ARCHITECTURAL SUPPORT FOR DATABASE VISUALIZATION

Dennis P. Groth

Submitted to the faculty of the Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy

in the Department of Computer Science

Indiana University

May 2002

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

Doctoral Committee

Edward L. Robertson
(Principal Advisor)

Dirk Van Gucht

Andrew Hanson

Andrew Dillon

For Melanie, Elizabeth and Joseph

Acknowledgements

The amount of space given to acknowledge support seems small compared to the amount of help I have received. Never the less, I will try and convey my deepest thanks to the people who have made a difference in my work and life.

I would like to acknowledge the members of my research committee. First, and foremost, I am especially thankful for all of the assistance, direction, and focus provided by Ed Robertson. It would not be an understatement to say that this work would not be possible without his direction. I would like to thank Dirk Van Gucht for instilling an interest in database research. In particular, Dirk's introduction to query languages, including Datalog, was great motivation. I would also like to thank Andy Dillon for demonstrating that systems are, in fact, meant to be used by people, and that developing an appreciation for this will ultimately lead to better systems. Last and not least, thanks to Andy Hanson for all of the friendly advice, which certainly contributed to the successful completion of this particular piece of work.

My deepest thanks to Memo Dalkilic, who showed great friendship, encouragement, and support throughout my time here. It is fair to say that without his friendship I would not have been nearly as motivated, or have had even a fraction of the fun that I have enjoyed. He was always there for me whenever I needed him.

Thanks to all of my other friends in the lab, including Chris, Cathy and John. To Jit, who was helpful when I first joined the lab. In addition, thanks to the numerous students who added their efforts to my work: Deepa, Abishek, Arvind, Deepa, Jeff and Susan. I would like to thank the members of the Computer Science department for their assistance. In particular, Dennis Gannon, Sherry Kay, Linda McCloskey and Pam Larsen. The technical support staff has been incredible, especially Rob Henderson.

When I first came to Bloomington I was lucky to meet and work with some very talented people in UITS. Thanks to Don Grinstead, Barry Walsh, Steve Gribble and Jeff Morris for the work, as well as several challenging problems. I would also like to thank the 65 volunteers who participated in the usability experiments.

My numerous friends back in Chicago have also provided support, encouragement, and laughter through numerous lengthy email exchanges that will certainly continue. In no particular order, thanks to Paul, Rich, Mark, Ross, Jim, Bart, Pat, Ted, and Sinon. (CRSRIH)

Outside of my academic studies and research I have developed an intense interest

in the martial arts. I would like to thank my instructors Steve and Linda Scott for instilling this interest. I would also like to thank my training partner Robene for teaching me intensity, and especially for not hurting me (too much). Thanks to my other friends Sarah, Gary, Rob (The Total Package, aka "The Little Angry Man") Lee, Eric, Jason, Jenny, Aaron, Roy, Randy, Judy, Mike and Trisha.

Of course, my wife bears the most responsibility for the successful completion of this research. To my wife Melanie, for all of her years of assistance - this is really her research too. For Elizabeth and Joseph, who have grown up in Bloomington, it was worth it. To my mom, Kathy, I thank her for everything she has ever done for me. To my dad, Dan, I thank him for teaching me to work hard. I would especially like to thank my in-laws, Jim and Jan Fetzer, for their support. Thanks to my brothers and sisters: Mike, Sue, Teresa, Peggy, Tom, Jim, John and Jennifer. Of course, other family members: Darlene, Bob, Jackie, Bill, Wally, Roslyn, Eric, Bryan, Megan, Haley, Brett, Harmony and Laura. I could keep on going, but in the interest of time (and space), thanks to everyone else.

When you commit to completing a doctorate many people may see the dissertation as the culmination of their studies. I see it differently. While this dissertation is a significant accomplishment for me personally, I see it as just the beginning:

"It ain't over till it's over." - Yogi Berra

Abstract

The rapid proliferation and growth of database management systems has resulted in the retention of massive amounts of information for data processing and analysis needs. Many data processing requirements can be satisfied through the use of traditional database languages, such as SQL. These languages retrieve and present query results in record-oriented tables. The table of records format is best for presenting every record, but it cannot give a feel for the overall character of the data set.

Database visualization differs from other types of information visualization due to the diverse nature of the data stored in the database. The attributes used to organize the presentation may not have a regular scale and, in fact, may not even have a semantically meaningful order. Data can be categorized based on whether it has inherent order or scale. For example, numeric data has both an order and a scale. In contrast, geographic data (latitude, longitude) can be easily scaled according to a well-defined metric, yet ordering is not as straightforward. This problem is further complicated by the frequent use of reference values in database systems, for which

the order and scale are defined by an external application and cannot be inferred by the type of data.

This thesis presents a system architecture to address the problem of database visualization. The essential and unique component of this architecture is the mapping functionality, which adds order and/or scale to the data. A map may take into consideration the domain of the data as well as other information. The features of a particular map can be programmed to support a wide variety of visualizations. Naturally, the architecture supports the addition of new maps in a modular fashion. In addition, the architecture contains the following more conventional components: query specification, database storage, filtering, plotting and image display.

In addition to the development of the architecture, we propose to analyze the usability of the database visualization system by conducting an in-depth usability evaluation. The evaluation will be structured to measure the effectiveness and efficiency of users solving realistic problems using the mapping language.

Contents

List of Tables

List of Figures

1

Introduction

With the advent of knowledge discovery in databases (KDD), databases have moved beyond operational support of a business. Indeed, as vast repositories of critical information are being amassed at an ever increasing rate, users are now interested in applying advanced data mining techniques in order to discover previously unknown information.

Described in [FPSS96], the KDD process is frequently depicted in terms of a number of iterative steps, as shown in Figure 1.1.

A critical aspect of the process is the implied interaction with a user. Obviously, the user is involved with problem selection, as well as the interpretation of the results. Often, the user may review the results and develop a more refined problem statement, which initiates another iteration of the process.

Frequently, the KDD process is associated with specific methods that are employed

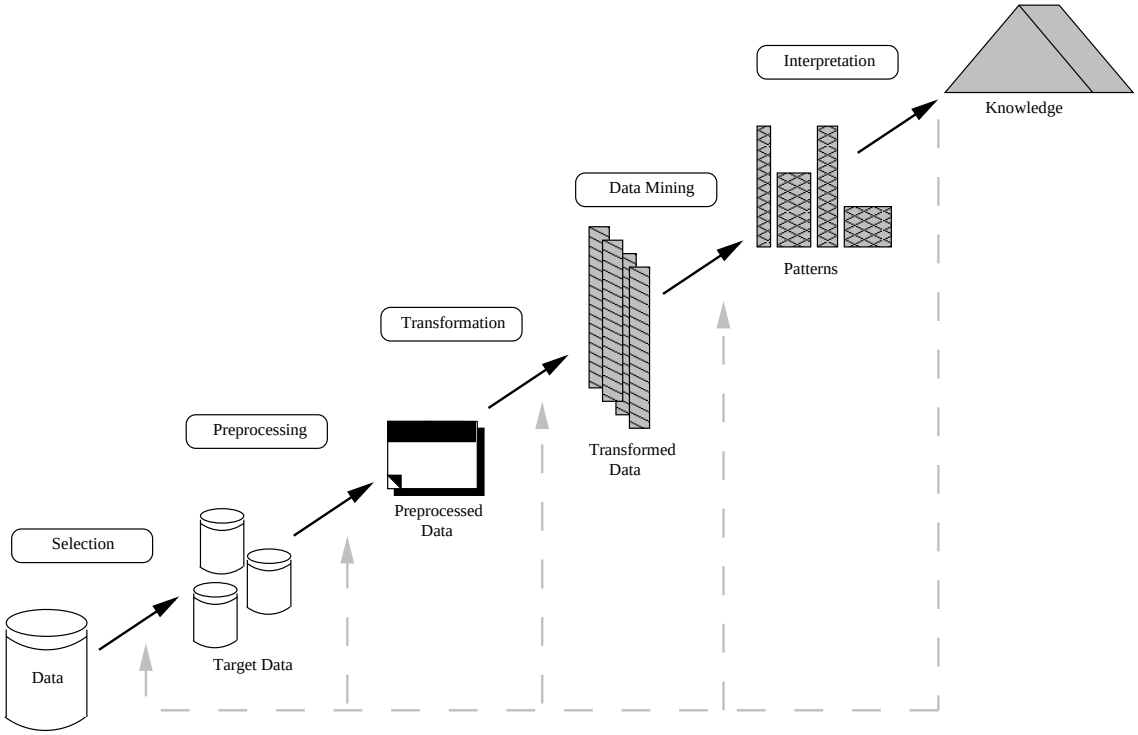


Figure 1.1: The KDD Process

within the data mining task. We delineate the most common methods in order to provide motivation for the problem we are addressing with this research.

Classification determines a function that maps data items to predefined classes. Classification can be used to predict the value of unknown variables, or future data items.

Regression determines a function that maps data items to a real-valued variable. Regression techniques can be used when correlations are significant in the data.

Clustering determines a set of categories or groupings of data items.

Summarization calculates descriptive statistics from data items, such as mean and standard deviations.

Dependency Modeling determines a model that describes dependencies between variables. A common use of this method is to generate association rules.

Deviation Detection discovers significant changes, typically over time, among data values.

There are many surveys in the literature for these methods. Of particular note are [FPSS96, BA96], which outline the process in greater detail.

Visualization and KDD

As a segue to the problem addressed by this research, let us note that each of the previously described methods may employ visualization techniques in order to report results. For example, Figure 1.2 shows a bivariate scatterplot of product sales before clustering. Figure 1.3 shows the resulting clusters based on the k-means technique.

Visually, the clusters are made evident in Figure 1.3 by the use of color. Of course, a non graphical output can report the same results. However, a text report of the same results would span several pages. Imagine a similar result on a much larger dataset and the advantages of a graphical representation of the results is quite compelling.

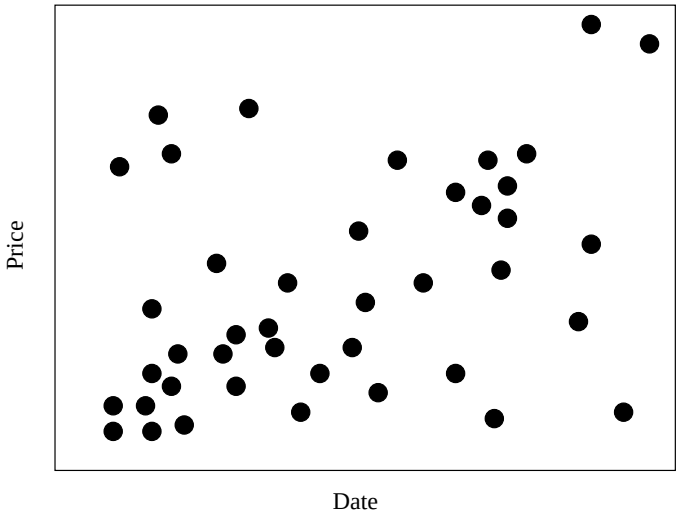


Figure 1.2: Scatterplot of product sales data.

Visualization research is sometimes confused with the study of graphics. Graphics researchers are interested studying and developing techniques associated with rendering pixels on computer displays. Visualization research employs graphics techniques. However, the primary task for visualization researchers is determining what information to display, whereas graphics research is primarily interested in efficiently displaying the information.

From a human computer interaction (HCI) viewpoint, visualizations are extremely interesting. In particular, when considering the usability of computer systems, HCI experts are interested in measuring the efficiency, effectiveness and satisfaction of users based on their interaction with the system being studied.[BM94] While the previous example demonstrated that visualizations can be effective and efficient, further study would be warranted to better understand user satisfaction. In addition, a bad

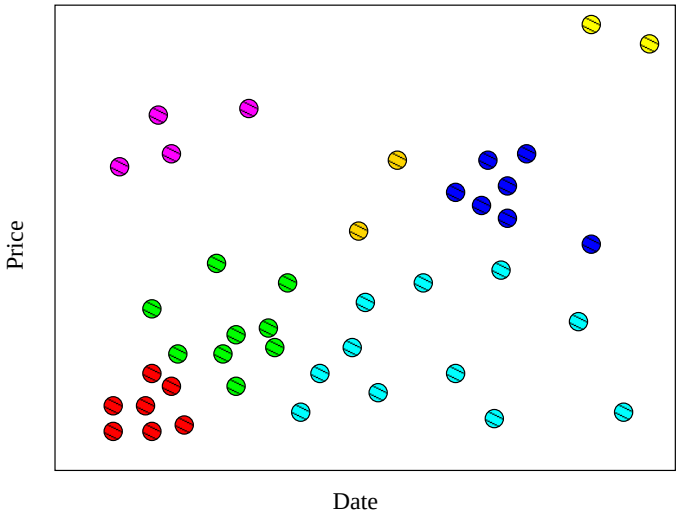


Figure 1.3: Scatterplot of product sales data with clusters.

visualization can be deceiving.

A relatively new focus of study within KDD is that of Visual Data Mining, which aims to provide graphical representations of the discovered patterns as a means to communicate results.[Ank00, Kei97, BKK97] The goal of such research varies from visualizing the output of data mining (either final or intermediate results) to more generic data visualization of the content of databases.

Many current approaches for visualizing the output of data mining activities are application specific. While this type of approach is useful from the knowledge discovery standpoint, what is necessary is the development and integration of generalized toolkits to handle a wider variety of problem domains.

1.1 Visualization and Databases

When considering the problem of visualizing data contained in databases there can be a variety of approaches. As shown in Figure 1.4, a database can be used to store data, such as fluid flow measurements. Access to the data is provided via queries, which extract the data from the database and feeds the data to the visualization system.

