DEVise: Integrated Querying and Visual Exploration of Large Datasets

M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic,
S. Lawande, J. Myllymaki and K. Wenger
Department of Computer Sciences, University of Wisconsin–Madison
{miron,raghu,beyer,guangshu,donjerko,ssl,jussi,wenger}@cs.wisc.edu

Abstract

DEVise is a data exploration system that allows users to easily develop, browse, and share visual presentations of large tabular datasets (possibly containing or referencing multimedia objects) from several sources. The DEVise framework is being implemented in a tool that has been already successfully applied to a variety of real applications by a number of user groups.

Our emphasis is on developing an intuitive yet powerful set of querying and visualization primitives that can be easily combined to develop a rich set of visual presentations that integrate data from a wide range of application domains. While DEVise is a powerful visualization tool, its greatest strengths are the ability to interactively explore a visual presentation of the data at any level of detail (including retrieving individual data records), and the ability to seamlessly query and combine data from a variety of local and remote sources. In this paper, we present the DEVise framework, describe the current tool, and report on our experience in applying it to several real applications.

1 Introduction

It is being widely recognized that the traditional boundaries of database systems need to be extended to support applications involving many large data collections, whether or not all these collections are stored inside a DBMS. In this paper we describe an effort to apply the query optimization and evaluation techniques found in a DBMS to work on datasets *outside* a DBMS, and to combine querying features with powerful visualization capabilities. The main goals of the DEVise project include:

- **Visual Presentation Capabilities:** Users can render their data in a flexible, easy-to-use manner. Rather than provide just a collection of presentation idioms (e.g., piecharts, scatterplots, etc.), we have developed a simple yet powerful *mapping* technique that allows a remarkable variety of visual presentations to be developed easily through a point-and-click interface (or easy-to-write 'plug-ins', if necessary). A distinguishing feature is that a user can interactively *drill down* into a visual presentation, all the way down to retrieving an individual data record.
- **Ability to Handle Large, Distributed Datasets:** The tool is not limited by the amount of available main memory, and can access remote data over a network as well as local data stored on disk or tape. Distributed database query optimization is carried out to speed evaluation of queries over the Web; we do not discuss this aspect of DEVise here. The ability to deal with datasets larger than available memory is central to DEVise's support for 'drilling-down' into the data.
- **Collaborative Data Analysis:** DEVise enables several users to share visual presentations of the data, and to dynamically explore these presentations, independently or concurrently (so that some of the changes made by one user are seen immediately by several other users browsing the same data).

By integrating querying with data visualization features, DEVise makes it possible to optimize data-level accesses that arise due to visual operations more effectively; the semantics of how different parts of the visual presentation are 'linked' offers many hints for what to index, materialize, cache or re-compute. Further, memory can be managed by a single buffer manager that supports both visualization and query evaluation.

The DEVise exploration framework is extremely powerful, but to appreciate this power fully, one must work with the system or at least look at several applications in some detail. This is especially true with respect to understanding just how flexible the DEVise visual model (Sections 2, 3 and 4) really is.

The real power of DEVise's visualization capabilities lies in the support for interactively exploring the data visually at any level of detail, including retrieving individual data records. This results in complex queries being generated through simple visual operations, and effective optimization of these 'visual queries' is crucial for interactive use.

In this paper, we concentrate on describing the visual model and visual operations rigorously in set-oriented terms, to provide a foundation for database-style processing of visual queries. The seamless integration of visual queries and database-style queries in DEVise is one of its unique and most useful features.

1.1 Motivating Examples

DEVise is a novel tool in many ways, although many existing tools support some of its features. We now present some example scenarios to illustrate its capabilities, and to help the reader to understand how it goes beyond other related tools. *(For details on these applications, including example full-color DEVise screens, see the DEVise home page at* `http://www.cs.wisc.edu/~devise`)

Financial Data Exploration: In collaboration with the Applied Securities Analysis program in the UW Business School, we've developed an environment for integrated visual exploration of financial datasets from several vendors, including Compustat, ISSM and CRSP. This application illustrates DEVise's ability to access data from a variety of formats, without requiring users to store all data in a common repository, and its use in integrating information from many sources—users can now look for correlations and trends using the combined information from a variety of vendors. It also highlights DEVise's ability to support complex, large datasets: for example, the Compustat data contains records with over 350 fields, and a hierarchical view of this schema, supported by DEVise, makes it much easier to work with. DEVise also makes it easy to 'slice' such multidimensional data along any two axes and to correlate the ranges seen in different slices; thus, it allows a user to navigate through the multidimensional space to identify interesting regions. The total size of the Compustat dataset is over 1GB.

In contrast to the 'wide' Compustat data, ISSM provides trade and quote histories for over 5000 stocks; while each history contains just a few fields and relatively few records, the total amount of data is enormous. (The IBM history for 1992, for example, contains about a million records and is over 20MB.) DEVise makes it possible to browse several histories simultaneously, at various levels of detail, and to move between them easily. Thus, DEVise deals with not only large volumes, but also large data complexity.

R-Tree Validation: The well-known R-tree multidimensional index organizes a collection of points and boxes (which 'bound' spatial objects). Each leaf node (page) contains several points or boxes, and each index node contains several boxes (each of which 'bounds' all the contents of a child page). While developing R-tree algorithms, it is important to understand how different datasets are 'packed' into R-trees, and this can be accomplished naturally by visualizing the tree. An R-tree can be visualized in DEVise as follows. First, note that each box is a data record with fields $(x_{lo}, y_{lo}, x_{hi}, y_{hi})$; this information can be used to 'map' each data record to a rectangle on-screen. By mapping all records in a node, we can 'see' the node as a collection of boxes, and by mapping all the nodes in a given level, we 'see' a horizontal slice of the R-tree. Given such a visual presentation, the visual operations supported in DEVise allow a user to explore the tree, level by level, to scan around in a level and on a page, to zoom into a specific region of the tree, and even retrieve individual data records ('boxes' in leaf nodes, in this example).

The 'visual presentation' of an R-tree can be applied to any R-tree dataset, since there is a clean separation between the definition of the presentation and the data that it operates on; this is analogous to the separation between a query and the input relations in a DBMS. Defining the R-tree visual presentation in DEVise is straightforward, and can be done using a point-and-click GUI.

This example illustrates the flexibility of the presentation mechanism. We were able to develop a sophisticated presentation for a specialized data structure with ease, using the DEVise point-and-click interface for *defining* new visualizations. It also highlights DEVise's ability to deal with large datasets, and demonstrates the value of visual 'mining' for unusual patterns: examining some real datasets (Tiger data for Orange county, a few hundred thousand records), we noticed some unusual arrangements of boxes near page boundaries, and by retrieving the relevant records (simply by clicking on them!) we were able to find some subtle bugs in our R-tree bulk loading algorithms that would otherwise have been extremely difficult to spot.

Family Medicine and NCDC Weather Data: DEVise is being used by the UW Family Medicine department to provide physicians access to data that is collected and maintained independently by five clinics in the Madison area. In addition to the clinic data, which is presented visually in such a manner as to allow physicians to look for certain trends and correlations, we provide uniform access to weather data for the Madison area from the National Climate Data Center (NCDC) data repository. For example, when a physician looks at a series of patient visits in January 96, she may want to look at the temperature in Madison over the same period to see if there is a correlation. (The physicians indicated that they wanted to look for such correlations!)

A common usage pattern is that a physician zooms and scrolls on the visit data, which is local, and the 'linked' temperature view must then be automatically updated. DEVise does this intelligently, by translating visual operations into queries on the underlying data, and utilizes form-based query capabilities at the NCDC archive— one can specify a region and period of interest for a particular time-series— to ensure that only the desired data is fetched. In this example, visual operations generate simple selections on the remote data; more generally, joins of remote or remote and local datasets may be involved, and DEVise generates a suitable distributed query evaluation plan and evaluates the query accordingly.

