

# An Integrated System for Database Visualization

Dennis P. Groth\*  
School of Informatics  
Indiana University  
Bloomington, IN 47405, USA  
dgroth@indiana.edu

Edward L. Robertson\*  
Computer Science  
Indiana University  
Bloomington, IN 47405, USA  
edrbtn@cs.indiana.edu

## Abstract

*This paper present details of an integrated database visualization system. The system supports the visualization process from an end-to-end perspective. Included in the system is a mechanism for performing transformations to the data being visualized through the use of database relations. This mapping process provides an abstract mechanism for supporting data to geometry transformations under the control of a user-defined, declarative language. The system supports a wide variety of visualization techniques, including scatterplots, bar charts and surface plots.*

## 1 Introduction

Supporting visualization activities within a database environment can focus on a variety of areas, ranging from query formulation to graphical representation. Database systems, architected to support storage, management and extraction of data, do not provide visualization methods. Extraction of data is specified by a query (typically SQL), with the results formatted in tabular fashion.

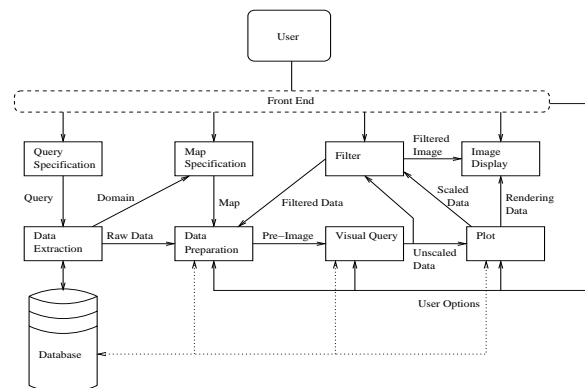
Visualization of the database information is accomplished by transforming tabular data into a graphical representation of the data. Every visualization system that extracts data from a database system must perform this transformation. This transformation process, as outlined in [3], may require restructuring of the data to get it into a form that can be visualized. Specific aspects of mapping from the data to the presentation space are often developed using application specific algorithms and systems. Taxonomic studies of information visualization approaches [4, 2] seek to generalize integration between data and display. This paper describes these techniques

through a proposed architecture. The architecture (and a prototype system) incorporates these types of transformations in a seamless fashion.

By leveraging existing database techniques, however, many visualization applications can be supported in a general way. In particular, the system we present in this paper supports 2D and 3D scatterplots, various types of line and bar charts, 2D and 3D surface plots, as well as others. In addition, the system provides users with the capability to interact with the visualization output. In the remainder of the paper we present the architecture and provide a basic outline and description of our system.

## 2 Architecture

Figure 1 depicts our proposed architecture supporting database visualization.[5, 6] The architecture supports specification of queries for extracting data, as well as the specification of the transformations necessary for visualizing the data.



**Figure 1. An overview of the architecture supporting visualization of database information.**

\* Both authors were supported by NSF Grant IIS-0082407

The following list provides a brief synopsis of the function performed by each component of the architecture.

**Front End** Provides access to each of the components of the architecture through an integrated user interface.

**Image Display** Presents the rendered visualization to the user and provides certain direct manipulation capabilities.

**Plot** Creates the rendered visualization from the input data.

**Filter** Provides the user with tools for selecting data behind the visualization for the purposes of either exclusion or selection for further processing.

**Visual Query** Provides the user with options for defining how the input data relates to an aspects of the output form.

**Map Specification** Provides capabilities for defining and constructing maps using a declarative language, which allow for externalizing the data transformation process.

**Data Preparation** Provides a mechanism for transforming data into a form that is necessary for the visualization.

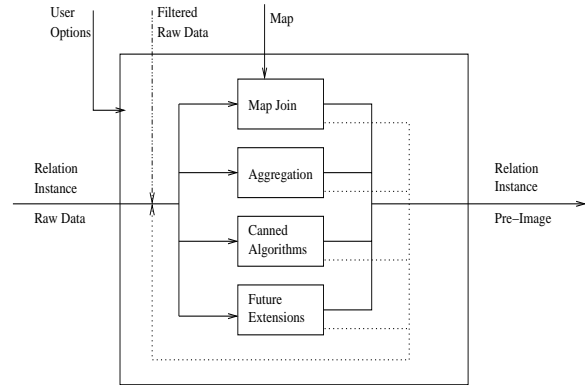
**Query Specification** Provides the user with tools for creating queries, which can be stored and executed on demand.