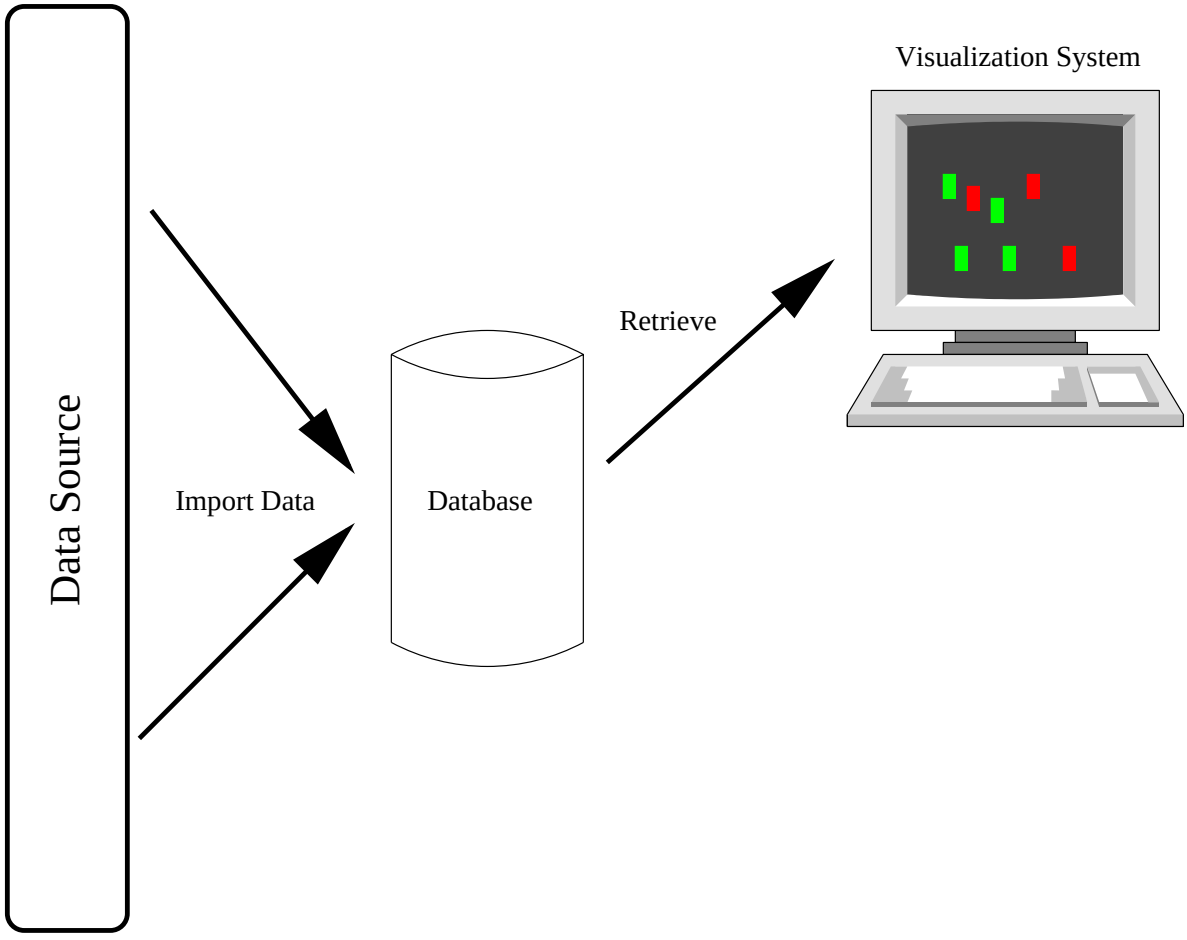


Figure 1.4: A database used as a repository of information for visualization

Another approach, shown in Figure 1.5, uses a visualization system to display the results of queries. The difference between this approach and the previous one is subtle. The first approach uses a database system to enhance a visualization system, whereas the second approach uses a visualization system to enhance the functionality of a database system by providing a more integrated approach to the problem.

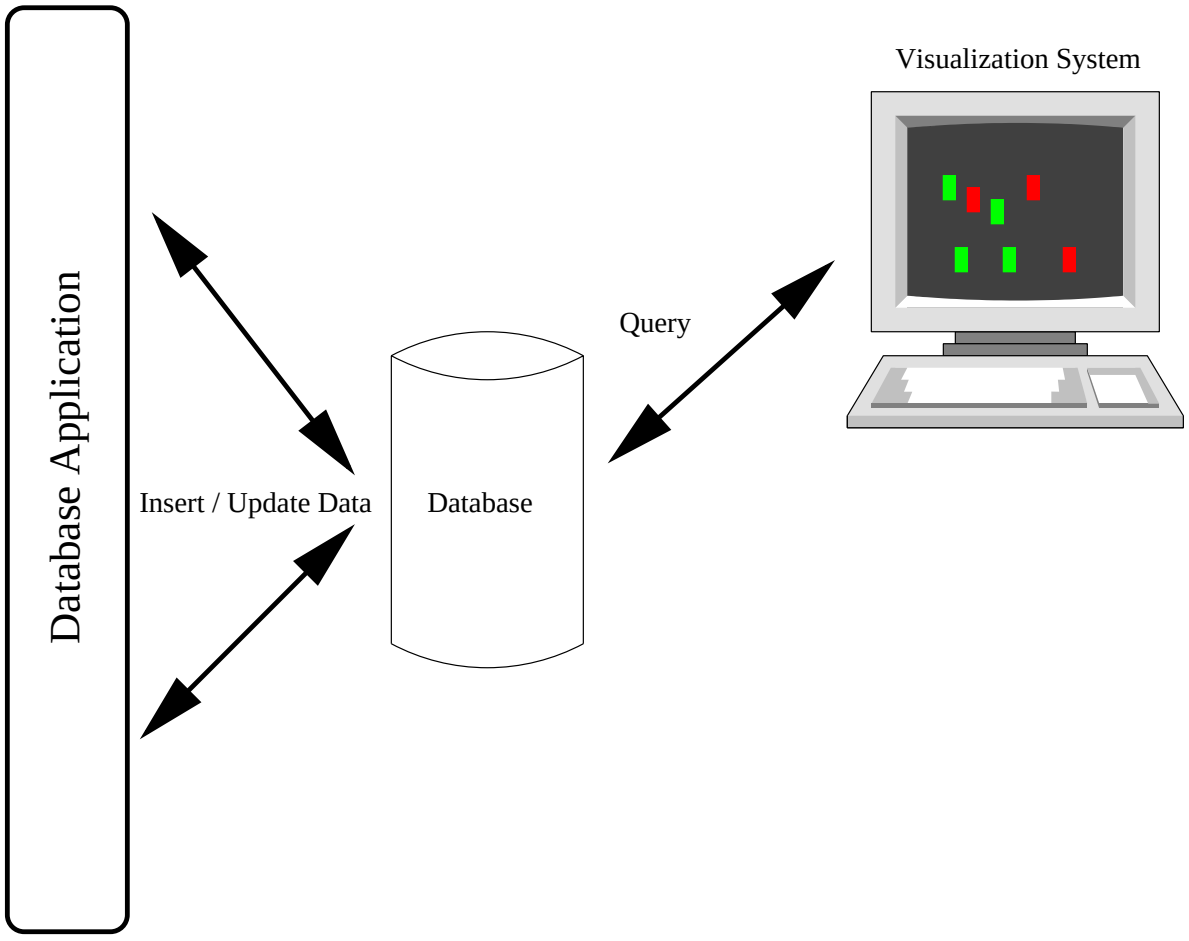


Figure 1.5: A visualization system used to display the results of database queries

In the context of database systems, access to the data is provided via queries. The alternative access method (when using flat files) is to hard-code the input into a

pre-specified format. Queries allow for flexible reorganization of the data.

Database systems offer a number of advantages over file structures. For instance, the ability to scale for extremely large databases is of great importance for KDD activities. Query languages are accessible to a wide range of users, with query by example (QBE) [Zlo77] tools as well as direct structured query language (SQL) interfaces.

Perhaps the most relevant contribution of database theory to the problem of data visualization is that of *independence*. By this, we mean the separation between the physical structure of the data and how the data is accessed. This separation allows for filtering and transformation of the data to occur according to a logical description, as opposed to a physical description of the transformation.

The Visualization Process

When it comes to analyzing large, complex datasets, visualization techniques are often employed. Indeed, it can be shown that visual representations are helpful for understanding relationships between data, identifying outliers, etc. Typically, we tend to view the graphic output of the visualization as the key component. As a consequent of this results-oriented view, we have lost sight of the context within which visualization occurs.

The process of visualizing data is at the core of our research. Figure 1.6 describes

a simple process model for visualization. While the goal of a visualization may be to gain insight into data, it is easy to gloss over the essential elements of this model. For instance, a results oriented view (shared by many users) will focus on the graphical output of the process.

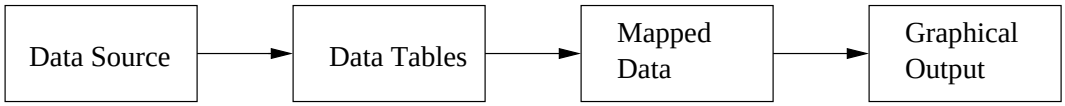


Figure 1.6: Process model for visualization

Of critical importance with this model is the transformation of data into a format that can be visualized. This step must be taken by any system involved with the visualization of data, regardless of the problem domain.

In this research, mapping is the process of transforming data from its original format to the format required for visualization. This transformation process is essential due to the varied data types and formats contained in databases.

Displaying information visually requires that each element in the graphical display be both ordered and scaled appropriately. By order, we mean the relative positioning of items in the display. It is important to distinguish between datatypes that have a natural order and those that do not. However, even data stored as a number in the database may still be categorical, and hence not ordered. In such cases the ordering may need to be defined according to the particular requirements of the problem domain. For example, most people are accustomed to an ordering scheme

for numeric and temporal values as shown in Figure 1.7. The chart shows the value of the stocks comprising the Dow Jones Industrial Average for approximately the past 85 years.

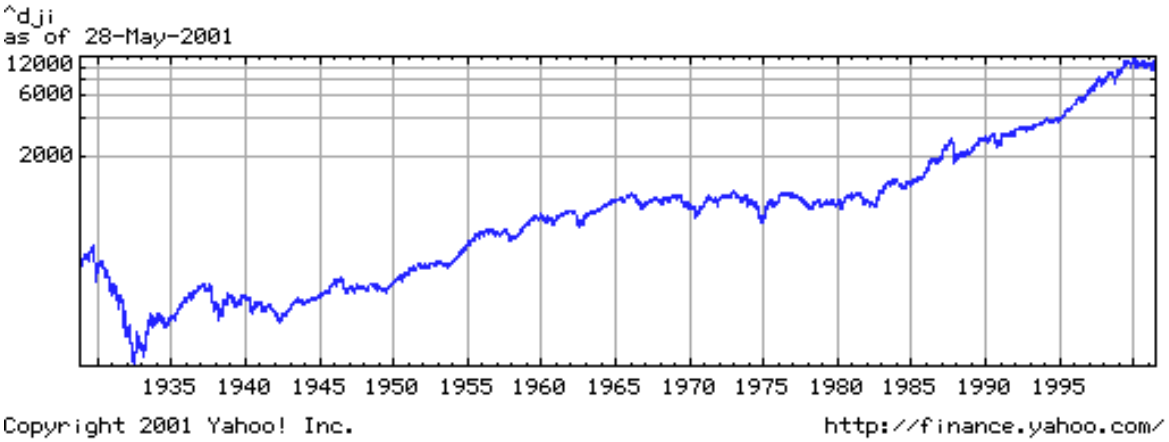


Figure 1.7: Ordering of numeric values

Methods for ordering of non-numeric data vary greatly, depending on the application. For text values, lexicographic order may be used. However, the individual constructing the graph may apply some form of expert knowledge in order to group items intelligently. Figure 1.8 shows a graph of family types from the Indiana census.

For scientific applications, the ordering of information on the display often coincides with spatial information. The visualization shown in Figure 1.9 depicts sonic shock waves from a simulated mach 1.9 jet engine exhaust with areas of high pressure (red) to low pressure (blue) illustrating the shock waves streaming outward.[web01]

In contrast with the order of values is the concept of scale. The scale of values can be represented spatially, with the relative distance between two items in the display

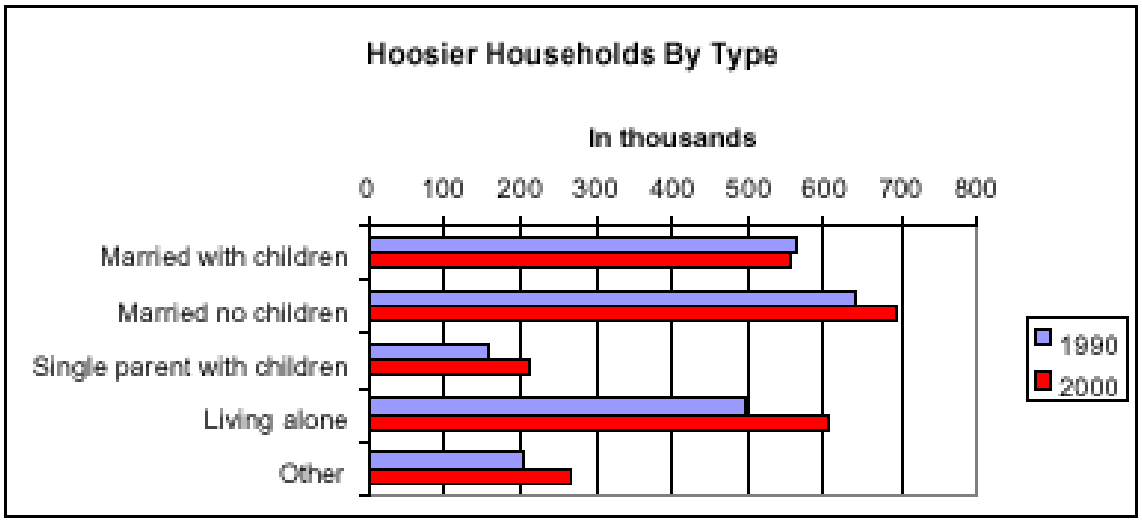


Figure 1.8: Ordering for non-numeric values

implying the differences in the underlying values. In addition to distance, the scale can be represented by color, size, or shape of graphic elements. If two items are far apart, this implies that the underlying values are far apart. For visualization purposes, color is frequently used to depict the differences in scale. Figure 1.10 shows the use of color to depict differences in expected temperatures for the United States. Note the use of colors closer to red depicting warmer temperatures and the bluer colors depicting cooler temperatures.