Cell Image-set Exploration: In this application, we are working with biologists who are dealing with large sets of images of cells, where each cell image has an associated record with over 30 fields, containing information about when and where the image was recorded and details about the content of the image. The biologists working with these images are looking for correlations in the records that can be used to predict pathological features in the associated images. Using DEVise, we have developed a visual presentation that allows a biologist to extract records satisfying certain selection criteria, identify subsets of the selected records that satisfy further conditions, and then retrieve the associated images at any desired level of resolution. The development of the DEVise application was done using a visual interface, using the notions of *views, mappings, links* etc. supported by DEVise, and the biologists' exploration is also done entirely through a visual interface supporting DEVise's notion of *visual queries*. Executing user operations involves a combination of evaluating SQL-style queries and then updating the visual presentation of the results, but the biologists can think (and express desired operations) entirely in terms of what they see on-screen.

If a biologist finds an interesting correlation in the data, he can send an *active report* to a colleague. The active report consists essentially of the definition of the visual presentation, and, at the sender's discretion, parts of the actual data being visualized. The recipient can open the report using her own copy of the data, see the identical screen as the

sender, and then proceed to interactively explore the data further. This is extremely useful for collaborative analysis of the biologists' experimental data. Indeed, the DEVise architecture makes it possible for two or more biologists to concurrently view the same report, so that changes made by one are instantaneously visible to others, although the tool does not support this capability yet. Another feature enabled by the architecture, and which we are currently working on, is called *hyperdata*. Biologists may find several trends, each of which can be shared with others through an active report: in addition, they can create a summary presentation (say) that draws upon the underlying data and also 'points' to the various active reports of interest. This enables a reader of the summary presentation (which is itself just another active report) to interactively bring up any of the referenced active reports simply by clicking on the relevant portion of the summary report; the referenced report can then be interactively explored. Intuitively, an active report is like a photograph that can 'come alive'—users can scroll, zoom etc. on it—and hyperdata enables references to other reports, not just data values.

Soil Sciences Classification: This application illustrates an important point: users often want to generate various kinds of summaries of their data, explore the summary information, and then be able to interactively look at the 'corresponding' portion of the underlying data. This makes it necessary for the visualization component of DEVise to understand the semantics linking the summary and the summarized data. A research group in Soil Sciences is working on automatic classification of forestry-canopy images, which are being generated in large numbers as part of the BOREAS field experiments. They want to process images and classify the pixels into categories like 'trees' and 'sky', and even 'branches', 'soil', 'sunlit leaves', etc. We've combined a tool called BIRCH [15], which was developed for finding clusters of points in multidimensional datasets, with DEVise to create an analysis environment that they are currently using on a daily basis for classifying images. By applying BIRCH, they obtain a collection of clusters, each of which corresponds to a category (e.g., 'trees'). This collection of clusters can be thought of as the summary of the data for that image. A scientist can iteratively see the clusters, refine the parameters of BIRCH, and re-cluster, until the clustering is satisfactory. They can then take the data points that are summarized by a cluster, say 'trees', and identify clusters within this set of points (e.g., 'sunlit leaves' and 'branches').

The crucial point here is how the relationship between clusters (such as 'trees') and the points summarized by them is preserved, and communicated by BIRCH to DEVise. Such integrated interactive exploration of data and summary 'metadata' is an extremely powerful paradigm, and one of the challenges facing us is to develop general mechanisms that allow any analysis tool (e.g., a tool that finds association rules, or even an SQL query that finds averages by some group like department!) to communicate the semantics linking the summary information and the summarized data to DEVise.

1.2 Related Work

DEVise is related to tools that support data visualization, data integration, distributed query processing, Web browsers, and collaborative computing. Clearly, a comprehensive discussion of all the related work is beyond the scope of this paper, but we now briefly discuss the relationship of DEVise to well-known tools in each of these categories.

An introduction to existing visualization software can be found in the surveys by Kornbluh[7] and Braham[2]. From the standpoint of data visualization, DEVise is a general-purpose tool for visual exploration of tabular datasets, unlike tools like Vis5D [5], LinkWinds [6], Traceview [10], ParaGraph [4], etc., that are specialized for a particular application domain. Other visualization tools (e.g., Vis5D, LinkWinds, AVS [14], Khoros [13]) also assume that the datasets are sufficiently small for them to run entirely in main memory; such an assumption limits the ability of the tool to 'go back' to the source data record from its visual presentation. Recently, the Tioga project at Berkeley and the DataSpace project at Bell Labs [11] have also addressed the problem of visualizing large datasets [12, 1], which is indicative of the growing importance being attached to this issue. Their approach, however, differs from ours in important ways. DataSpace is not as flexible in terms of the kinds of visualizations that can be developed, although it supports 3D presentations much better than DEVise (at least currently) does. However, DataSpace assumes that very large datasets are stored in an external database, whereas all its data structures are assumed to fit in memory: thus, it cannot handle visualizations in which the data to be rendered on-screen exceeds these memory bounds. We have taken a declarative approach to defining our visualization primitives, whereas Tioga supports a more imperative, programming-oriented style of defining visual presentations. DEVise is also more comprehensive in its support for distributed query optimization over the Web, its novel buffer management features, and its collaborative computing features.

While DEVise has aspects in common with data integration systems like IBM's DataJoiner, we will not cover these aspects in the present paper; we therefore omit discussion of related work in this area as well.

With respect to collaboration tools, such as groupware like Lotus Notes or workflow products, DEVise is largely complementary. There is no support in DEVise for many of the functions provided by such tools. However, DEVise enables several users to share visual presentations of the data, export such presentations over the Web, and to dynamically explore these presentations, independently or concurrently (so that some of the changes made by one user are seen immediately by several other users browsing the same data). Thus, DEVise adds an important capability for collaborative analysis of large datasets.

In two previous papers ([3, 8]), we reported on early versions of DEVise, with a focus on how its visualization features could be used to develop a variety of applications. While the basic mapping technique has remained unchanged in the current version, the visualization capabilities of DEVise have evolved considerably since, and we have added data transformation/querying capabilites and extended the framework to support collaborative computing. In this paper, for the first time, we give rigorous set-oriented semantics for all visual operations, thereby establishing a firm connection between visualization in DEVise and relational queries, and laying the foundation for database-style optimization of visual queries.

1.3 Paper Outline

The rest of this paper is organized as follows. We describe visual presentations in DEVise in Section 2, queries over visual presentations in Section 3, and illustrate the power of visual presentations in Section 4 by showing how sophisti-

cated SQL queries are generated through intuitive user-level operations on visual presentations. We briefly discuss data transformation/query capabilities and the challenges posed by the DEVise combination of visualization and querying, especially in the context of Web data, in Section 5. We discuss optimization issues in Section 6 and DEVise support for complex tasks such as uniform data/metadata exploration and collaborative data analysis in Section 7.