**Data Extraction** Executes queries against the underlying relational database and returns a relation instance to the calling module.

The key element of the architecture is the data preparation component, which enables the user to define and implement the appropriate data transformations for their application. Figure 2 provides a detailed view of this module.

While the data preparation module may contain arbitrary sequences of data transformations, the module maintains the closure property of databases. The module takes as input a relation instance and returns a relation instance. This approach simplifies the plotting component, since it always works over a consistent input representation.

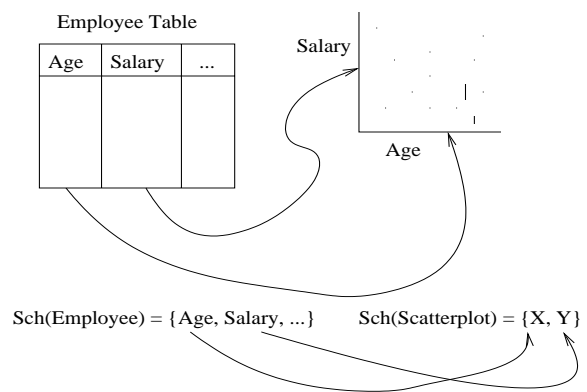
The architecture supports transformations according to user-defined maps, which add both order and scale to the data being manipulated. A logic-based language provides a mechanism for defining the maps, which are stored in the database as relations. The application of a



**Figure 2. A closer view of the data preparation module. Note that the architecture supports arbitrary sequences of transformations.**

map uses relational database join operations. Note that this approach abstracts the transformation process into an externally modifiable object (the map), allowing for transformations to occur without actually changing the original data.

The actual mapping between the data and the visual representation is designed analogously with a query. In database terms, a query is a mapping from an input relation instance to an output relation instance. For visualization purposes, we consider each display type (scatterplot, line graph, surface, etc.) as having a fixed schema. For example, a two-dimensional scatterplot has the schema  $\{X, Y\}$ , in which each  $x, y$  pair represents a point in space, as shown in Figure 3.



**Figure 3. Query mapping age, salary to X, Y.**

Using our approach, new visualization display formats need only define a schema in order to describe its external interface. Internally, of course, the system

needs to support the display. However, the standardization of the interface between the data and the display greatly simplifies this process.

Because of the standard way in which data is accessed and passed through the modules of the architecture, each element of the display can be traced back to the original data. For example, the user can drill down through the user interface to see the raw data behind a point. In addition, it is possible to extract points of interest in the display and reincorporate them in other visualizations, either to show context, or as a filter.

### 3 User Interface

In this section we provide details on an implementation based on the previously described architecture. The system is written entirely in Java 1.3. Database access is provided via the JDBC API. The visualizations are generated using the Java 3D API. A client application provides access to the specification of data, queries, maps and visualizations. A separate, server component provides access to the mapping process, off-loading more costly data transformation processes to improve performance.

Users interact with the system through a graphical user interface that is designed using a desktop metaphor. Data, maps and visualizations are organized within an *Application* object. Using the application manager interface, a user adds various objects to the desktop. In order to give a flavor for the process that a user employs, we provide a series of screen captures for a simple application. As shown in Figure 4, the user right-clicks with the mouse on the desktop and a popup menu is displayed. The menu provides access to the various objects that can be placed on the desktop. Figure 5 shows a list of available input data sources, in the form of user-defined queries.

The example data we are using is unemployment data, downloaded from the U.S. Department of Commerce. The data is comprised of (*Year, Month, Rate*) triples for years 1948-2001. The data is already numeric, but we will still apply a map to the month field in order to demonstrate the process. In particular, we will map the months of the year to quarters, using the following rules:

```
1 <- month <= 3
2 <- month <= 6
3 <- month <= 9
4 <- Else
```

Using the same process as before, the user adds the map to the desktop. Then, as shown in Figure 6, the user

right clicks on the query object and selects the "Connect" option. Then, the user moves the mouse over the map object and clicks the mouse button. Feedback is provided to the user during this process by coloring the connecting line from the query to the map. If two application objects are connectable, the color of the line is green, otherwise it is red and the user cannot connect the objects. When the objects are connected the server component applies the map the input data. Figure 7 shows the result of the operation, in which a new object is added to the desktop that represents the input data transformed according to the map.

The system currently supports a variety of display types, including histograms, scatterplots, parallel-coordinates [7], and surface plots. While certain display types may be less than three dimensional, each of the display types is implemented within a 3D interface.