Missing Values

One reason for transforming the data is closely related to the KDD process. In particular, the data may not be "clean", or may have missing values. For instance,

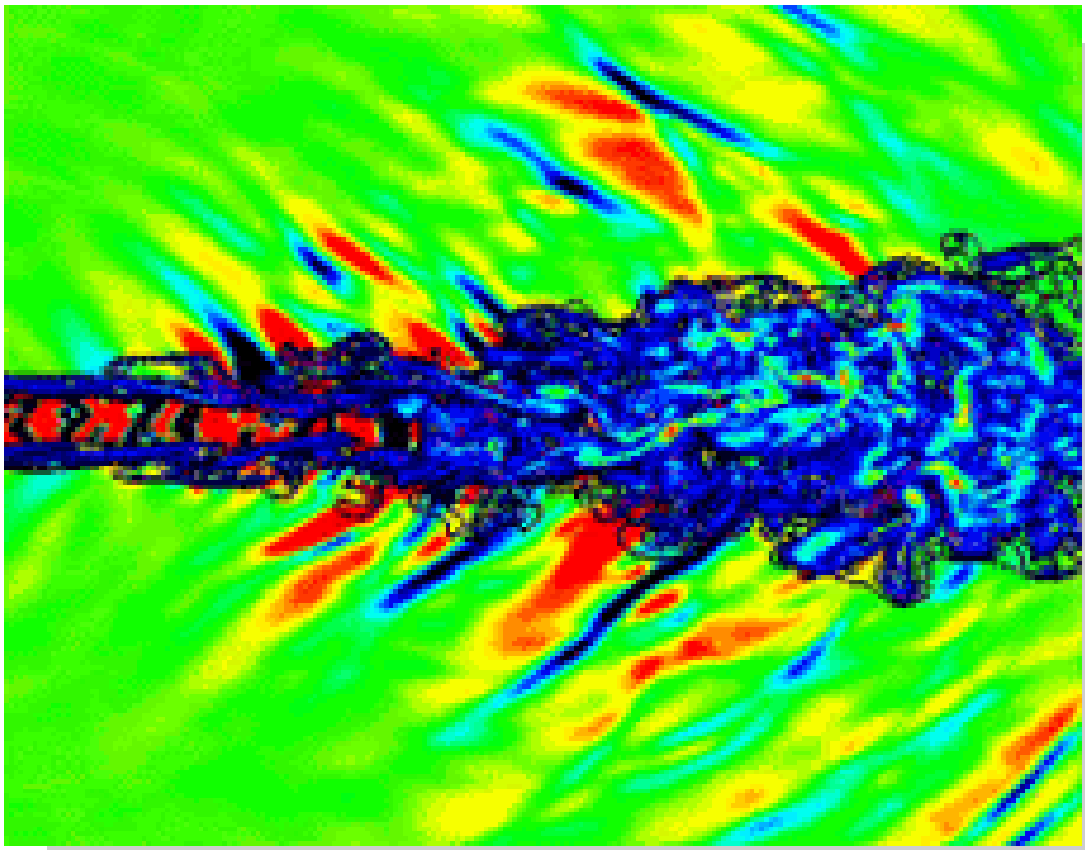


Figure 1.9: Visualization of sonic waves from a jet engine simulation

the latest ANSI standard defines 24 interpretations for null values. For visualization purposes, we could choose any of the known interpretations. The chosen interpretation may be dependent on the particular application or user. Obviously, in order to be applicable across a wide variety of problem domains, a visualization system must be flexible with respect to the definition or selection of data transformations.

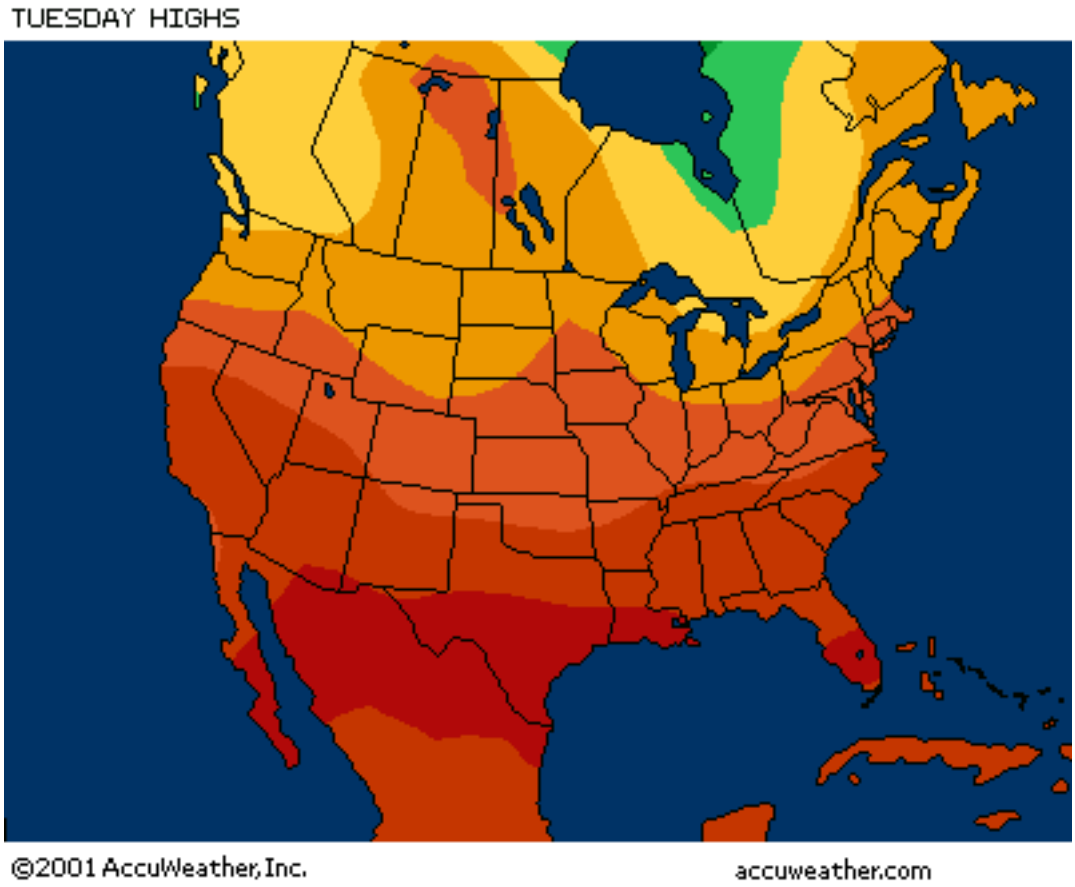


Figure 1.10: A weathermap showing expected temperatures by using colors

1.2 Thesis

The main contribution of this research is the formalization of a generalized mapping construct, aimed at supporting visualization of database information. By factoring out the mapping capability, we show that a wide variety of visualization techniques are supported within a generalized framework.

Figure 1.11 shows a generalized view of the visualization process. While this

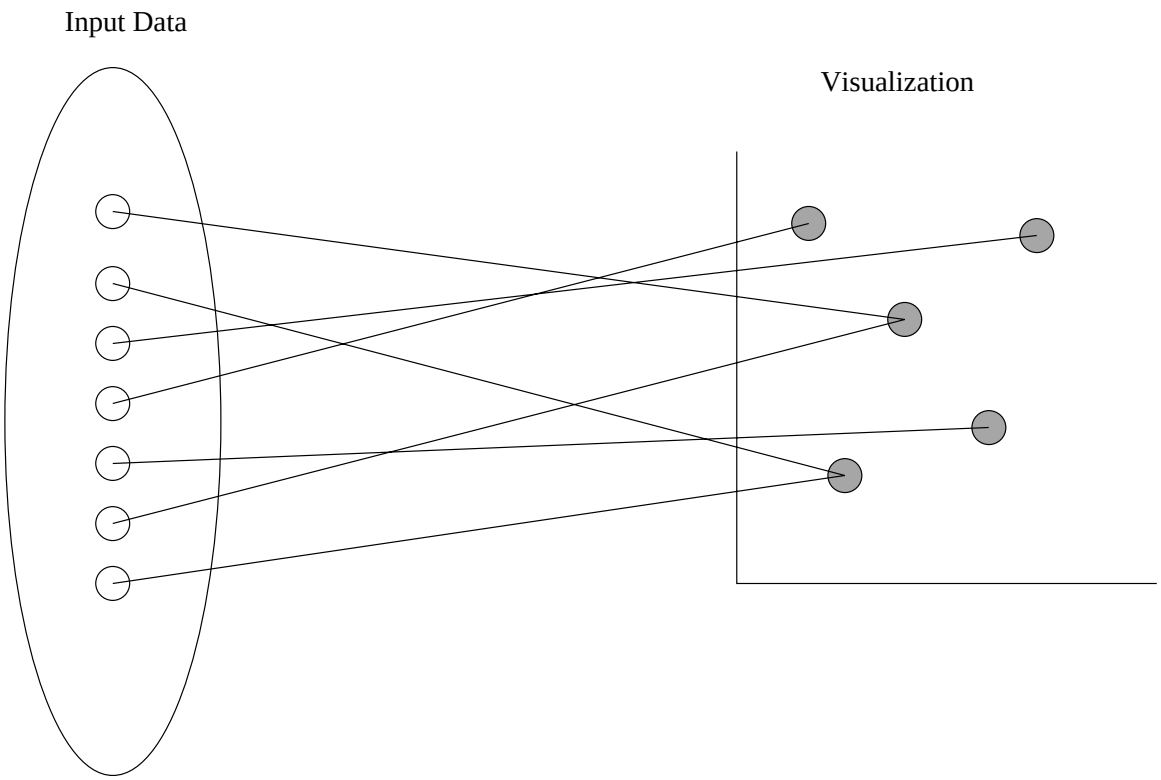


Figure 1.11: A generalized view of the mapping process

is a simple representation, the mapping of the input data to the output display is non-trivial. Note that one of the following situations are required.

1. The data is already in the necessary form required by the visualization system.

2. The visualization system contains the necessary logic to transform the input data.

Our approach, which abstracts the data transformation process is shown in Figure 1.12. Note that both of the previous situations can still occur. However, by creating a

mechanism for externalizing the transformation process, both the visualization system and the data are shielded from each other.

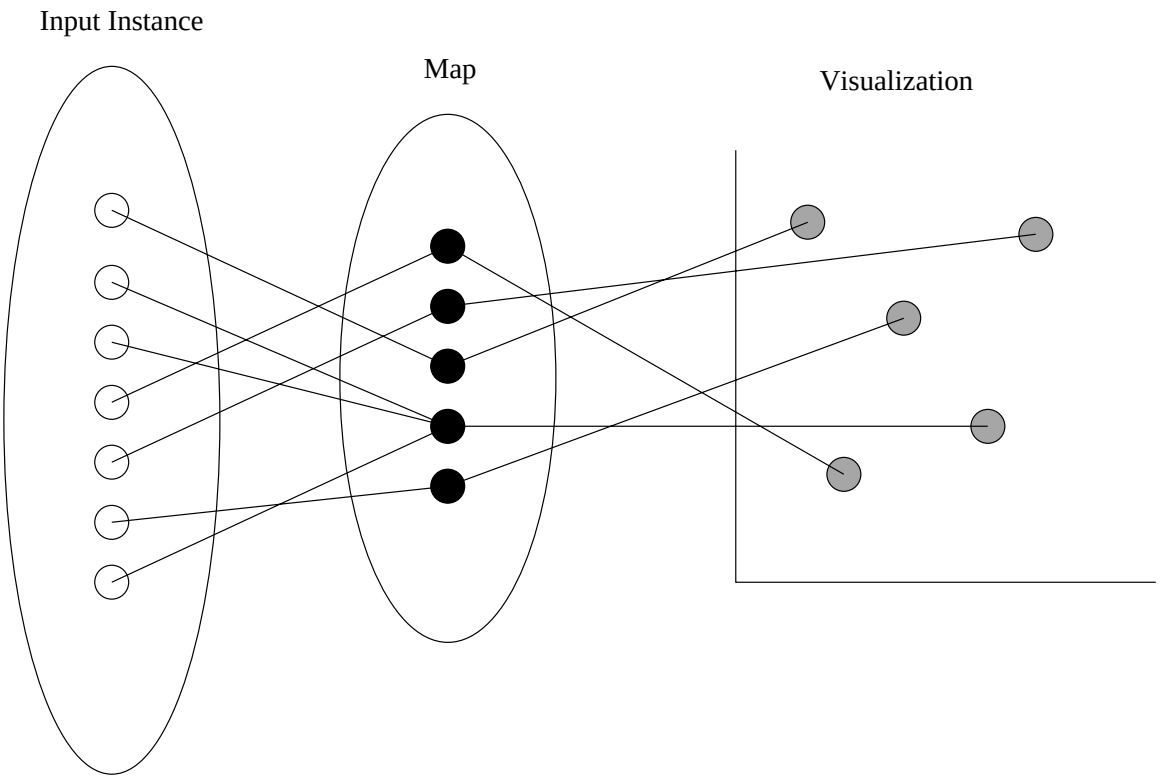


Figure 1.12: An abstraction of the mapping process

A survey of related database visualization research is discussed in Chapter 2. We focus on the systems and research associated with database systems and visualization, as opposed to a broad survey of visualization techniques. For a more comprehensive study of scientific visualization research we encourage a review of [NHM97]. The topic of information visualization is greatly covered in [CMS99].

Chapter 3 provides the basic definitions and notations we use in later chapters.

In particular, we focus on relational database terminology. In Chapter 4, we describe an architecture for visualizing database information. The goal for developing such an architecture is to provide sufficient details to an overall design for a system supporting visualization using generalized techniques.

The mapping concept is more fully defined in Chapter 5. The chapter describes our formalism for mapping and provides details surrounding the implementation based on the formalism.

A significant issue faced by all computer applications is that of understanding the impact of technology upon the users. This is particularly challenging for the field of visualization, since the user plays the key role of interpreting the results. Accordingly, Chapter 6 provides empirical results from an in-depth usability evaluation of the mapping process.

We provide an example application of our approach in Chapter 7, in which we explore techniques for visualizing the structure of information stored in databases. Lastly, we conclude with a discussion of open problems and future efforts in Chapter 8.

2

Related Work

In this chapter we present an overview of other research that is most closely related to our topic. We have separated the chapter into four main areas. First, we present a number of projects related to the problem of visualizing query results. Second, we review activities related to visualizing the results of data mining operations. Third, we provide details on systems oriented around providing database functionality to visual displays. Fourth, we provide details on visual query languages.

2.1 Visualizing Query Results

In this section, we provide an overview of research efforts focused on graphically displaying the results of a database query. Much of the focus is on maximizing the amount of information on the display. As a result, users are able to identify points of

interest in the data.

Starfields and Dynamic Queries

One approach to integrating visualization systems with databases is demonstrated by dynamic queries.[WS92, AS92, AS94b, Shn94, AW95, Ahl96] Figure 2.1 shows a screen capture from the Homefinder system, in which homes are displayed as points on the display. The user interface for this system has two principle components: on the right, the user specifies input parameters to a fixed form query, while results are shown on the left.