2 The DEVise Visualization Model

Visualization in DEVise is based on *mapping* each source data record to a visual symbol on screen. Source data tables are called *TData* (for 'tabular data'), and the result of applying a mapping to a TData table is a *GData* (for 'graphical data') table, which is a high-level representation of what is to be painted on-screen. The actual painting is carried out by drawing routines that are typically platform specific (e.g., using X-window primitives or Windows NT drawing primitives); we will not discuss the details of how GData is 'painted' any further. Mappings, TData and GData form the building blocks for abstractions such as *views* and *visual presentations*. We define below the various elements of the DEVise model and its visual idioms. As an illustration we consider visualization of data in the following tables:

```
DEPARTMENT ( DID, DNAME, BUDGET, NUMEMPS )
   department id, name, budget and
   number of employees
ITEMS ( ITEMID, INAME, COST, DID )
   item id, item name, cost of item and
   department selling it
SALES ( DATE, ITEMID, CUSTID, NUMBER )
   items sold, their number and customer ID,
   on each date (mm/dd)
OVERALL_SALES ( DATE, DID, TOTREV )
   total sales revenue by dept id and date
```

2.1 Basic Concepts

TDATA: This is a collection of records with one or more attributes, along with a *schema* that specifies the domain (type) of each attribute. In our illustrative example, each table (DEPARTMENT, ITEMS, OVERALL_SALES and SALES) represents a TData source. We assume that an appropriate type is specified along with the attribute in each schema.

GDATA: To create a visualization, each TData record is mapped to a visual symbol. A GData record has a set of **visual attributes**: *x, y, size, color, pattern,orientation* and *shape*.

MAPPING: This is a function that is applied to a TData record to produce a GData record. The mapping is associated with the TData schema (and not with the data itself—thus the same mapping may be applied to different data sources with the same schema).

Figure 1 shows a visualization of the TData sources described earlier. **V1** shows TOTREV on the y-axis and DATE on the x-axis. Also, each DID is mapped to a different symbol (square, circle, triangle). Each symbol on the screen represents a single TData record. Thus the mapping is (x = DATE , y = TOTREV, shape = DID). An alternative mapping may use a different color for each DID. **V1** is an example of a DEVise **view** and is enclosed in **W1**, a DEVise **window**, both of which we define below.

2.2 View: The Unit of Presentation

A **view** is the basic display unit in DEVise, and consists of three layers: the *background, data display* and *cursor display*. The background includes the actual background on which the data is drawn and decorations such as title and axes. The *cursor display layer* is a data-independent layer that gives additional information about the data display layer. For instance, it can be used to highlight a portion of the data display, as in view **V3** in Figure 1.

Before describing the data display layer, we observe that each view has an associated mapping and TData, and an associated *visual filter*. A **visual filter** is a set of selections on the GData attributes of the view. For instance, a visual filter may select a range of x and y attributes and a certain color. View **V1** in Figure 1 has a visual filter restricting the x-axis to DATEs for the month of July. The *data display layer* is GData obtained by applying the mapping to TData and then selecting the GData records that satisfy the visual filter. We use **VGData** to denote the GData records visible in the data display layer, and **view template**, or **view definition**, to refer collectively to all components of a view except the TData and data display layer. Intuitively, a view template is the data-independent portion of a view, and the VGData, which is computed from the TData, is the data-dependent portion. Together, they define a view completely.

2.3 Coordinating Views

Cursors and *links* are two kinds of view coordination mechanisms in DEVise. A **cursor** allows the visual filter of one view (called the *source view*) to be seen as a highlight in another view (the *destination view*). Cursors are bidirectional in that a change in either the source or the destination view causes a corresponding change in the other view. For instance, Figure 1 shows a cursor with view **V1** as source and **V3** as destination. Notice that the two views have the same x-attribute and the highlight in **V3** extends over the range of DATE values displayed in **V1**, i.e., the month of July. If the highlight is moved over to the month of December, View **V1** will show the data corresponding to December.

A **link** is a constraint that allows the contents of two views to be coordinated. Figure 1 illustrates different types of links supported by the DEVise model.

A **visual link** is a selection condition that is added to the visual filters associated with each of the linked views (obviously, the GData attribute sets for each of the linked views must contain the attributes mentioned in the visual link selection). For example, the views **V1** and **V2** have a visual link on the x axis. This means that both the views display data for the same range of DATE values. So if the user zooms in on **V1** to see the data for the last week of July, view **V2** will also change appropriately.

A **record link** links two views (with possibly different TData sources T_1 and T_2), on a set of common TData attributes, say A. A record link requires that the projection of the VGData on the linked attributes for first linked view (called the master) should act as a *filter* on the TData of the second linked view (called the slave). A record link could be either *positive* or *negative*. Consider the set of TData records, say T, that contribute to the VGData for the first view. (Some TData records do not satisfy the selections in the visual filter for the view, and therefore do not contribute to the VGData for the view.) The positive (negative) record link intuitively says that the second view should behave as if its TData consists of only those records in T_2 that have (do not have) the same A values as those in T. A positive record