With the map applied to the data we can plot the data using similar steps. Due to space limitations, we describe the process. First, the user adds a plotting object to the application, in this case a 3D plot. After connecting an input data source to the 3D plot object, the user is presented with display options based on the input data and the plot object. The user can select any of the input attributes as the data being visualized. In addition, the user can select how the data is to be displayed. For example, the user can choose to display the data as spheres in the 3D space. The user has control over the color of objects, which can be based on an attribute of the input data, or a derived value such as frequency.

With the data connected to the plot object and the appropriate options selected, the user can view the resulting visualization by right-clicking on the plot object and selecting "View". Figure 8 shows the unemployment data in using a 3D scatterplot display. The original, unmaped data is shown in the right pane of the figure.

The visualizations that we create allow for interactive manipulation by the user. Standard features, such as zooming, translation and rotation of the display are supported. The user can drill down into the visualization, using the mouse to display the underlying data values. Similar to brushing [1], portions of the visualization can be selected with the mouse. The selected points can be saved to the application desktop for further use. For example, the selected points can be viewed either in a different context, or in a different display type in seamless fashion.

Multiple visualizations can be combined into a single display. The user accomplishes this by connecting multiple plot objects to a combined-plot object on the application desktop. The system supports three combination modes: tiled, offset, and overlaid. The tiled format displays a two-dimensional grid of the connected visualiza-

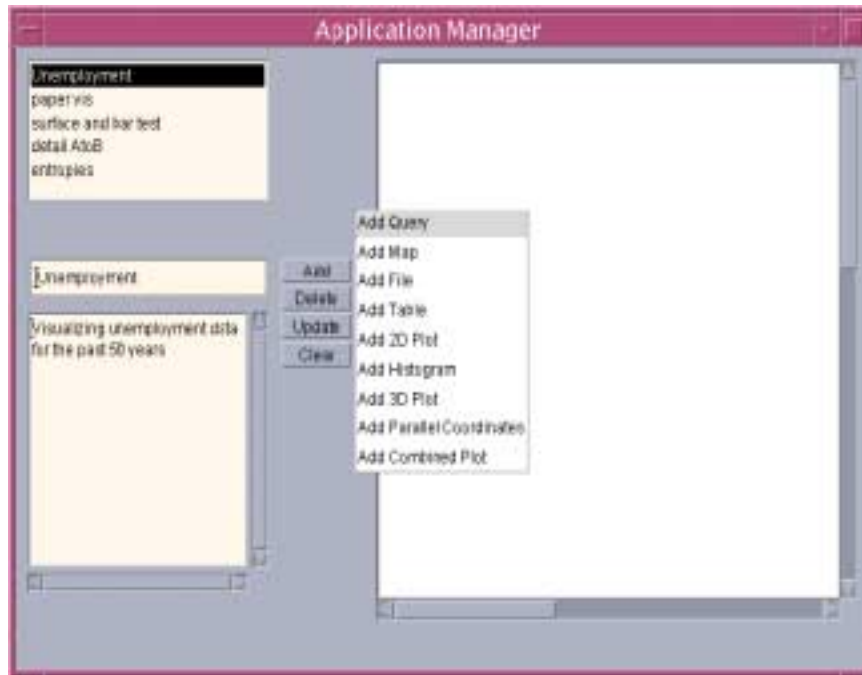


Figure 4. The user right-clicks with their mouse to see a menu of available application objects.