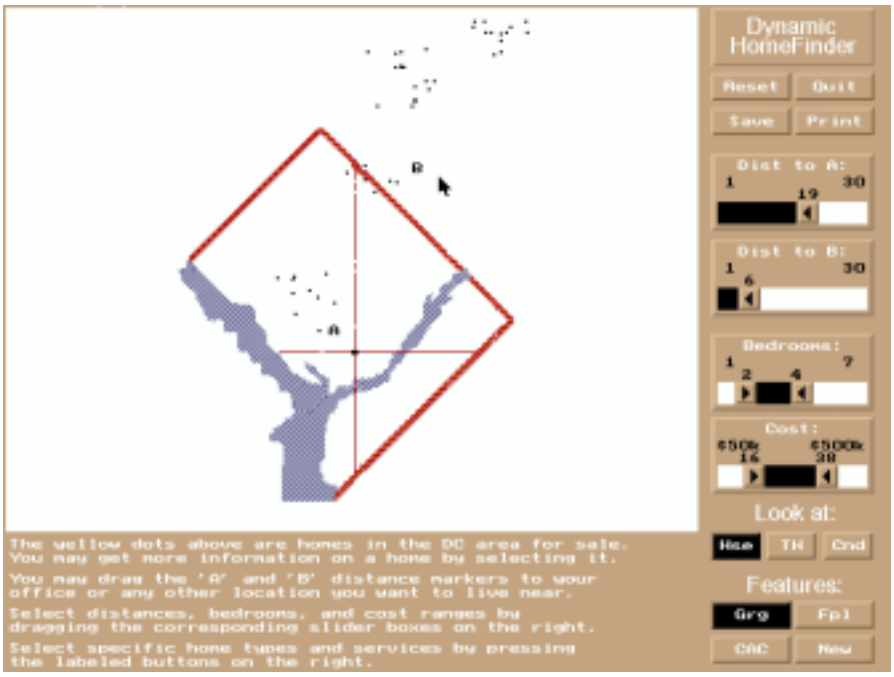


Figure 2.1: Homefinder starfields display

The mapping structure for this particular application is geographic. That is, the

points representing homes for sale are overlayed on a geographic map. The user interacts with the results by manipulating the sliders. This manipulation is the basis for the dynamic queries capability. Subsequent work, such as [BSP+93] adds to the interaction by combining movable filters to the display.

VisDB

A significant amount of research in database related research was contributed by Daniel Keim.[KAK94, KKS94, KK95a, KK95c, Kei95, KK95b, Kei96a, Kei96b, KK96] The VisDB system displays each element of the query results as a pixel in the display. Figure 2.2 shows a sample screen capture from the VisDB system.

The mapping of data elements to screen location and color is based on each data values distance from the query. Tuples that most closely match the query are closer to the center of the display, with the least closest matches further from the center. Using this technique, users are able to evaluate similarity between multiple datasets.

Users interact with the visualization in order to discover relationships and patterns that would not be visible via other mechanisms. This approach is consistent among other systems, which we will cover in subsequent sections.

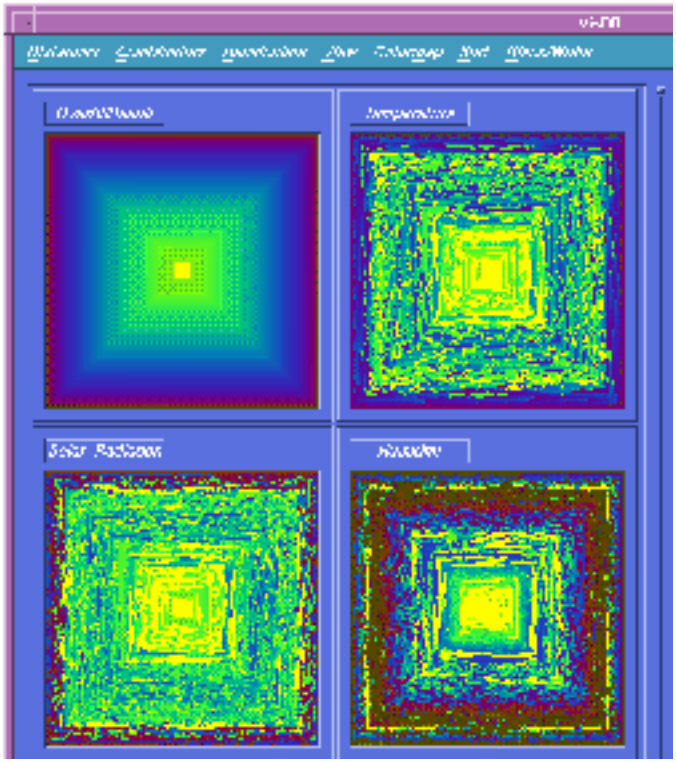


Figure 2.2: Query results from VisDB

2.2 Visualizing Data Mining Results

In this section we describe research activities involved with visualizing the results of data mining. For a comprehensive overview of the area, we strongly advise [UFW01] as a starting point.

Discovering association rules from market basket data continues to be an extremely active research area.[AS94a, Bay98, STA98, BMUT97, SON95, HPY00] From a visualization standpoint, there have been a number of attempts at graphically depicting the discovered rules.[FMMT96, WWT99, HHW00]

Visualization techniques are frequently employed by cluster detection applications.[CC92, MR97, ALS97, Ank00] Figure 2.3 shows a parallel coordinates display of flower data. The clusters are revealed by the use of color in this display, which shows three clusters (Blue, Pink and Green).

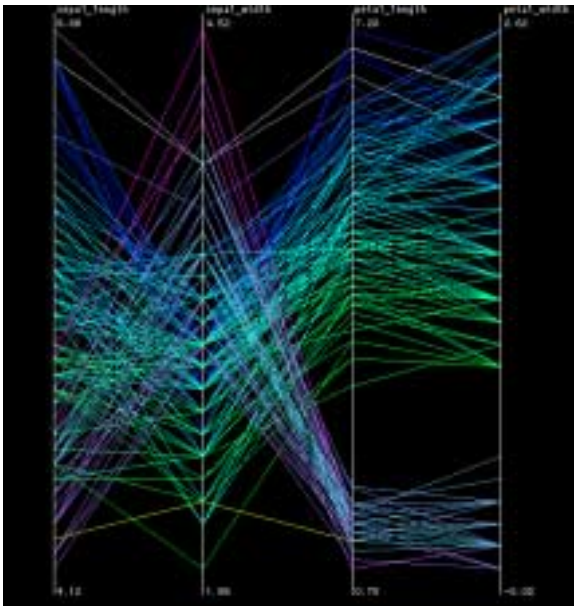


Figure 2.3: Parallel Coordinates

Scatterplot techniques can also be used to identify clusters, as shown in Figure 2.4.[Kea99]

2.3 Integrating Database Functionality

A different, but related problem for visualization research is described in this section. Namely, how can database functionality be integrated with a visualization

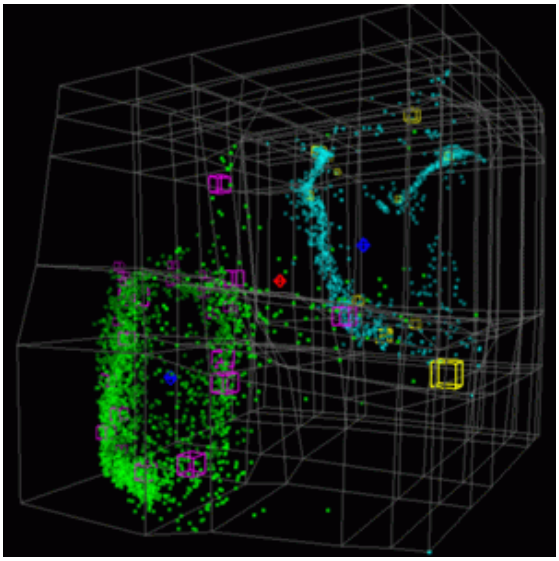


Figure 2.4: 3 dimensional scatterplot used for cluster detection

system.

Tioga

The Tioga system's original focus was to enhance scientific visualization applications with database support.[SCN+93b, SCN+93a, ACSW96, WS97]

The system has evolved from its original objective to be a visual browser of relational data. Figure 2.5 shows a screen capture from the Tioga-2 system, in which average commute time for U.S. workers is visualized.

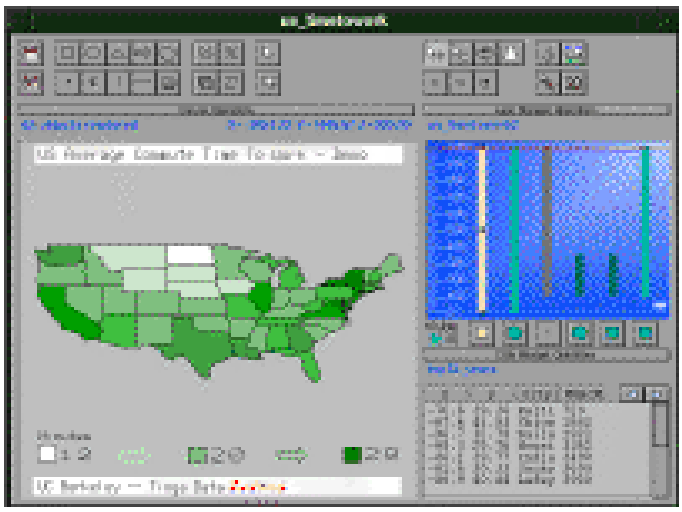


Figure 2.5: The Tioga-2 system, showing average commute time

ExBase

The ExBase sytem and its predecessor ExVis [GP88, GPW89, GSB91, LG94, LGa, LGb, Lee] illustrate another database visualization approach. In this system, the database concept of a view is extended to encompass a visualization view. The end result is a layered approach, which brings the user and the data closer together. The ExBase system proposes an architecture that supports the definition of mappings from the data to the visual display. This support, called *representation mappings*, maps fields from the input query to visualization parameters. Again, the focus is on output, and not on dealing with issues related to early stage transformations of the data.

DEVise

The DEVise system [LRM96, LRB+97, WL00, HYL00] allows users to integrate information from diverse data sources into visualizations. The DEVise system places emphasis on querying and visualization primitives. In particular, the system provides a record level mapping from the data to spatial coordinates, which provides the ability to control the color and shape of the data elements. Figure 2.6 shows a visualization of bank transactions using the DEVise system. This application is attempting to identify owners of multiple accounts with suspicious transaction activity.

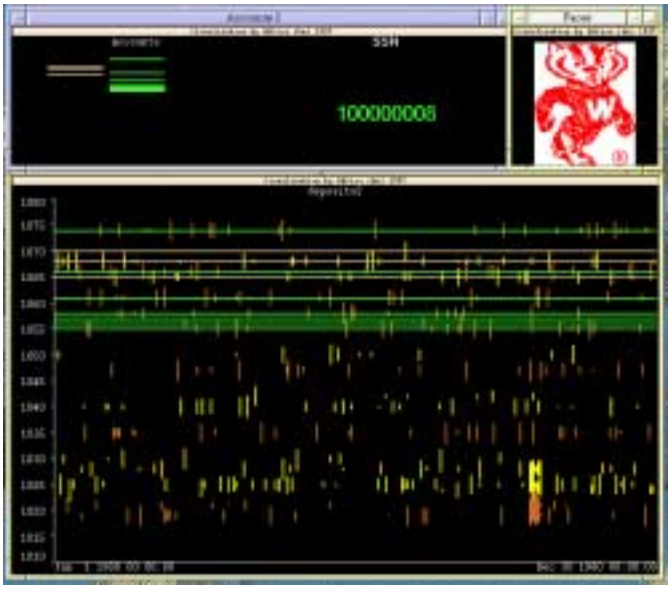


Figure 2.6: Visualization of bank transactions in DEVise

2.4 Visual Query Languages

A separate, but related, research topic is the development of techniques for graphically specifying queries. In some cases, certain approaches couple the diagrammatic query specification with visualization of the output. A fairly recent survey is provided in [CCLB97]. Other examples of visual query languages are provided in [PdBA+92, PK94].

Query By Example

The earliest example of a visual query language is Query By Example (QBE).[Zlo77] The main idea behind QBE is to provide an interface to the user based on the tabular representation of the the database tables.

Employee	SSN	LastName	FirstName	Birthdate	···
p.				>'8−1−61'	

Figure 2.7: Query By Example

The user specifies the constraints in the tabular display in order to define the query. Figure 2.7 shows an example QBE query of employees born after 8-1-61. Note that the user can define the output of the query by using the "p." command in the desired columns.

DeLauney

The DeLauney system, described in [CAL+97, Cru93a] and the description of the visual query language DOODLE [Cru93b, Cru92, CL00], is an example of an approach that combines graphical output of query results with a graphical method for specifying the queries.

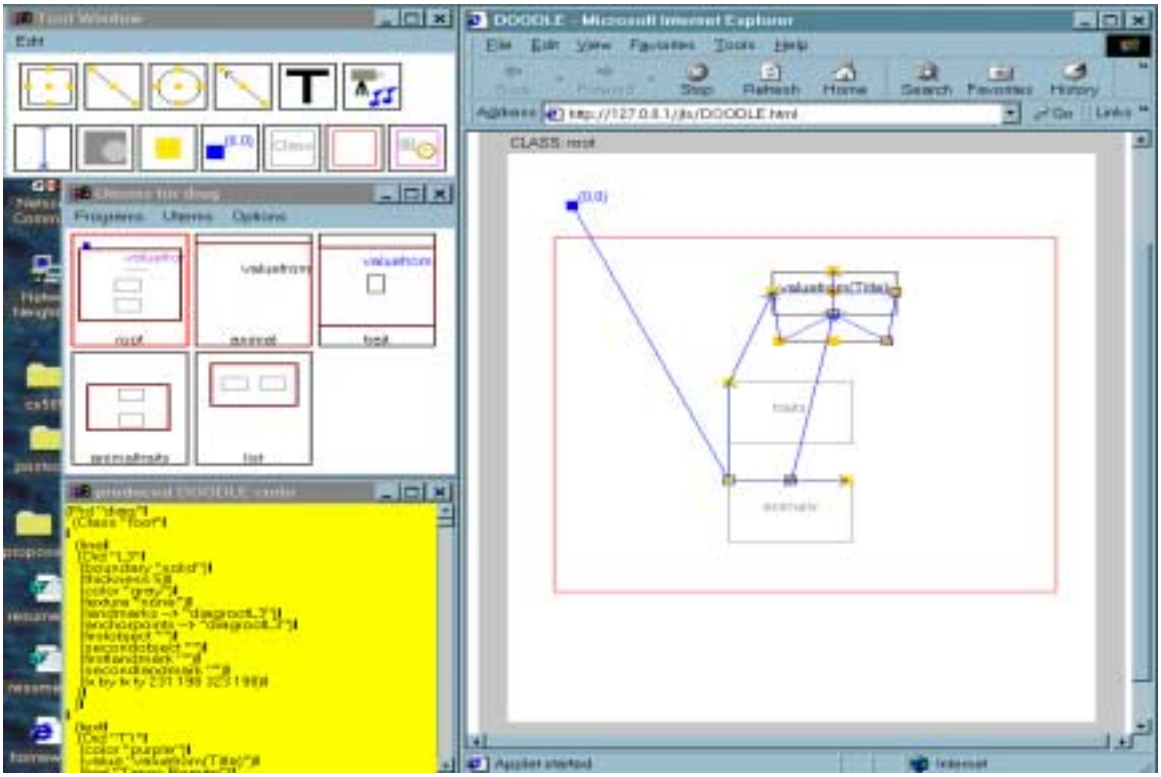


Figure 2.8: Screen capture from the Delauney system, showing the creation of a visualization specification using DOODLE.