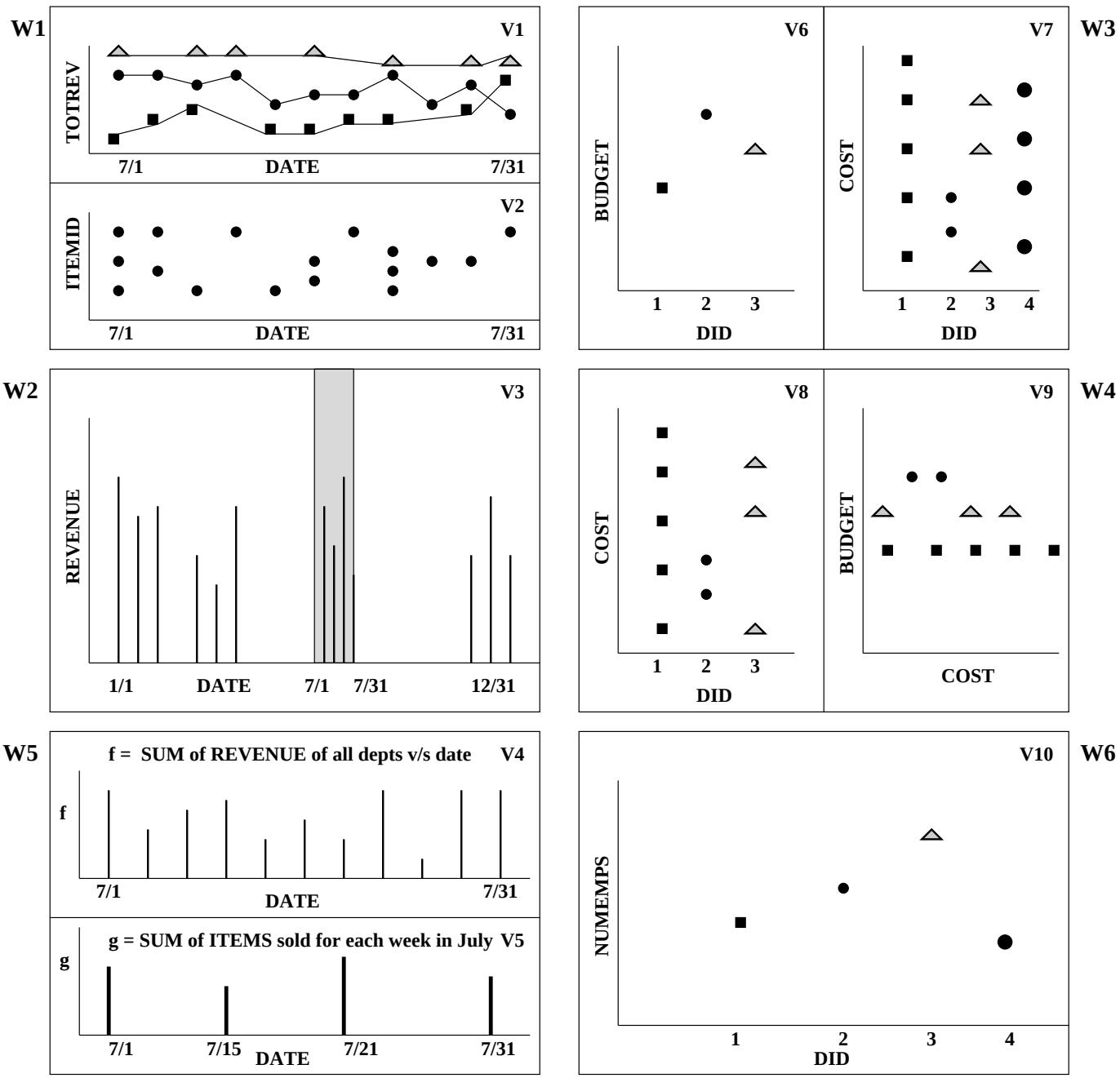

Figure 1: An Example of a Visual Presentation

link is useful for synchronizing two views that display different attribute combinations from the same TData set. A negative record link gives us the ability to do set differences. Figure 1 shows a positive record link from Views **V6** to **V1** on DID. Notice that view **V6** shows three DID records and **V1** shows the TOTREV corresponding to these DIDs only (as indicated by the shape attribute of the GData in both the views). The record for $DID = 4$ has not been selected in **V6** and so does not appear in **V1**. Assuming there are only four DID values, if the record link had been negative, then **V1** would only display TOTREV (for the month of July), for $DID = 4$.

An **operator link** is associated with views that are called the link masters and an operator (such as union, intersection, negation or join). The link creates a TData source that is the result of applying the operator on the TData(s) corresponding to the VGData(s) of the link masters; whether this TData table is materialized or computed as needed in response to user operations is implementation dependent. The user can now define a view (called a slave) on this TData source by specifying a mapping. Once the slave view is created any visual query on the link masters would affect the data in the slave view, just like in a visual or record link. Figure 1 illustrates union and join links. View **V9** has a **join link** from views **V6** and **V7**. Thus, the TData records in **V6** (DEPARTMENT) and **V7** (ITEMS) are joined on DID, to produce a new TData source consisting of attributes BUDGET, DID and COST. Note that the join is performed only on those records (determined by the visual filter) contributing to the VGData of the views. View V9 is a visualization of attributes COST and BUDGET. Contrast this with a *visual join* shown in views **V6** and **V8** where a join is performed on DID by visual alignment of the x-axes of the views. The visual join in this case gives the same information as the join link at a considerably lower cost. View **V10** has a union link from views **V6** and **V7**, on the DID attribute.

A careful reader may have observed that a record link provides a mechanism similar to operator links for intersection and negation operators, without the need for explicitly creating an intermediate TData source. Notice however that a record link, unlike operator links, is always binary.

An **aggregate link** is a link between two views, with an explicit (user-defined) or implicit (value-based) grouping of attribute values for the TData in the first view. The second view visualizes some aggregate function (such as sum, average) performed on each group of records in the first view. For instance, Figure 1 shows an aggregate link from **V1** to **V4** showing the sum of TOTREV of all departments (whose DIDs appear in **V1**) for each day in July. Another aggregate link exists between views **V2** and **V5** showing the total number of items sold for each week in July. The grouping in **V5**) is defined by the user and in **V4** is implicit (every value of DATE).

2.4 Organizing Complex Visual Presentations

A **window** is a collection of views, together with a set of cursors and links on these views. A window has an associated *layout* that specifies the relative location of views within the window. A **visual presentation** is a collection of windows, plus a collection of links and cursors that relate views in different windows. We use **visual template** to denote the data-independent portion of a visual presentation, i.e., a collection of view templates, cursors and links for each window in the visual presentation, plus the links and cursors that span two windows. Thus Figure 1 is an example of a visual presentation. Views **V1** and **V2** are in a window **W1**. Notice the different layouts of views in windows **W1** and **W3**. DEVise supports other layouts such as **tiling** and **stacking** of views. It also provides a mode for transparent overlays of views. These features are not central to the visualization model and we do not discuss them further for lack for space.

3 Visual Queries

Once a visual presentation is created, a user can express selections on the visual attributes of a view, or change a cursor, and we refer to these operations as **visual queries**. A visual query is applied to a visual presentation, and the result is another visual presentation.

Visual queries can be classified into three kinds:

- op_1 Create an x-y 'rubberband selection' on a view, or zoom in/out in a view, or scroll; these are all examples of x-y selections. In general, a user can express selections on any visible GData attributes.
- op_2 Click on a point in the view to display the actual TData record; this is an x-y point selection, but with a different display behavior.
- op_3 Move a cursor highlight by first clicking on it and then clicking on the new position to which it should be moved.

When the user performs one of the above operations on a view V, queries may be generated on views that are linked to V. A *linked query* is a query generated as a side-effect of a visual query.

The presentation of the DEVise visualization model in Section 2 is sufficient for purposes of understanding how to create visual presentations and ask visual queries, but is not sufficiently rigorous to define equivalence of alternative implementation strategies. We now define the semantics of mappings, views, cursors, links and visual queries in DEVise in terms of relational operations on TData. In addition to giving queries a formal semantics, this lays the foundation for database-style optimization of visual queries.

We use the operators selection (σ), projection (π) and function composition ($\circ$).

3.1 Mappings and VGData

A mapping μ is a function that is applied to a TData record to produce a GData record. In the current implementation of DEVise, a mapping is in fact a set of selections $\{\sigma_1, \sigma_2, \cdots, \sigma_n\}$ such that if $< t_1, t_2, \cdots, t_m >$ is a record of TData and $< g_1, g_2, \cdots, g_n >$ is a record of GData, then

$$< t_1, t_2, \cdots, t_m > \xrightarrow{\sigma_1} g_1$$
$$< t_1, t_2, \cdots, t_m > \xrightarrow{\sigma_2} g_2$$
$$\vdots$$
$$< t_1, t_2, \cdots, t_m > \xrightarrow{\sigma_n} g_n$$

The mapping function need not be one-to-one; several TData records could be mapped to the same GData record.

A view V can be represented as a 5-tuple (B, σ^G, μ, T, C) where B represents the background, σ^G the visual filter, μ the mapping, T the TData associated with the view and C the cursor layer of the view. The GData G associated with the view is $\mu(T)$, and the VGData displayed in the view is $\sigma^G(G)$.

3.2 Visual Links

A visual link between two views on attributes L means that the selection conditions in their visual filters that involve only attributes in L are identical. In other words,if views $V_1(B_1, \sigma^{G_1}, \mu_1, T_1, C_1)$ and $V_2(B_2, \sigma^{G_2}, \mu_2, T_2, C_2)$ are two views with a visual link on attributes L:

$$vlink(v_1, v_2, L) \Rightarrow$$
$$\sigma^{G_1} = \sigma^{G_1}_{1-L} \circ \sigma^{G_1 2}_{L} and$$
$$\sigma^{G_2} = \sigma^{G_2}_{1-L} \circ \sigma^{G_1 2}_{L}$$

When any visual query operation changes the visual filter on L, both views change accordingly.

