of IEEE Workshop on Visual Languages (Kobe, Japan, 8-11 Oct 1991), 183-189.

[2] Carey, M., Haas, L., Maganty, V., and Williams. J. 1996. PESTO : an integrated query/browser for object databases. In Proc. of the Int. Conference on Very Large Databases (VLDB), (Mumbai, India, August 1996). 203-214

[3] Conklin, N., Prabhakar, S., and North, C. 2002. Multiple Foci Drill-Down through Tuple and Attribute Aggregation Polyarchies in Tabular Data. In Proc. of the IEEE Symposium on information Visualization (InfoVis'02) (October 28 - 29, 2002). IEEE Comp. Soc., 131–134

[4] Dix, A. 1992. Human issues in the use of pattern recognition techniques. In Neural Networks and Pattern Recognition in Human Computer Interaction Eds. R. Beale and J. Finlay. Ellis Horwood. 429-451.

[5] Dix, A. and Patrick, A. 1994. Query By Browsing. In Proc. of IDS'94: The 2nd International Workshop on User Interfaces to Databases, P. Sawyer, Ed. Springer Verlag. 236-248.

[6] Dix, A. 1998. Interactive Querying - locating and discovering information. Second Workshop on Information Retrieval and Human Computer Interaction, (Glasgow, 11th Sept. 1998). http://www.hcibook.com/alan/papers/IQ98/

[7] Greene, S. L., Gomez, L. M., and Devlin, S. J. (1986). A Cognitive Analysis of Database Query Production, In Proc. of the Human Factors Society, 9-13.

[8] Jetter, H.-C., Gerken, J., Konig, W., Grun, C. and Reiterer, H. (2005): HyperGrid - Accessing Complex Information Spaces. In: Proc. of the HCI05 Conference on People and Computers XIX 2005. 349-364.

[9] Liang, G. 2007. Method and System for Visual Query Construction and Representation. United States Patent 20070260582. Publication Date: 11/08/2007. http://www.freepatentsonline.com/20070260582.html

[10] MySQL 5.1 Reference Manual, Section 5.2.3. The General Query Log. Accessed 19th December 2007. http://dev.mysql.com/doc/refman/5.1/en/query-log.html

[11] Pirolli, P., Schank, P., Hearst, M., and Diehl, C. 1996. Scatter/gather browsing communicates the topic structure of a very large text collection. In Proc. CHI '96. ACM, New York, NY, 213-220.

[12] Pollitt, A. S., Ellis, G. P., and Smith, M. P. 1994. HIBROWSE for bibliographic database. J. Inf. Sci. 20, 6 (Nov. 1994), 413-426.

[13] Polyviou, S., Evripidou, P. and Samaras, G. 2004. Query by Browsing: A Visual Query Language Based on the Relational Model and the Desktop User Interface Paradigm. The 3rd Hellenic Symposium on Data Management, (HDMS04), (Athens, Greece, 28-29 June 2004).

[14] Prud'hommeaux, E. and Seaborne, A. (eds.) 2007. SPARQL Query Language for RDF. W3C Recommendation, 12 November 2007, http://www.w3.org/TR/2007/PR-rdf-sparql-query-20071112/. Latest version available at http://www.w3.org/TR/rdf-sparql-query/.

[15] Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81-106.

[16] Rao, R. and Card, S. K. 1994. The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In Proc. CHI '94. ACM, New York, 318-322

[17] Robertson, G., Cameron, K., Czerwinski, M., and Robbins, D. 2002. Polyarchy visualization: visualizing multiple intersecting hierarchies. In Proc. CHI '02. ACM, New York, NY, 423-430.

[18] schraefel, m. Karam, M., and Zhao, S. 2003. mSpace: interaction design for user-determined, adaptable domain exploration in hypermedia. In Proc. AH2003 Workshop on Adaptive Hypermedia and Adaptive Web-Based Systems,, 217–235

[19] Witkowski, A., Bellamkonda, S., Bozkaya, T., Naimat, A., Sheng, L., Subramanian, S., and Waingold, A. 2005. Query by Excel. In Proc. of the 31st international Conference on Very Large Data Bases (Trondheim, Norway, August 30 - September 02, 2005). Very Large Data Bases. VLDB Endowment, 1204-1215.

[20] Zloof, M. (1975). Query by example. Proc. AFIPS National Computer Conf. 44, AFIPS Press, New Jersey. 431-438.