Figure 2.8 shows an example of a visual query in this system. The user specifies the output declaratively using a toolkit of available constructs. What is particularly important with this work is the concept of allowing the user to construct the mapping

(in DOODLE) necessary for their application.

Query By Diagram

The Query By Diagram (QBD) system [ACS90b, ACS90a] utilizes ER models [Che75] as the representation of the data. Users then specify the query by interacting with the diagram, as shown in Figure 2.9. The developers of the QBD system performed usability experiments [CS95], in which they learned that users preferred visual queries over textual queries. A more recent result by the same group is provided in [Cat00].

2.5 Commercial Products

Visualization systems that incorporate ties to database systems have been implemented in commercial systems. In particular, the Data Explorer by IBM [IBM] incorporates a visual programming interface with access to database information via SQL queries. Figure 2.10 shows a screen capture from this system. Many products support presentation graphics generation, including Microsoft's [mic] Excel Spreadsheet and Access Database. A system by Quadbase [qua] incorporates SQL queries as a means of extracting data and formatting as a chart or graph.

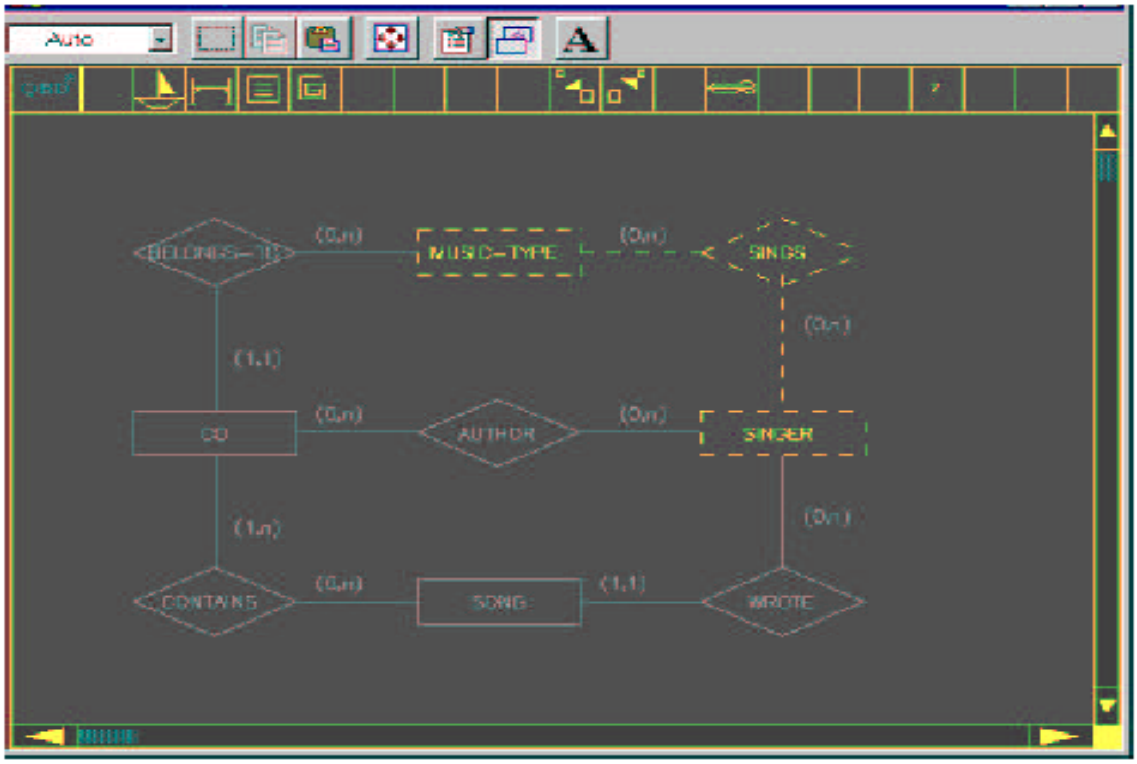


Figure 2.9: Screen capture from the QBD system – the query is over an ER representation of the data.

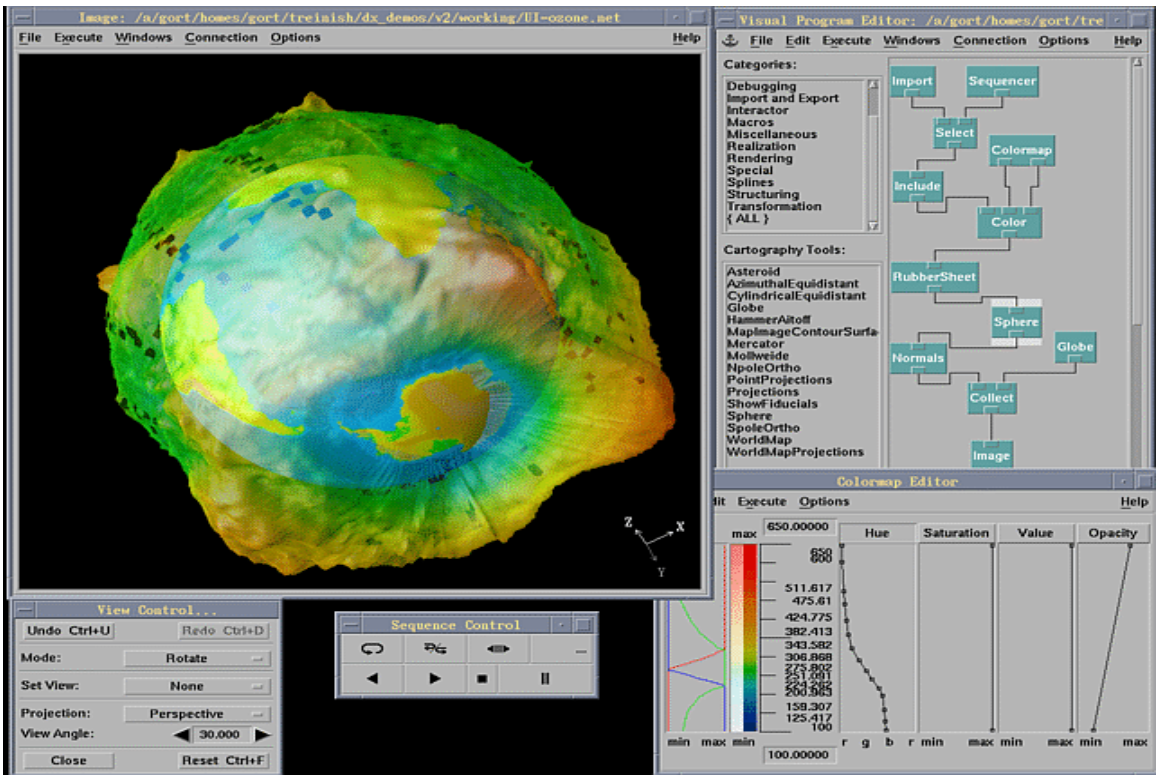


Figure 2.10: Screen capture from the IBM Data Explorer

3

Definitions

This chapter provides the formal, fundamental aspects of the relational model, which is the foundation of this research. The relational model was introduced by Codd in [Cod70]. Beyond the relational model, we provide an overview of query languages. In particular, we focus on the logic based language - Datalog.

3.1 The Relational Model

Let R be a relation schema and $\mathbf{r}$ be an instance of R. Associated with R is a finite set of attributes $\{A_1, ..., A_n\}$. For each attribute A_i we associate a set of possible values $\mathbf{dom}(A_i)$, called the domain of A_i. When considering an instance $\mathbf{r}$ we define $adom(A_i)$, the active domain of A_i, to be the set of values existing in $\mathbf{r}$. Note that, while $adom(A_i)$ is finite, $\mathbf{dom}(A_i)$ may be infinite. A tuple t of $\mathbf{r}$ is a

function $\mathbf{dom}(A_i) \rightarrow adom(A_i)$. Define $\mathbf{dom}(R) = \mathbf{dom}(A_1) \cup, \ldots, \mathbf{dom}(A_n)$.

We sometimes refer to values with the notation $t.A$, meaning the value of the A attribute in tuple t. For sets of attributes, $t.X$ means the values of each attribute of X in tuple t. For a relation R, the instance $\mathbf{r}$ is a finite set of tuples.

When describing a set of attributes we will utilize uppercase letters, such as X, Y, or Z. Distinguished from the attribute names are the attribute values, for which we will utilize lowercase letters, such as a, b, or c. Functions performed on attribute values will be specified in terms of the name of the function (i.e. ABS, $CEIL$, ...) and the attribute names. Generic functions will be specified by $f(X)$. Mathematical functions, such as addition and subtraction, will be specified by standard math notation.

A query q is a mapping between relation instances. In the next section we will describe the syntactic construction of queries. In the context of the relational model, the results of the application of a query mapping to an input instance is a relation. This concept is referred to as the *closure* property.

3.2 Query Languages

In this section we provide the basic formalisms for the specification of queries. For a more in depth coverage of the theoretical aspects of relational query languages see

[AHV95]. Of particular note with these languages is their equivalence. This equivalence allows us to blur the distinction between procedural and declarative languages. A procedural language is defined in terms of atomic operations on data, whereas a declarative language is defined in terms of a logical expression.

Relational Algebra

The relational algebra is a procedural language, with queries defined by composition of the following, basic operations. Again, R is a relation schema.

Selection: This operator, written $\sigma_\theta(R)$, selects tuples from its input that satisfies the boolean condition θ. The θ expression is typically of the form $\alpha\phi\beta$, where $\alpha, \beta \in R \cup \mathbf{dom}(R)$ and ϕ is a binary relation. For instance $\mathbf{r}$, $\sigma_\theta(\mathbf{r}) = \{t | t \in \mathbf{r} \wedge \theta\}$.

Projection: This operator, written $\pi_X(R)$, projects attributes of R. For instance $\mathbf{r}$, $\pi_X(\mathbf{r}) = \{t.X | t \in \mathbf{r} \wedge X \subseteq R\}$.

Cartesian Product: This operator, written $R \times S$, combines relations. For example, if $R = \{A, B, C\}$ and $R = \{D, E\}$, $R \times S = \{A, B, C, D, E\}$. For instances $\mathbf{r}$ and $\mathbf{s}$, $\mathbf{r} \times \mathbf{s} = \{t | \langle t.A, t.B, t.C\rangle \in \mathbf{r} \wedge \langle t.D, t.E\rangle \in \mathbf{s}\}$.

In addition to these operators, and since relations are considered as sets, the algebra allows for Union, Intersection and Difference operators, written as $\cup, \cap$, and $-$.

Using the relational algebra, a query q is simply an expression composed from the basic operators. The application of each of these operators (to the proper number of relation instances) produces another relation. Thus composition of operators is automatic. As a result, the relational algebra is considered a *Procedural* language, in that the query may be implemented directly from the operations.

Sometimes included as an operator in the algebra is the Join operator, which allows for more succinct expressions. The join operator is written as $R \times_\theta S$, where θ is a boolean expression, as used in the selection operator. The join, $R \times_\theta S = \sigma_\theta(R \times S)$. A special type of join, called the natural join is written by $R \bowtie S$, in which the resultant tuples agree on the attributes common to R and S.

A property of certain queries is that of monotonicity, defined as:

Definition 1 *A query q is monotonic if for any relation R, $R \subset R' \implies q(R) \subset q(R')$*

Rule-Based Queries

An alternative method for specifying queries is by using a logic-based approach. In this approach we distinguish between the existing relations and new relations that are derivable from the existing relations. The existing relations are commonly referred to as the *extensional* database (edb), while the new relations are called the *intensional*

database (idb).

To make this notion more precise, we will formally define the notion of a rule-based query. Let $\mathbf{R}$ be a database schema and $\{R_1, \ldots R_k\}$ be the extensional relations defined in $\mathbf{R}$. A rule-based query is an expression of the form:

$$Ans(u) \leftarrow R_1(u_1), \ldots R_n(u_n)$$

where Ans is an idb relation not in $\mathbf{R}$ and u is a set of literals, such that each literal in u is a constant (i.e. $a \in \mathbf{dom}(\mathbf{R})$) or an variable name occurring in one of the edb relations. Each u_i contains either variable names, or constants.

An alternative formulation for a rule based query is:

$$Ans(u) \leftarrow R_1(u_1), \ldots R_n(u_n) \wedge \psi$$

where ψ is a logical expression involving attribute names from the edb relations, or constants. Under this formulation, ψ is identical to the θ expression contained in the RA selection operator. The left hand side of the expression is called the head of the rule, while the right hand side is called the body.

A rule-based query has a straightforward semantics, which we describe here informally. Essentially, if there exists a valuation of the variables in each u_i in the body of the rule, then $Ans(u)$ can be inferred to be true. Note that this formulation utilizes the closed world assumption. In addition, rule-based queries can be shown to express only monotonic queries, since negation is not allowed in either the head or the body

of the rule.

Datalog

The rule-based queries represent a theoretically defined language that is equivalent to the relational algebra restricted to the π, σ and $\times$ operations. In general, this results in a language that is too weak to solve many problems. Extensions to the rule-based language provide more power, and is commonly referred to as the language Datalog.

A Datalog query is written in similar fashion as a rule-based query. It is common, however, for a Datalog query to be represented by a collection of rules. The rules have an interesting evaluation, in that each rule fires simultaneously. A Datalog program completes its execution when a fixpoint is reached. In other words, a Datalog query continuously executes until no new information is generated in an idb relation contained in the head of a rule. In addition, some classes of Datalog allow for idb relations to occur in the body of a rule, providing some limited form of recursion.

There are a number of Datalog language classes that are interesting to consider:

Datalog Equivalent to the rule-based conjunctive queries.

Datalog$^\neg$ Allows for negation in the body of a rule. Negation in the head of the rule is allowed and is considered safe.

While Allows recursive queries.

While^{+} Incorporates integer arithmetic in support of looping.