3.3 Record Links

To define the semantics of record links, we must identify the set of TData records that contribute to VGData for the first linked view. We do this by defining an implicit selection on TData, as illustrated in Figure 2. Consider a

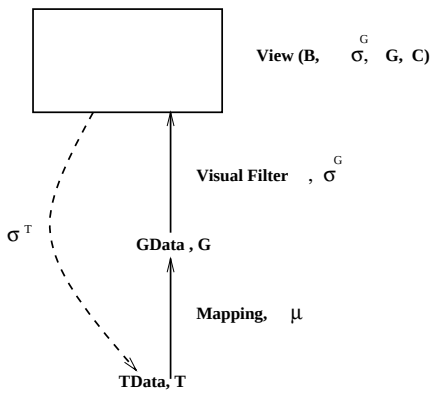

Figure 2: VGData and Implicit TData Selections

view $V_1 = (B_1, \sigma^{G_1}, \mu_1, T_1, C_1)$. The VGData for this view is $\sigma^{G_1} \circ \mu_1(T_1)$. Let σ^T be a selection on TData such that applying this selection and then applying mapping μ yields the same VGData as before:

$$\sigma^{G_1} \circ \mu_1(T_1) = \mu_1 \circ \sigma^T (T_1)$$
$$i.e.,$$
$$\sigma^{G_1} \circ \mu_1 = \mu_1 \circ \sigma^T$$

Equivalently, σ^T can be defined using the following equation:

$$\sigma^T(T_1) = \{t \in T_1 \mid \sigma^{G_1} \circ \mu_1(t) \; is \; non \; empty\}$$

We can now define the semantics of a record link using the selection σ^T. A record link between two views $V_1(B_1, \sigma^{G_1}, \mu_1, T_1, C_1)$ and $V_2(B_2, \sigma^{G_2}, \mu_2, T_1, C_2)$ implies that VGData for V_2 is equal to:

$$\sigma^{G_2} \circ \mu_2 \circ \sigma^{T_1}(T_1) \quad for \; a \; positive \; record \; link.$$
$$\sigma^{G_2} \circ \mu_2 \circ (1 - \sigma^{T_1})(T_1) \quad for \; a \; negative \; record \; link.$$

Notice that the two views related by a record link have the same TData component.

3.4 Operator Links

An operator link consists of master views $V_1(B_1, \sigma^{G_1}, \mu_1, T_1, C_1)$, $V_2(B_2, \sigma^{G_2}, \mu_2, T_2, C_2) \cdots V_n(B_n, \sigma^{G_n}, \mu_n, T_n, C_n)$ and an operator *op*. Suppose $\sigma^{T_1}, \sigma^{T_2} \cdots \sigma^{T_n}$ be the TData selections (as defined for record links) corresponding to the VGData's in $V_1, V_2, \cdots, V_n$. Then we can define a TData T_op given by

$$\sigma^{T_1}(T_1) op \sigma^{T_2}(T_2) op \cdots op \sigma^{T_n}(T_n).$$

Now a view $V(B, \sigma^G, \mu, T_op, C)$ may be defined using T_op as TData. The VGData for this view will be

$$\sigma^G \circ \mu \circ (T_op)$$

The kind of operator associated with the operator link puts certain constraints on the TData's in the master views. The number of master views should be consistent with the arity of the operator *op*. If the operator is union or intersection, then the TData's $T_1, \cdots, T_n$ should have the same schema. Finally, for the operation to be a join an appropriate join condition must be specified with the operator.

3.5 Aggregate Links

An aggregate link between two views $V_1(B_1, \sigma^{G_1}, \mu_1, T_1, C_1)$ and $V_2(B_2, \sigma^{G_2}, \mu_2, T_2, C_2)$ has an associated grouping on attributes of G_1 or T_1 and an aggregation function f. A grouping on attributes $A_1, \cdots, A_k$ provides a grid of values of attributes. The aggregate function is computed for each coordinate $(A_1, \cdots, A_k)$ on the grid. Let T_agg be the TData whose records are of the form $(A_1, \cdots, A_n, f(A_1, \cdots, A_n))$. The second view is some mapping defined on T_agg. The grouping grid may be implicitly specified by having one point on the grid for each value of $(A_1, \cdots, A_k)$, as a range of attribute values or some other manner which we leave unspecified.

3.6 Cursors

A cursor links a source view and a destination view, and has two parts:

1. A selection on visible GData attributes that operates on the *display* layer of the *destination* view, and results in highlighting some range of values for the GData attributes in the selection. For instance, an x-y selection could be shown as a highlighted area with a lighter color. For selections on other attributes, a different highlighting technique would have to be used; we leave this unspecified, as an implementation detail.

2. A selection on the GData of the *source* view that selects the same range as the selection on the display layer of the destination view. Note that the GData layer of the *destination* view is not constrained by the cursor.

Formally, a cursor between V_1 (source) and V_2 (destination) imposes the following conditions on the visual filters of the two views. For the source view, the visual filter should be of the form:

$$\sigma^{G_1} = \sigma^{G_1}_{1-L} \circ \sigma^{G_1}_{L}$$

where $\sigma^{G_1}_{L}$ is a conjunction of range selections on visible GData attributes in L.

For the destination view, the visual filter σ^{G_2} can be any selection on GData attributes, but ranges of attributes selected by $\sigma_L^{G_1}$ that lie within the ranges determined by the visual filter σ^{G_2} have to be highlighted in the display layer.

3.7 Semantics of a Visual Query

A visual query can be represented formally by a 2-tuple (op, V_1) where op is one of the three operations described in Section 3, and V_1 is the view on which the operation is performed. A visual presentation consists of a collection of views, $\{V_1, V_2, \cdots, V_n\}$ and if other views are linked to V_1 by cursors, visual links or record links, additional (sub-)queries are generated on these views.

We now define the visual query (and sub-queries) generated by each visual operation.

Let an operation op be performed on view V_1 represented by $(B_1, \sigma^{G_1}, \mu_1, T_1, C_1)$. Assume that the following (types of) views exist in the visual presentation; this set of views includes an example of every kind of view that can be affected by a visual query on V_1:

- View V_2 represented by $(B_2, \sigma^{G_2}, \mu_2, T_2, C_2)$ with a visual link $vlink(V_1, V_2, L)$ on attribute L between V_1 and V_2.
- View V_3 represented by $(B_3, \sigma^{G_3}, \mu_3, T_1, C_3)$ with a record link $rlink(V_1, V_3)$ from V_1 to V_3.
- View V_4 represented by $(B_4, \sigma^{G_4}, \mu_4, T_4, C_4)$ with a cursor $cursor(V_1, V_4)$ that has V_1 as source and V_4 as destination.
- View V_5 represented by $(B_5, \sigma^{G_5}, \mu_5, T_5, C_5)$ with a cursor $cursor(V_5, V_1)$ with V_5 as source and V_1 as destination.
- View V_6 represented by $(B_6, \sigma^{G_6}, \mu_6, T_6, C_6)$ with an aggregate link on attribute L from V_1 to V_6. Let the grouping be based on values of L. Let the aggregation function be sum(L) and let μ_8 map L to x and sum(L) to y attributes of GData.
- Views V_7 and V_8 represented by $(B_7, \sigma^{G_7}, \mu_7, T_7, C_7)$ and $(B_8, \sigma^{G_8}, \mu_8, T_8, C_8)$ such that there is an operator link from V_1 and V_7 to V_8 with the operator being union.

We now describe the effect of op on V_1. Views V_2 through V_5 are also affected by the visual query on V_1; the effect on some of these other views can be described directly, and in other cases is described by specifying a subquery that is generated as a consequence of op on V_1. Several cases arise depending on the nature of operation op, which can be one of the three kinds op_1, op_2 or op_3 as described in Section 3:

Case 1 ($op = op_1$): This operation is a selection $\sigma^{G_1'}$ on attributes of GData. As a result, the visual filter of V_1 changes to $\sigma^{G_1'}$, which means that the VGData displayed in the view is now $\sigma^{G_1'}(\mu_1(T_1))$.

According to the semantics of a visual link, a sub-query (op_1, V_2) will be generated on V_2, with the selection being $\sigma^{G_1'}$. Thus the VGData displayed in V_2 is now $(\sigma_{1-L}^{G_2} \circ \sigma_L^{G_1'})(\mu_2(T_2))$. This is illustrated in Figure 3.