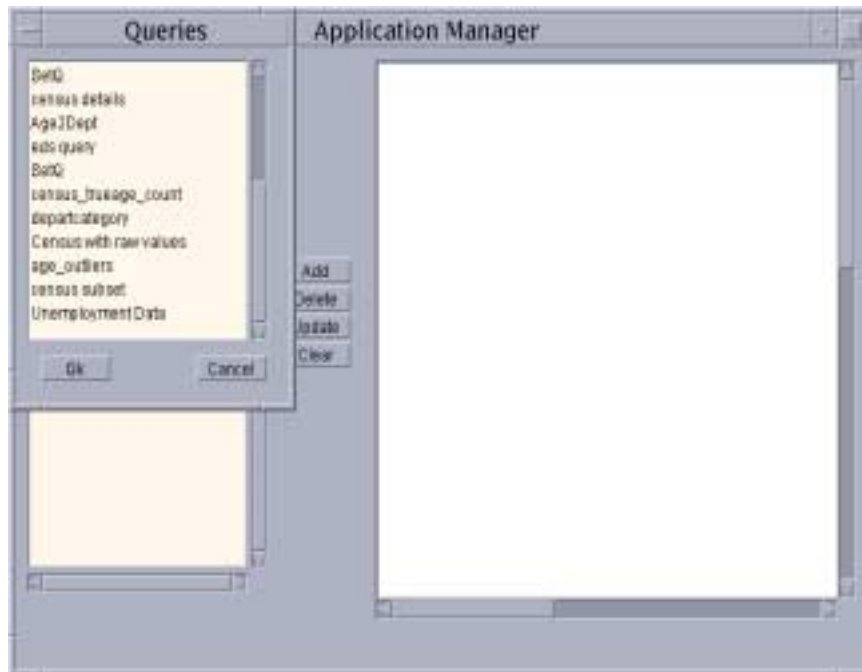


Figure 5. Selecting "Add Query", the user is presented a list of available queries that can be visualized.

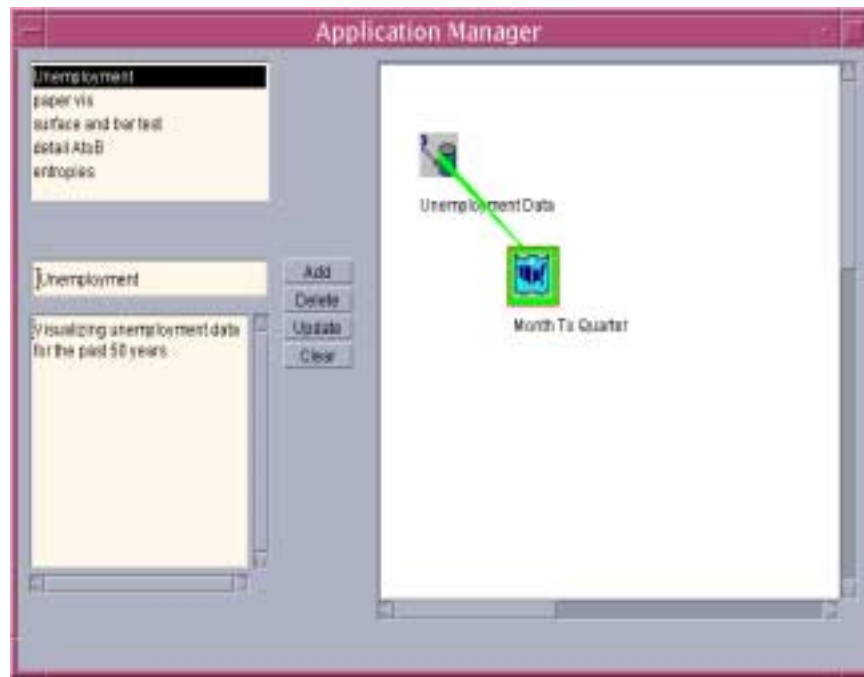


Figure 6. The user uses the mouse to connect an input data source to a map.