While^{new} Provides the ability to create new values.

Each variation of Datalog is strictly more powerful than the previous. In practice, production database systems restrict query expressibility to be less powerful than the While language. Even so, the more powerful languages can be programmed in a higher level language.

In this dissertation we propose a variation of Datalog as a user query language for performing transformation of data necessary to enable effective visualizations. We chose the Datalog language as the basis for our approach due to its succinct syntax, which essentially allows for the incorporation of a case statement in a relation query language.

SQL

In practice, users interact with relational database systems through an implementation of the Structured Query Language (SQL) [Int89, Int92, Int99]. The basic syntax of an SQL query is:

```
Select  <select field list>
```

```
From    <table list>

Where   <conditional expression>
```

In terms of the types of problems that can be solved, SQL does have its limits. In particular, recursive queries cannot be directly stated.[1] One of the benefits of SQL is its efficiency, with each query executable in LOGSPACE. Its declarative form relieves the user from worrying how the query will actually execute. In fact, the role of the query optimizer is to rewrite queries into a form that is most efficient.

For more complex operations, SQL can be embedded in a high-level language, such as C++ or Java, to provide finer levels of control over the processing of data. In addition, many of the vendors provide a procedural language to embed more functionality within the confines of the database in the form of stored procedures and functions.

While SQL predominates as the de facto query language, there have been several human factors experiments designed to identify relative strengths and weaknesses of SQL.[YS93, WS81, Cat00] In some cases, SQL has been shown to be inferior to other, proposed approaches, and yet, SQL has not been replaced. In Chapter 6, we report on a usability comparison of SQL to our proposed rule-based language. In the usability experiment, we measured the effectiveness of the language by having users write queries that performed classification tasks.

[1] Oracle has a limited form of recursion with its CONNECT BY clause.

4

Architecture

The primary contribution of this dissertation is presented in this chapter. The key concept is that of an architecture that integrates visualization systems with database technologies. The goal is to leverage the strengths of each technology in order to arrive at more powerful solutions that positively impact user's capabilities.

4.1 The Architecture

Figure 4.1 depicts the overall architecture that we are presenting. While there are several key elements, in this chapter we will focus primarily on the portions of the architecture that are involved in presenting and interacting with the data in a visual form. 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.

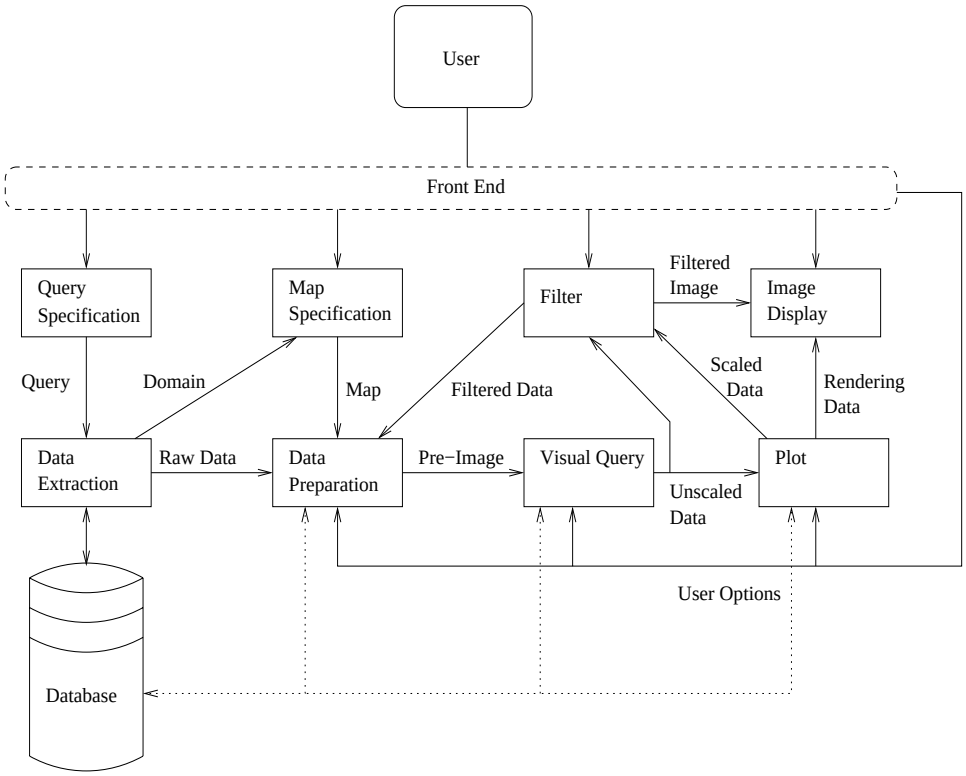


Figure 4.1: The architecture supporting visualization of database information.

Note that the architecture encompasses the complete spectrum of activities that are envisioned by the KDD process that was described in Chapter 1. In order to help present a clear description of the architecture we will utilize screen captures from the proof of concept system that we have developed as part of this research.

While the architecture pictured in Figure 4.1 describes the individual components, it is important to note that the data preparation process supports a feedback within itself. Figure 4.2 shows a detailed view of the Data Preparation module. The dataflow

indicates the nature of the feedback. For example, the output of the map join process can itself be joined with a different map. The user can define the sequence of operations that comprise their application.

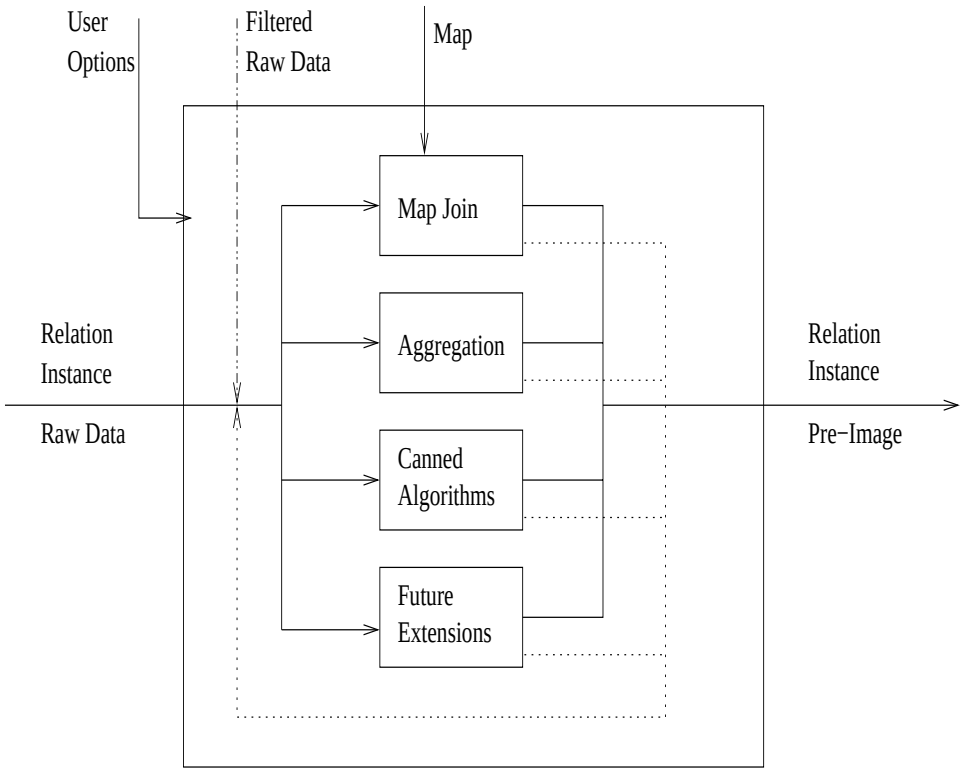


Figure 4.2: A closer view of the data preparation module. Note that the architecture supports arbitrary, user-defined sequences of transformations.

Dataflow within the Architecture

Each module of the architecture has specific requirements for its input and output. In this subsection we define the details of the dataflow between the architecture's

modules.

The Query Specification module provides for the creation of user-defined SQL queries. For this module the output is an SQL query, in the form of a string. The implementation provides the user with the ability to save the query specifications. Figure 4.3 shows an expanded view of the module.

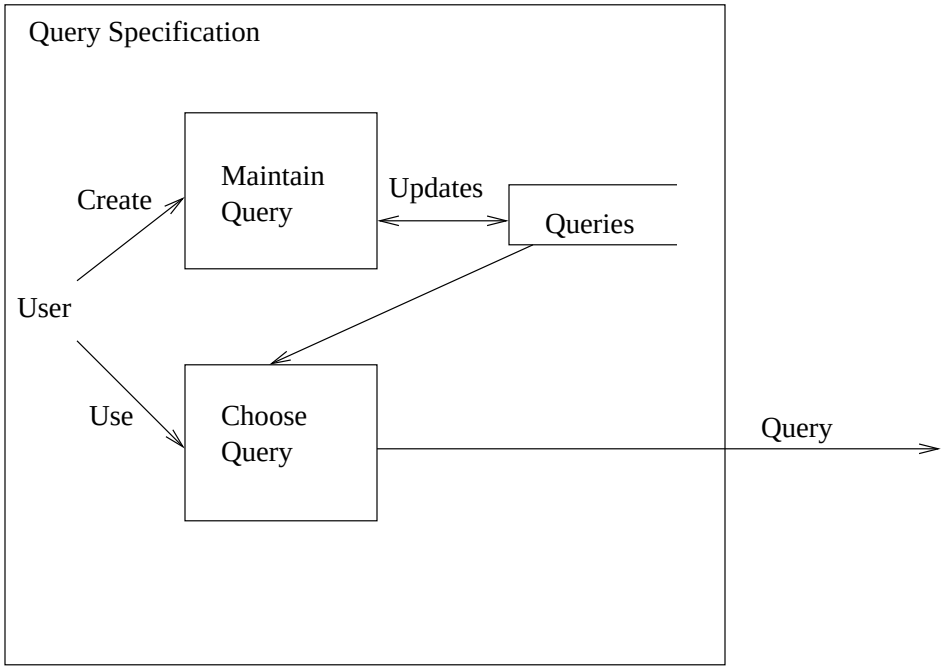


Figure 4.3: Detailed view of the Query Specification module.

The Database module supports all of the interactions between the other modules and the underlying database management system. The module takes as input an SQL query and returns the result of the query. No assumption is made as to the "correctness" of the input. Figure 4.4 shows a view of this module.

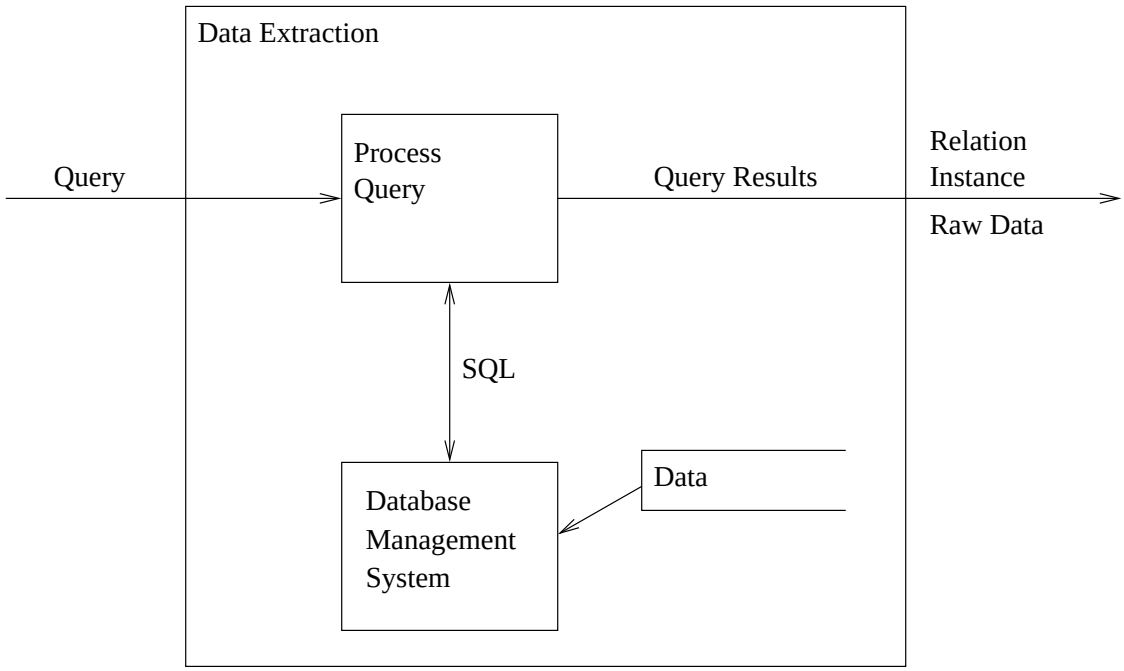


Figure 4.4: Detailed view of the Data Extraction module.

The Data Preparation module is the focal point of our architecture, within which a variety of interesting characteristics surface. From a dataflow standpoint, Figure 4.5 shows the details of this module. Note that regardless of the specific transformation being applied to the data the output remains a relation instance. This closed property provides the capability for the feedback process to occur within the module. Alternative approaches to data preparation may involve custom programming, which does not yield efficiencies when changes are required. The user may apply a *Map*, which adds order and scale to the input data. We defer details of Maps until Chapter 5. The output of the data preparation module is called the pre-image data, which

is managed by the architecture as a relation instance. Certain "canned" algorithms are provided in this module as well. For example, algorithms to extract frequency information from the input data are contained in the prototype implementation. In addition, future extensions to the toolkit are provided for, such as data mining algorithms.