Let $\sigma^{T_1'}$ be the implicit TData selection determined by $\sigma^{G_1'}$. Then, due to the record link between V_1 and V_3 the VGData now displayed in V_3 changes to $(\sigma^{G_3} \circ \mu_3 \circ \sigma^{T_1'})(T_1)$. If this had been a negative record link, the VGData displayed in V_3 changes to $(\sigma^{G_3} \circ \mu_3 \circ (1 - \sigma^{T_1'}))(T_1)$.

This is illustrated in Figure 3.7.

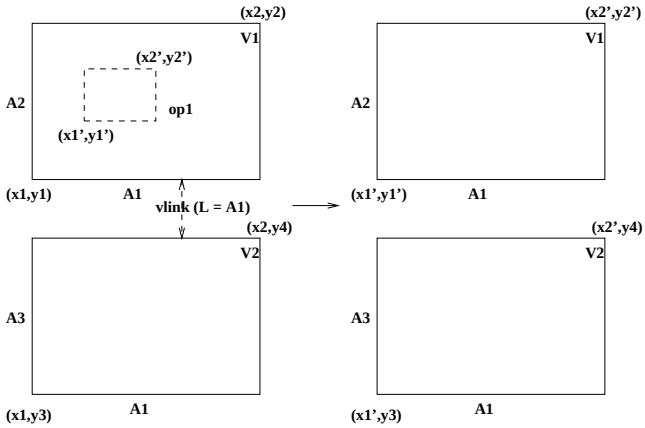

Figure 3: Effect of op_1 in the Presence of a Visual Link

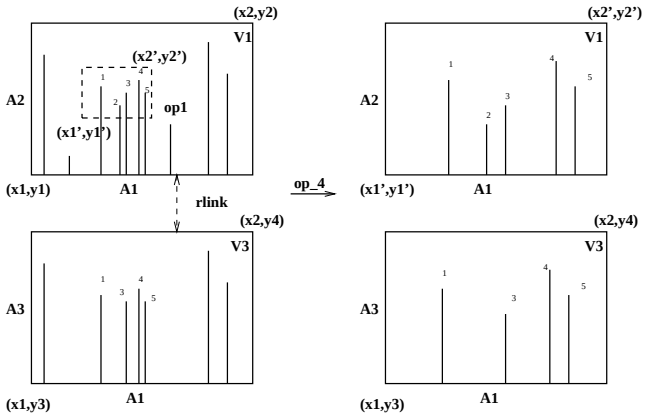

Figure 4: Effect of op_1 in the Presence of a Record Link

Note that if the record link had instead been $rlink(V_3, V_1)$ then the operation on V_1 would have *no effect* on V_3.

Let $\sigma^{G_1'}$ be of the form $(\sigma_{1-L}^{G_1'} \circ \sigma_L^{G_1'})$ where $\sigma_L^{G_1'}$ specifies a conjunction of range selections on visible GData attributes in L. Then the ranges of attributes selected by $\sigma_L^{G_1'}$ that lie within the ranges determined by the visual filter σ^{G_4} will be highlighted in the cursor layer of V_4. This is illustrated in Figure 5.

On the other hand, no change occurs in the VGData of V_5 due to $cursor(V_5, V_1)$. The highlighted area in V_1 may change depending on the visual filter $\sigma^{G_1'}$.

Due to the operator link from V_1 and V_7 to V_8 the TData, T_8, corresponding to V_8 now changes to $(\sigma^{T_1'} \bigcup \sigma^{T_7})$ where $\sigma^{T_1'}$ and σ^{T_7} are the TData selections corresponding to $\sigma^{G_1'}$ and σ^{G_7}. And the corresponding VGData in V_8 is now $\sigma^{G_8} \circ \mu_8(\sigma^{T_1'} \bigcup \sigma^{T_7})$ Note that T_1 and T_8 must be union compatible i.e. have same attributes.

Finally, if $L_1, L_2, L_3, \cdots L_n$ are the values of attribute L, in the VGData corresponding to V_1, then the GData

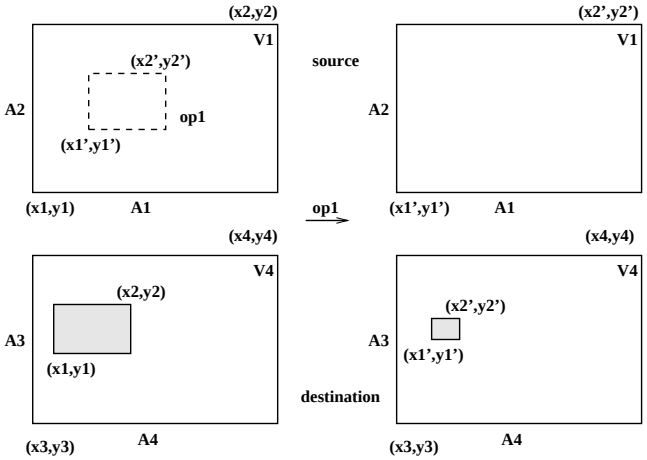

Figure 5: Effect of op_1 in the Presence of a Cursor

(x,y) in V_8 would be $\{(L_i, sum_t(L_i)),\ where\ t\ \in\ \sigma^{G_1^{'}} \circ \mu_1(T_1)\ and\ t.L = L_i)\}$.

Case 2 ($op = op_2$)**:** This operation specifies a GData record given by a selection $\sigma^{G_1^1}$ which is a conjunction of equality constraints. The operation results in a pop-up window displaying the set of TData records:

$$\{t\ \in T_1 |\ \sigma^{G_1^{'}} \circ \sigma^{G_1}(\mu_1(t))\ is\ non\ empty\}$$

This query does not generate any sub-queries on the linked views and does not have any effect on the display of V_1. This is illustrated in Figure 6.

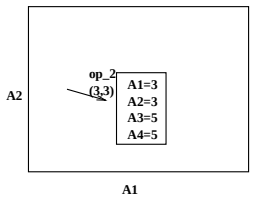

Figure 6: Effect of op_2

Case 3 ($op = op_3$)**:** This operation changes the position of the cursor highlight on the destination view. Thus, the highlight in V_1 is centered around the new position given by the user. As a result a new GData filter $\sigma_L^{G_1^{'}}$ is created. According to the bidirectional semantics of a cursor, the VGData displayed in V_5 is now given by $(\sigma_L^{G_1^{'}} \circ \sigma_{1-L}^{G_5})(\mu_5(T_5))$. This is illustrated in Figure 7.

We will see how these definitions provide a foundation for database-style visual query optimization in Section 6.

4 Visual Queries and SQL

The visual query paradigm enables users who are not database experts to generate sophisticated SQL queries through intuitive graphical operations. We illustrate this now through several examples. The point of this section, however, is not to argue that DEVise can be an SQL front-end (although it is indeed a very good front-end for a large class of SQL queries!). Rather, we demonstrate the close interaction of data visualization and relational querying in DEVise. Of course, the visual presentation offers the—significant!—additional value of rendering the answers in a desired visual

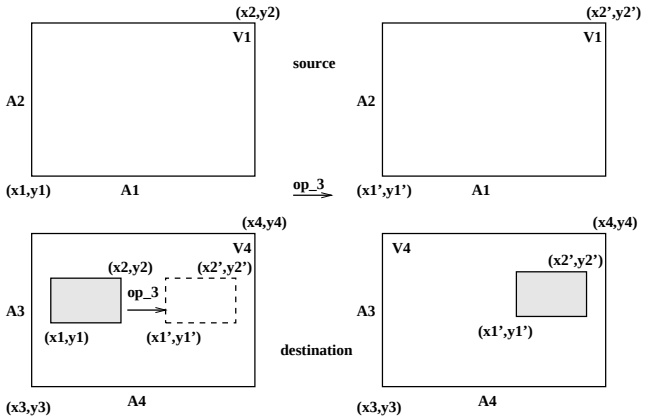

Figure 7: Effect of op_3 in the Presence of a Cursor

form, but we will concentrate on the set of answer tuples in this section.