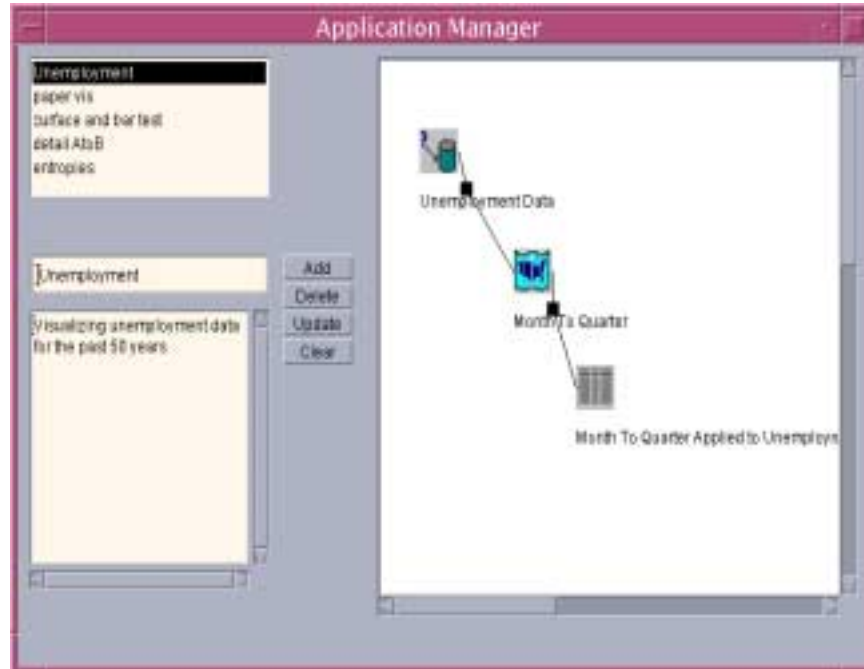
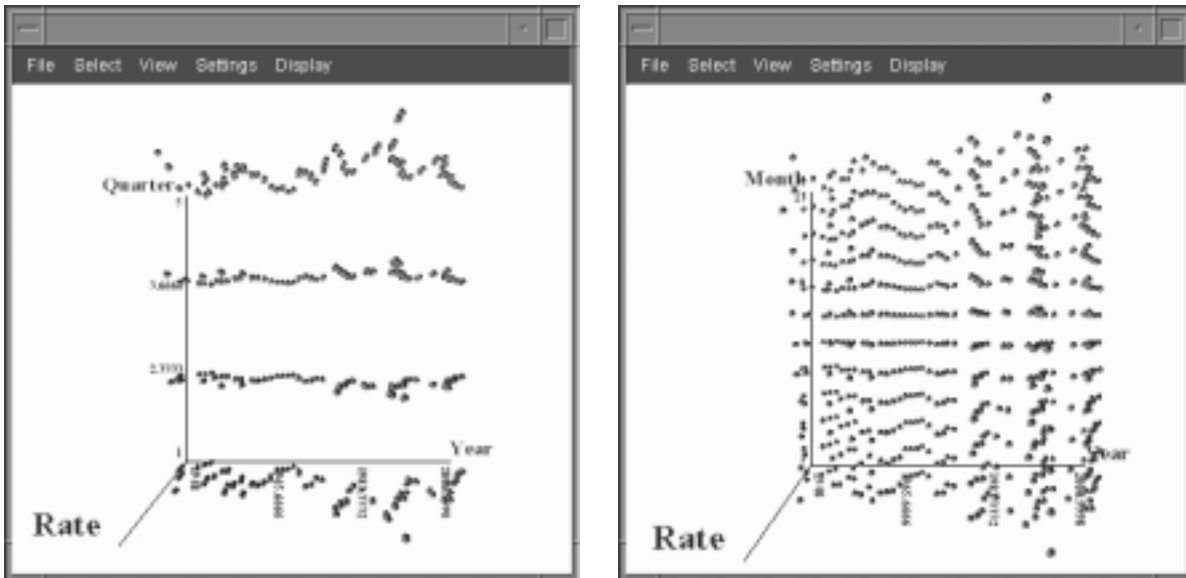


Figure 7. After the map has been applied to the input data, a new object is automatically added to the desktop.



**Figure 8.** The left pane shows the mapped unemployment data within a 3D scatterplot. The right pane shows the original data in its unmapped form.

tions. The offset format displays the connected visualizations offset from each other along the Z-dimension. The overlay format displays each visualization within the same space.

#### 4 Conclusion and Future Work

This paper presented an overview of an integrated database visualization system. The architecture of the system incorporates the mapping process from data to geometry, elevating and exposing the concept of a map. The user interface provides for a building-block approach to the creation and management of visualizations.

Future work with this research allows for numerous directions. First, by abstracting the mapping component, we have created a separation between the visualization and the data, which provides for more flexibility for users. Another area that we are exploring is focused on visualizing database structure, such as functional dependencies, which would enhance a database designer's understanding of data.

#### References

[1] BECKER, R. A., AND CLEVELAND, W. S. Brushing scatterplots. *Technometrics* (1987), 127–142.

[2] CARD, S., AND MACKINLAY, J. The structure of the information visualization design space. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '97)* (1997), pp. 92–99.

[3] CARD, S. K., MACKINLAY, J. D., AND SHNEIDERMAN, B., Eds. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, Inc., 1999.

[4] CHI, E. A taxonomy of visualization techniques using the data state reference model. In *Proceedings of InfoVis 2000 (Salt Lake City UT, October 2000)* (2000), pp. 69–75.

[5] GROTH, D. P., AND ROBERTSON, E. L. Architectural support for database visualization. In *Proceedings of the Workshop on New Paradigms in Information Visualization and Manipulation* (1998).

[6] GROTH, D. P., AND ROBERTSON, E. L. An integrated approach to database visualization. In *Proceedings of the Conference on Advanced Visual Interfaces (AVI 2002)* (2002).

[7] INSELBERG, A., AND DIMSDALE, B. Parallel coordinates for visualizing multi-dimensional geometry. In *Proceedings of Computer Graphics International '87* (Tokyo, 1987), Springer-Verlag.