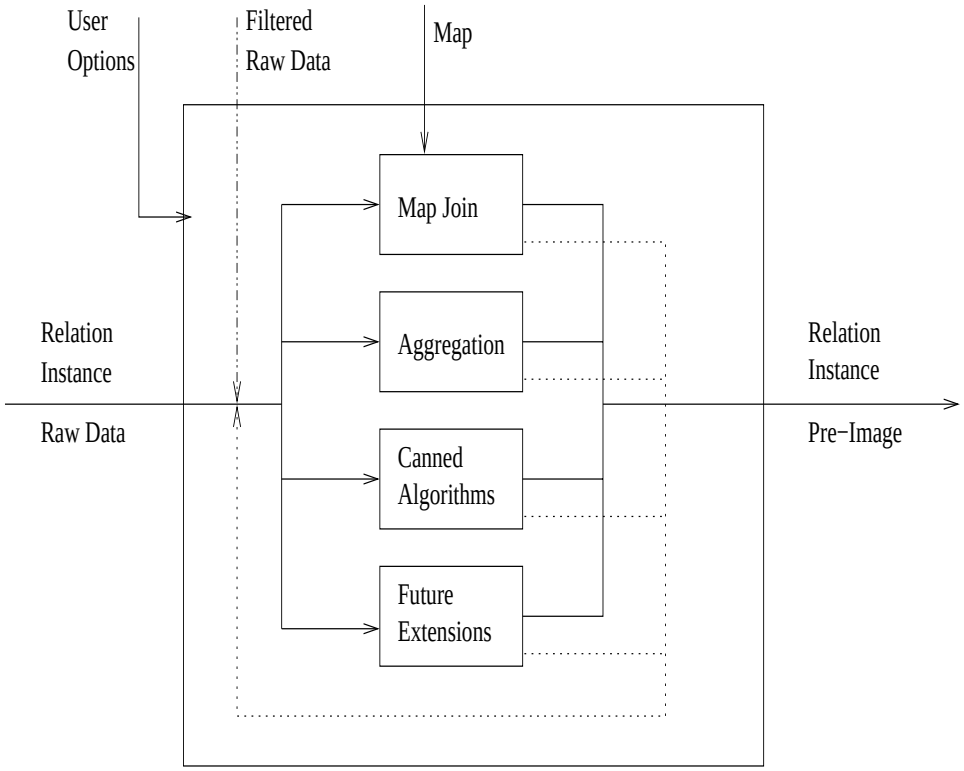


Figure 4.5: A detailed view of the data preparation module.

Note that the submodules of the data preparation module take as input relation instance(s) (in the form of query results) and return another relation instance. This design allows for the construction of arbitrary chains of data transformations, which

is supported by the front-end.

The Map Specification module provides for the definition and construction of maps. A map is a relation that can be joined to another relation, the effect of which is to transform input data according to the specific requirements of the users' application. In particular, a map transforms categorical data to numeric values, as required for plotting. This specification module supports a declarative language, which we provide further details of in Chapter 5.

The Visual Query module takes the arbitrary relation produced by Data Preparation and projects those columns suitable for a particular rendering. That is, any visual output format chosen by the user requires a certain type of input, typically two to four attributes, which are arranged by the Plot module into the appropriate format. Details of this capability are provided in Section 4.2.

The Plot module takes as input the pre-image data and performs visualization specific operations on the data. For example, certain normalizations may be performed on the data in order to create a [0,1] range prior to rendering. The output of the module is an array of objects to be visualized. The structure of each object depends on the visual output format that was selected by the user.

The Image Display module renders the graphic image. The prototype implementation of this module utilizes the Java3D toolkit to render and manipulate the output.

Each output style we support (scatterplot, histogram, surface, etc.) uses the normalized values to determine placement of the graphic elements and the original values for labeling and user drill-down features.

The Filter module supports the user's interaction with the visualization in three ways. First, the user can specify a restriction on the input data, which results in a subset of the data being visualized. Second, the user can drill down on portions of the image display to see the details of the data at certain points of interest. Third, the user can select regions of the display that can be given as input to the Data Preparation module. The Data Preparation module can combine the filtered data with other data to show comparisons across datasets.

In the remainder of this chapter we will provide the details of the architecture. The architecture suggest a natural linear order to the flow of data. Certain modules may pass the data through unchanged. In particular, the Data Preparation module will simply pass the input directly to the output from the module when no transformations are specified. In addition, the prototype system may collapse certain functionalities within the user interface so as to streamline the user's interaction.

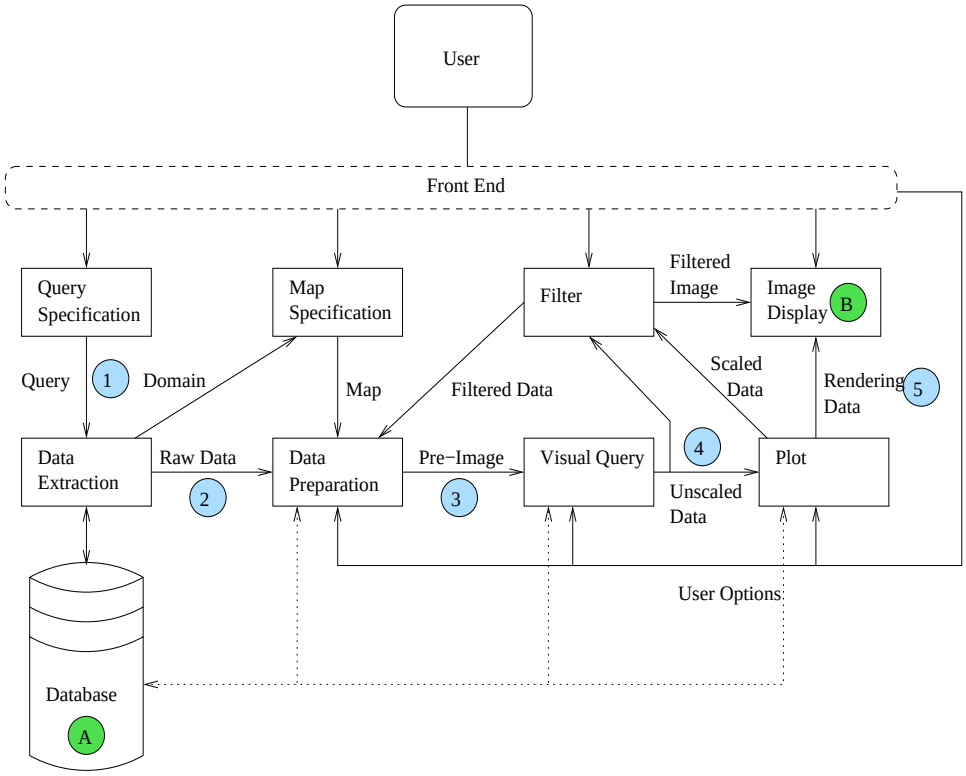


Figure 4.6: The architecture. Numbered points refer to specific dataflows.

4.2 From Data to Visualization

In this section we provide details of the flow between the modules of the architecture. We will use the structure of the architecture in conjunction with examples from the prototype implementation to illustrate the flows. First, note Figure 4.6, in which we have identified several points within the architecture. The lettered points refer to the data being visualized and the output generated by the architecture. The numbered points refer to specific dataflows that we will describe here.

Point A in the diagram refers to the data stored in the database. For the purposes

of exhibiting the features of the architecture we will use employee data, shown in Table 4.1.

Sex	Age	Salary	YearsOfService
M	47	47058	25
M	34	44396	9
M	49	46014	22
F	46	31477	23
F	42	44396	15
...	...	...	...

Table 4.1: A database of employee information

Typically, visualization systems focus on the mapping of the input data directly to an output form, represented by point B in Figure 4.6. Our architecture interposes several steps between points A and B. While the prototype system supports a variety of output formats, we will illustrate the architecture with a simple bi-variate scatterplot, comparing employee age to salary.

We begin with point 1 of the figure - the query. The object being passed from the Query Specification module and the Data Extraction module is a relational query. Underlying our prototype system is a relational database system that uses SQL as the language for query specification. The prototype system places no restriction on the query. Of course, invalid queries will not execute, and the flow ends in such cases. The following query retrieves the employee relation.

```
select * from employee
```

The Data Extraction module is straightforward, executing the query as is and returning the results in the form of a relation instance. Using the query given above, the returned instance is the entire table, shown in Table 4.1, and passes it to the Data Preparation Module - point 2 in the figure. This seemingly simple aspect of this module provides tremendous flexibility. The query string itself, serves as a boundary for other manipulations. For example, queries can be used as subqueries or joined to other queries in a seamless fashion.

The age and salary attributes of the employee relation are already in numeric form. For the bi-variate scatterplot display it is not necessary to perform any transformations on the input instance. Consequently, the instance flows through the Data Preparation module unchanged - point 3 in the figure.

Taking the pre-image relation instance and generating the visualization requires a linkage between the input relation instance and the output form. In this case, the linkage is visually depicted in Figure 4.7. Once the user has decided to produce a 2D scatterplot, the schema $\{X, Y\}$ is fixed. The Visual Query module performs this function, taking as input the relation instance and selections given by the user. Further details of the Visual Query module are given in Section 4.5.

The Visual Query module projects out the columns from the input instance. The prototype system performs this operation by augmenting the original query, since there were no transformations performed by the Data Preparation module. The

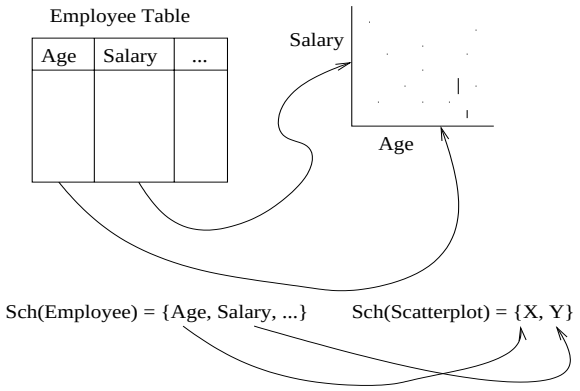


Figure 4.7: Linking age,salary from the input relation to X,Y.

query generated by the system is:

```
select vis.age, vis.salary
from   (select * from employee) as vis
```

The output of the Visual Query module is a relation instance, containing the values from the original input data is shown in Table 4.2. In Figure 4.7 this is located at point 4.

Age	Salary
47	47058
34	44396
49	46014
46	31477
42	44396
...	...

Table 4.2: Result given by the Visual Query module.

At this stage in the architecture, we scale the input data in the Plot module

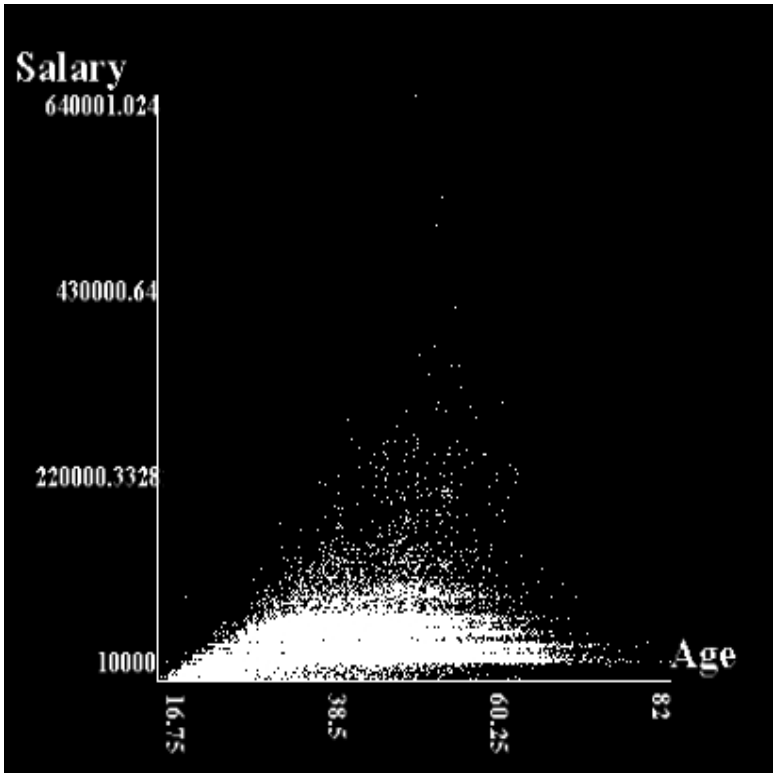


Figure 4.8: Visualizing employee age compared to salary.

and pass the scaled data to the Image Display module - point 5 in the figure. The scaling converts each data value to the range [0,1]. The scaled data is rendered by the Image Display module and the bi-variate scatterplot is displayed. Figure 4.8 shows the resulting visualization.

4.3 Data Preparation - Incorporating Aggregates

In the previous section, we illustrated the flow of data through the architecture. However, in that flow, the data did not require any modifications to prepare it for

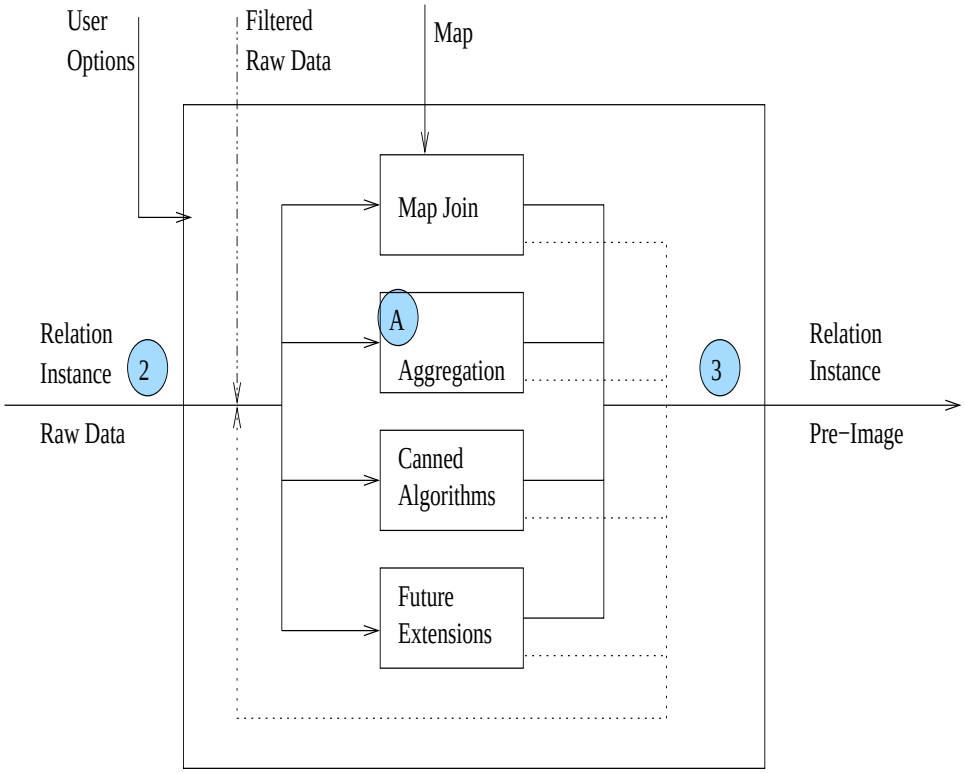


Figure 4.9: The Data Preparation module - Aggregation submodule.

visualization. The data flowed through the Data Preparation module unchanged. In this section we expand upon the Data Preparation module, illustrating the use of aggregate functions.

Figure 4.9 shows a detailed view of the Data Preparation module. Point 2 references the input to the module - a relation instance. Consider Table 4.1 as the source data for a visualization. Previously, we displayed the age and salary values as points. However, values duplicated in the data are not adequately represented in the display. Graphically, we can vary the color intensity to represent higher frequencies of values at a particular point.