Consider a TData schema representing the sales data of a company by location: *(latitude, longitude, orders, totalamount)*. This schema is a single data source and serves as a good example of the range of SQL queries that its visualization can produce.

Let T be a set of such TData records. We create the following presentation. Mapping μ_1 gives a scatter plot of *totalamount vs. latitude* and Mapping μ_2 gives a scatter plot of *orders vs. latitude*. This visual presentation is equivalent to the following SQL queries:

```
SELECT (totalamount, latitude) FROM T
SELECT (orders, latitude) FROM T
```

Next, we create a visual link on the x attribute. Now the same visual presentation with a xlink between View1 and View2, is equivalent to the single SQL query,

```
SELECT (totalamount, latitude, orders) FROM T
```

Now we issue a 'rubberband query', i.e., an x-y selection, on one of the views, say View1. Thus we create the selections:

```
10000 < y < 20000 AND 30 < x < 40 on View1
30 < x < 40 on View2
```

due to the x-link.

The equivalent SQL queries on View1 and View2 respectively are:

```
SELECT (totalamount, latitude)
FROM T
WHERE (10000 < totalamount < 20000)
  AND (30 < latitude < 40)

SELECT (orders, latitude)
FROM T
WHERE (30 < latitude < 40)
```

Next, suppose that we modify μ_1 and μ_2 so that longitude is mapped to the *color* attribute. Then, in the original views we can see the result of the following SQL queries on View1 and View2 respectively:

```
SELECT (latitude, totalamount, longitude) FROM T
SELECT (latitude, orders, longitude) FROM T
```

We create the visual link on x between the two views as before. Now if we perform a color selection on any of the views (say View1):

```
color1 (longitude = 50)  < color
   < color2 (longitude = 60)
```

we generate the following SQL queries:

```
SELECT (latitude, totalamount, longitude)
FROM T
WHERE (50 < longitude < 60)    : Query1

SELECT (latitude, totalamount, longitude)
FROM T
WHERE( SELECT( min(latitude)
       FROM Query1 )
       < latitude <
       SELECT( max(latitude)
       FROM Query1 ))
```

Consider an aggregate link giving Sum of totalamounts where the aggregation is done on the latitude. If this link is created from View1 to View3, whose mapping is *totalamount vs. latitude* then we have visualized the SQL query,

```
SELECT (latitude, sum(totalamount))
FROM  T
GROUP BY latitude
```

If a visual filter of $latitude < 30$, is applied to the view, then the query would have the appropriate selection condition added to it.

Finally, we consider an operator link with the operation being join. Suppose we have another table T1 with the same schema as T, for a different company. Let V1 and V2 display tables T and T1 respectively, with the mapping being *totalamount vs. latitude*. Consider a join operator link with V1 and V2 as the masters, join predicate ($T.latitude = T1.latitude$), creating a TData source table with attributes (latitude, totalamount1, totalamount2). We can then create a visualization of totalamount1 vs. totalamount2 in the created table, which gives us the answer to the following SQL query:

```
SELECT (T.totalamount1, T2.totalamount)
FROM  T, T1
WHERE (T.latitude = T1.latitude)
```

In contrast, observe that visual links allow us to display the output of some simple joins without explicitly computing the join. We call such joins **visual joins**. For instance, the information computed with a join operator link in the above example could also be obtained visually: Suppose mapping μ_1 on T is used to create a scatter plot of *totalorders vs. latitude*, μ_2 the same for T_1, and we have a visual link on the x attribute. If these views are laid out one below the other we can see the *totalamount* corresponding to the same *latitude* in the two views. However the queries evaluated in the two views themselves are:

```
SELECT (totalamount, latitude)
FROM T

SELECT (totalamount, latitude)
FROM T1
```

In fact, since a visual link on x implies that the two views have the same range of x attributes, but need not have exactly same attributes we can get a visual 'range' join by simply creating an x-y rubberband on one of the above views. The resulting query is:

```
SELECT (T.totalamount, T1.totalamount)
FROM T, T1
WHERE (30 < T.latitude < 40)
  AND (30 < T1.latitude < 40)
```

Similarly, we could write corresponding SQL queries for the more complex visualization in Figure 1.

4.1 Visualizing an SQL Query

We now show how an example SQL query could be expressed using a visual presentation. We use the schema for sales data described before for TData T_1. Consider the SQL view generated by the following query:

```
SELECT (latitude, longitude)
FROM T_1
WHERE (totalamount > 20000)
  AND (50  < longitude < 60)
```

This query intuitively asks the following question: "In a given geographical area, which locations had a totalamount sale greater than a threshold?"

The following visual presentation achieves this effect. Define mapping μ_1 as *(longitude vs. totalamount)* (View1). Create a rubberband on View1 to select *totalamount* > 20000. Define mapping μ_2 as *latitude vs. longitude* (View2). Create a record link from View1 to View2. This places the restriction that the records displayed in View2 are also displayed in View1. Now select the correct subset of records from View2 using a rubberband $50 < longitude < 60$. View2 now shows the result of the query.

Thus a query on TData attributes can be performed by a appropriate sequence of operations on GData. These examples hopefully illustrate the power of visual queries, although lack of space prohibits a fuller discussion of the expressive power of visual queries.

5 Data Transformation and Querying

As DEVise was utilized in real applications, we repeatedly received feedback from users indicating that more sophisticated database-style query and data transformation capabilities were needed. This might seem strange, considering that we have just finished discussing how many SQL queries can be effectively expressed in DEVise; in part, this was because visual queries in the earlier version of DEVise were not as powerful as the ones described in this paper. On the other hand, in enhancing the expressive power visual queries, we found ourselves implementing much of a database query facility. After considering this issue, we decided to re-design the system to support data transformation and query capabilities within the DEVise engine. DEVise now supports a subset of SQL queries (essentially, queries without nested blocks), and extensions to support sequence queries are under way.

An important feature of DEVise is that queries can operate on both local and remote data sources. At remote sites, if software is available that can provide query profiling and/or evaluation services, the DEVise optimizer seeks to exploit this; otherwise, it will retrieve complete relations and essentially do the rest of the query evaluation at the site where it is executing.

6 Optimization Issues

Operations on the cursor and background layers are inexpensive, and optimization must therefore focus on the impact of visual queries on the VGData components of views. The relational definitions given in Section 3.7 summarize how visual queries change the VGData components of the queried view, as well as all linked views, and suggest several alternatives for query evaluation. For example, selections in visual filters and links can often be used to filter TData records *before* applying the mapping associated with the view. These alternative evaluation strategies must be considered, their cost estimated, and the alternative with the least estimated cost chosen for execution. This is done to a limited extent in the current version of DEVise, and is an area for further work.

To see how new optimization opportunities arise because visualization and database-style querying are combined in a single tool, consider a very simple example: a view (in DEVise terms) V that is created by mapping records from a TData source T. Visual operations on V generate database-style queries on T. If T is a locally stored table, examining the mapping from T to V can tell us what indexes to create on T.

For a more complex example, consider the following scenario. Suppose that a particular selection can indeed be pushed down, and expressed against the TData. If the TData collection is defined by a database-style query, rather than being an explicitly stored set of tuples, run-time query evaluation is used to generate the tuples as needed. Clearly, knowing about the selections that can be expected—this is determined by the visual presentation— helps in planning the database-style query. To take this one step further, a visual presentation might contain several linked views. Even if selections cannot be pushed, the computation of their VGData sets (required, say, due to subqueries generated in response to a user operation on a linked view) can often be combined, especially if the views share a single TData source.

7 Advanced Exploration Tasks

In this section, we consider the use of DEVise for two advanced exploration tasks: *integrated exploration of data and summary information* and *collaborative data analysis*. The latter activity is supported by two DEVise features: *active reports* and *hyperdata*.

7.1 Integrated Access to Data and Metadata

Even with intelligent buffer management, interactive response times cannot be achieved for very large datasets, and too much information is lost by compressing a very large volume of data onto a single screen. A powerful paradigm for addressing this fundamental problem is to let users create summaries of data (which are typically much smaller than the original dataset) and to browse the summaries, or metadata, to get an overview of the entire dataset. Subsequently, users can look at interesting portions of the data in more detail; our experience has been that users find it very useful to interleave the browsing of data and metadata.

The Soil Sciences application described in Section 1.1 is a concrete example of interleaved data and metadata browsing. The visualization of clusters of image points using DEVise is illustrated in Figure 8. The important point to be noted in this example is that the clusters produced by BIRCH can be seen as a summary of the original data. Users explore the clusters produced by BIRCH to obtain a high-level overview of the data, and thereby narrow the scope of subsequent detailed analysis to interesting portions of the data.

The clusters produced by BIRCH are only one example of a summary description of data. Other examples of summaries include:

1. Statistical measures over subsets of the data. Indeed, such summaries are so useful that support is built directly into the current version of the visualization engine of DEVise.
2. Compressed versions of images [9]. Again, DEVise has built-in support for retrieving images at various levels of compression.

7.2 Collaborative Analysis

A visual presentation, as we noted earlier, has two parts: a data-independent *visual template*, and a data-dependent VGData. A user can save a visual template, if desired with some portion of the underlying TData, and send it to another user. The recipient can then re-create the exact visual presentation seen by the sender, if the rest of the TData is also available to the recipient, and continue exploring it. This is supported in the current version of DEVise. We call a visual template that is used in this manner an **active report**: intuitively, it is like a conventional report, except that the reader can interactively explore the data contained in it, i.e., it is 'active'.

A powerful extension that is allowed by the architecture, but is not fully supported in the current version, is that multiple users can share part of a visual presentation and changes made by one user to this part are automatically seen by all users; further, any user can make changes (with a mechanism for passing control between users to avoid conflicting changes). The basic mechanism here is similar to active reports; each user runs a copy of DEVise, and only the operations are communicated between copies (and executed independently by each copy). Clearly, this approach places little or no burden on network bandwith, in contrast to approaches that ship screen-snapshots.

DEVise currently allows field values in TData records to be images or text, and these can be GData field values as well. This allows the creation of visual presentations that look like conventional reports, with text and imagery interleaved with presentations of tabular data (e.g., bar charts or scatter-plots). The DEVise framework also allows TData and GData attribute values to be a *view* or a *window*, capable of being manipulated using all the DEVise power; we call this **hyperdata**. The tool does not yet support this functionality fully, and is being extended in this direction. Clearly, this greatly enhances the value of active reports, since they become much more expressive.

References

[1] Alexander Aiken, Jolly Chen, Michael Stonebraker, and Allison Woodruff. Tioga-2: A direct manipulation database visualization environment. In *Proc. International Conference on Data Engineering*, New Orleans, LA, February 1996.

[2] Robert Braham. Math & visualization: New tools, new frontiers. *IEEE Spectrum*, 32(11):19–36, November 1995.

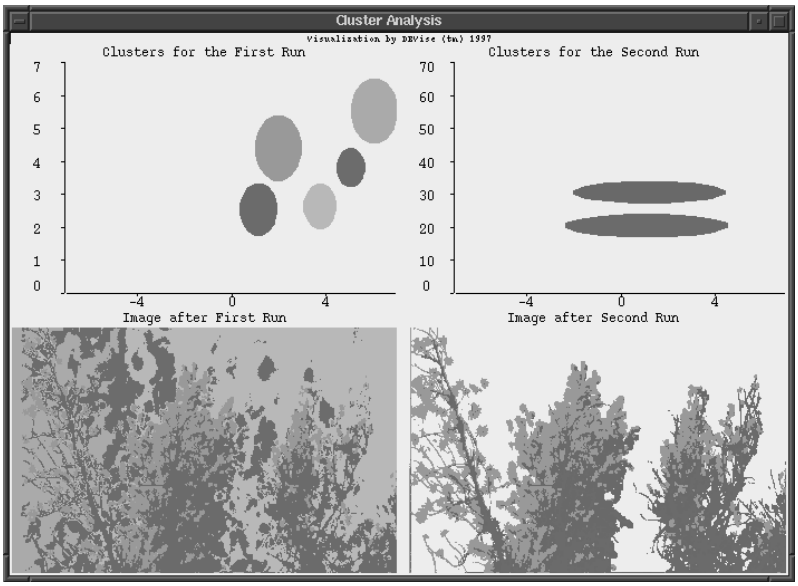

Figure 8: Clustering a Soil Sciences Dataset

[3] Michael Cheng, Miron Livny, and Raghu Ramakrishnan. Visual analysis of stream data. In *Proc. of SPIE – The International Society for Optical Engineering*, volume 2410, pages 108–119, San Jose, CA, April 1995.

[4] Michael T. Heath and Jennifer A. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, pages 29–39, September 1991.

[5] William Hibbard, Brian E. Paul, David Santek, Charles Dyer, Andre Battaiola, and Marie-Francoise Voidrot-Martinez. Interactive visualization of earth and space science computation. *IEEE Computer*, pages 65–72, 1994.

[6] A.S. Jacobson, A.L. Berkin, and M.N. Orton. Interactive scientific data analysis and visualization. *Communications of the ACM*, 37(4):42–52, Apr 1994.

[7] Ken Kornbluh. Active data analysis: Advanced software for the 90's. *IEEE Spectrum*, 31(11):57–83, November 1994.

[8] Miron Livny, Raghu Ramakrishnan, and Jussi Myllymaki. Visual exploration of large data sets. In *Proc. of SPIE—The International Society for Optical Engineering*, volume 2657, San Jose, CA, January 1996.

[9] Livny, M. and Ratnakar, V. Quality-Controlled Compression of Sets of Images. *Proceedings of International Workshop on Multi-Media DBMS*, pages 109–114, August 1996.

[10] Allen D. Malony, David H. Hammerslag, and David J. Jablonowski. Traceview: A trace visualization tool. *IEEE Software*, pages 19–28, September 1991.

[11] Eric Petajan, Yves Jean, Dan Lieuwen, and Vinod Anupam. DataSpace: An Automated Visualization System for Large Databases. In *Proceedings of SPIE, Visual Data Exploration and Analysis IV*, volume 3017. The International Society for Optical Engineering, 1997.

[12] Michael Stonebraker, Jolly Chen, Nobuko Nathan, Caroline Paxson, and Jiang Wu. Tioga: Providing data management support for scientific visualization applications. In *Proceedings of the 19th Conference on Very Large Data Bases*, pages 25–38, Dublin, Ireland, Aug 1993.

[13] The Khoros Group. *Khoros Manual*. University of New Mexico, Albuquerque, NM 87131, 1991.

[14] Craig Upson, Thomas Faulhaber Jr., David Kamins, David Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andris van Dam. The Application Visualization System: A computational environment for scientific visualization. *IEEE Computer Graphics & Applications*, 9(4):30–42, July 1989.

[15] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD*, Montreal, Canada, 1996.