The architecture supports counting, as well as other aggregate functions, in the Data Preparation module. In particular, the Aggregation sub-module, identified as point A in Figure 4.9, which takes the employee relation instance as input. Formally, we define an *Aggregation*:

Definition 2 *Given an input relation* $R = \{A_1, \ldots, A_n\}$*, an aggregation* M *is a pair* (R, L)*, where* R *is a relation and* L *is a set* $\{L_1, \ldots, L_p\}$*. The elements of* L *are defined to be* $L_i = A_j$*, or* $L_i = Agg(A_j)$*, where* $A_j \in R$ *and* Agg *is one of the standard SQL aggregate functions (e.g.* $count, sum$*, etc.)* □

Note that the first element of an Aggregation is an relation. In the context of this architecture, the Data Extraction module provides the relation R, allowing for aggregations to be defined in a very abstract manner, thereby freeing the user from the specific details of how an aggregation is performed. In particular, the form of the query easily incorporates the inclusion of database supported aggregate functions, such as COUNT, SUM, AVG, MIN and MAX.

This formalism is sufficient to support extraction of data for many visualization techniques. Constructing an SQL query given an aggregation is straightforward. Without loss of generality, assume L is ordered such that there is a k for which $i \leq k$ implies L_i is of the first form and $i > k$ implies L_i is of the second. The SQL query that is constructed is described by the following pseudo-query:

`Select` $L_1, \ldots, L_p$

`From` R

`Group By` $L_1, \ldots, L_k$

Importantly, an aggregation defines a relation with schema $\{L_1, \ldots, L_p\}$. Likewise, an instance of M is a relation instance $\mathbf{m}$. For example, to calculate the frequency of employees for each age, salary pair is given by the following query that is generated by the Aggregation module:

```
select   vis.age, vis.salary, count(*)
from     (select * from employee) as vis
group by vis.age, vis.salary
```

The output of the query is output from the Data Preparation module, identified as point 2 in Figure 4.9. The result from the previous query is shown in Figure 4.10, in which we have linked the frequency value to a simple coloring scheme. In this case, the color scheme denotes an ordering of the frequency: $red > yellow > green > blue$.

4.4 Data Preparation - Map Join

In the previous section we described how the Data Preparation module supports the incorporation of aggregate functions in a seamless fashion. In contrast, this section

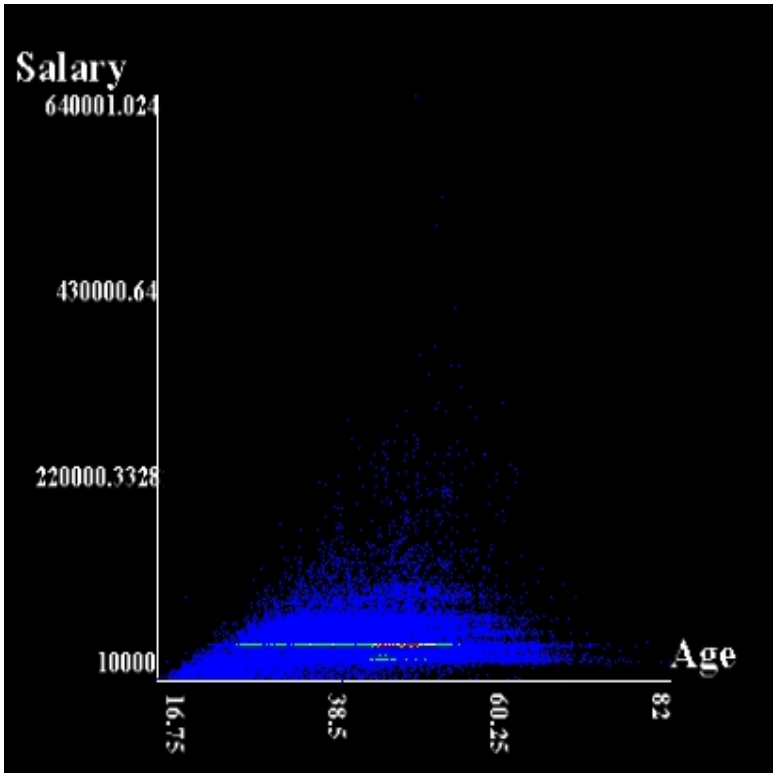


Figure 4.10: Visualizing employee age compared to salary - frequency of employees at each point is indexed to a color scale.

describes the Map Join submodule, in which externally defined transformations can be applied to the input data.

Recognizing that most visualization techniques rely upon numeric input is the first step in developing an effective architecture. From the data perspective, we need to consider the following basic types of data:

nominal Sometimes called categorical, the data is often character-based. This type of data does not have a natural (meaningful, independent of representation) order or scale.

quantitative Numeric data. This type of data obeys an ordering relationship and can be manipulated with arithmetic.

The differences between these data types is closely linked to visualization. At the heart of the matter is the issue of interpreting visual representations of the underlying data values. Of critical importance is the user's understanding of the correspondence between the actual data and visual representation, which must be understood by the user. It is this correspondence that provides the user with the capability to draw conclusions from the visual representation.

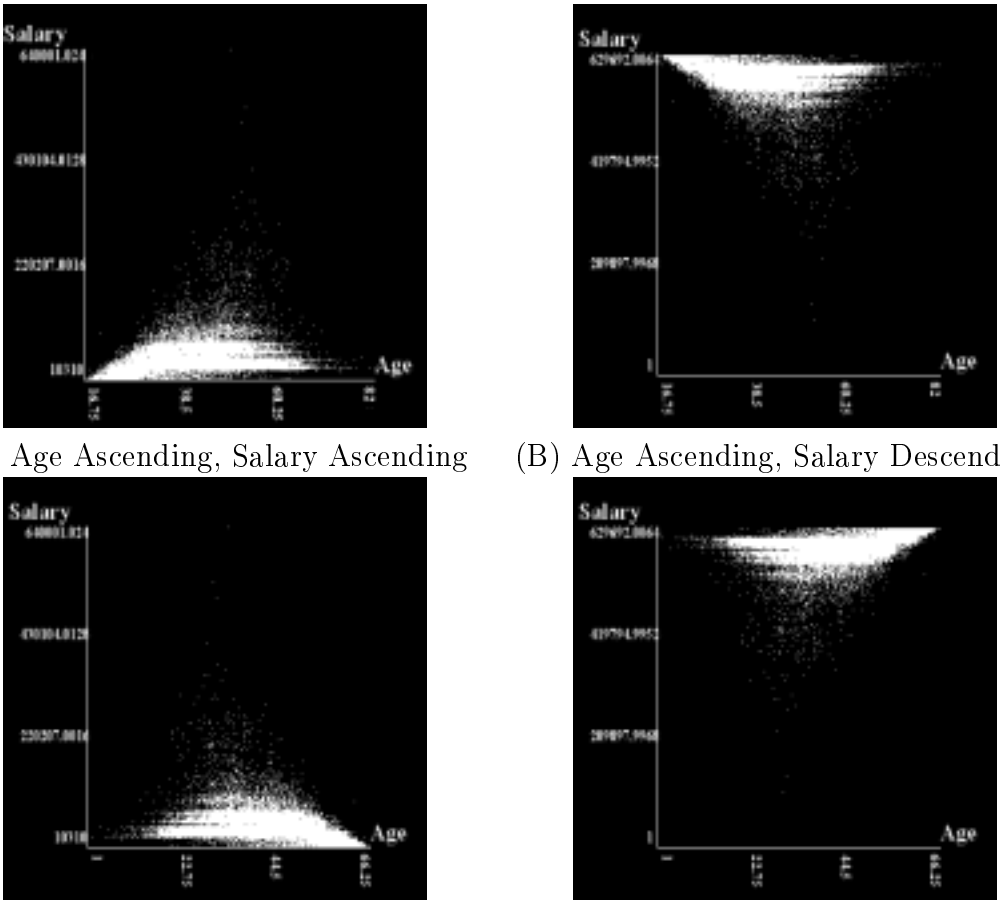
We will illustrate the scope of the problem by using two-dimensional scatterplots due to their straightforward interpretation. In particular, the concepts of *order* and *scale* have a natural meaning for this type of display.

We first consider properties related to the order of the underlying data values. Each axis of the display represents the domain of possible values. For now, we are ignoring limitations of the physical display, which may introduce scaling issues. With regards to the order of values, there are only two possible orderings for the axes: 1) ascending; or 2) descending. Because there are two axes, each axis could be ordered independent of the other, which gives rise to four possible displays. We use the same employee data from the previous section to illustrate these orderings in Figure 4.11.

Ultimately, it is up to the user to decide which of the possible orderings make sense for their application. However, the most natural ordering is probable shown in pane A of Figure 4.11. What makes this ordering natural is the interpretation that higher salaries are above lower salaries and higher ages occur after lower ages.

The order of presentation implies that data values are different from each other. How different on data value is than another data value is determined by the scale of the data values. Using the same employee database, we can see that the distance between points is related to the difference in the underlying values.

While quantitative data values, which have a natural order and scale can be visualized easily, the same thing cannot be said for categorical or nominal values. Problems may arise in the user's interpretation of the visual display based on the



(A) Age Ascending, Salary Ascending (B) Age Ascending, Salary Descending

(C) Age Descending, Salary Ascending (D) Age Descending, Salary Descending

Figure 4.11: Possible orderings for two dimensional scatterplots.

location of datapoints.

Maps

Rather than simply leave it up to the user to rely upon an ambiguous order or scale, the architecture incorporates a mechanism for transforming the data prior to the visualization of the data. This mechanism is embodied in the Map Join submodule of the data preparation module and is accomplished via a user-controlled artifact called a *Map*.

Definition 3 *A map is defined to be a relation S, and is applied to a relation R with a relational join expression θ.* □

Note that for any S and R, the expression $S\theta R$ is also a relation. In the context of the previous section, an aggregation extends to the pair $((S\theta R), L)$. This is a generic definition, in that the contents of S and the description of θ remains to be defined by the user. The key element of this design is that the architecture supports this construct.

Informally, a map can be used to define the order and scale of the underlying data being visualized. We will demonstrate the concept with the following simple example.

Example 1

We will again use the same employee relation that was used in the previous section. In this example, we will use the *sex* and *salary* attributes. The *sex* of each employee is stored as a string in the database with values M or F. The relation in Table 4.3 will serve as the map $SexMapRelation$.

Sex	SexMap
M	3
F	1

Table 4.3: A map for the employee sex attribute.

Note that each entry in the map explicitly defines an order for the attribute values. The map is applied to the employee relation with the expression ($SexMapRelation \bowtie Employee$), which can then be used in the following aggregation:

$$M = ((SexMapRelation \bowtie Employee), \{SexMap, Count(SexMap)\}).$$

The visualization generated from this map join is shown in Figure 4.12.

This is obviously an extremely simple example. The choice of 3, rather than 2, as the map value initially seems a little strange; but it illustrates the impact of the scale in that it creates a gap between the two sexes. □

The contents of the map control the order and the scale. Changes to the map can then affect the output without changing the underlying data. The strength of this approach is this flexibility. Multiple maps can be defined for the same data, each providing different organizations to the output.

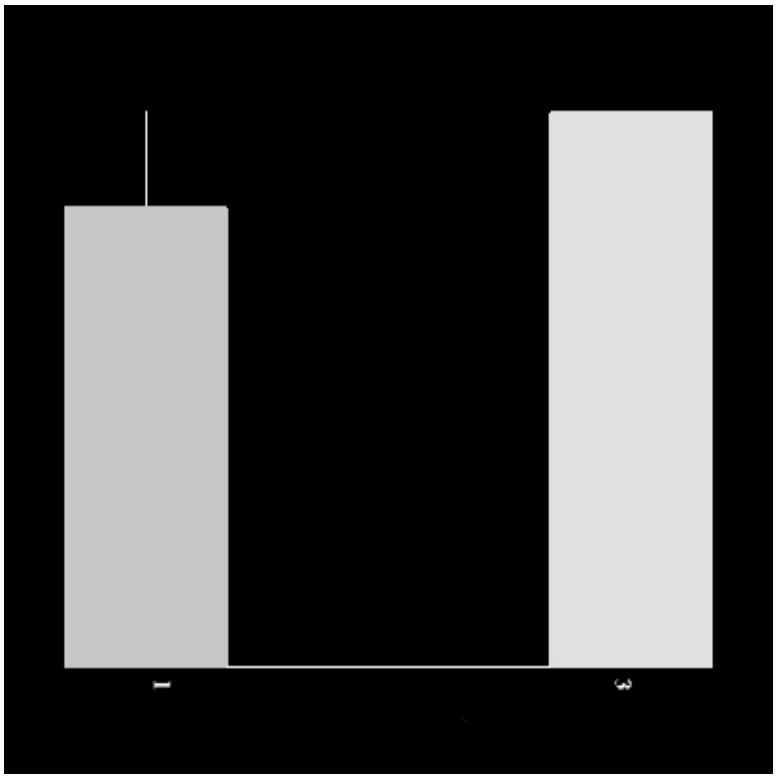


Figure 4.12: Visualizing the number of females (left) to males (right).

Conceptually, any relation in the database can serve as a map. It is likely, however, that users will probably have specific reasons for defining the maps. Since they are physically stored as tables, there can be several techniques used to populate the maps. Certainly, simple SQL insert statements suffice for many applications. In Chapter 5, we describe a query language for specifying the contents of maps.

4.5 Visual Queries

From a database perspective, visualization is closely linked with the concept of a query. Recall from Chapter 3 the definition of a query. Given a relation R, a query $q(R)$ is a function that maps an input instance to an output instance. Within the relational model, the closure property ensures that the output of a query is always a relation. Visualizations, in contrast, are not relations, but are graphic artifacts. Nevertheless, a similar statement can be made for visualizations by defining a *Visual Query*. A Visual Query provides a mechanism for the user to select the attributes from the input data that correspond to elements of the visual representation.

Let $\mathbf{r}$ be an instance of relation R (the pre-image data). Define $Sch(Graphic)$ to be the schema for a particular graphic. For example $Sch(Scatterplot) = X, Y$ for two-dimensional output in which each x, y represents the spatial coordinates for a point in the display. A visual query $vq(\mathbf{r})$ is a function mapping from R to $Sch(Graphic)$.

Note that this representation can be defined irrespective of the semantic content of the data being displayed.

Each graphical display has its own predefined schema. As described by Card and Mackinlay in [CM97], Chi in [Chi00], and Shneiderman in [Shn96] each element of the display corresponds to a specific component of the data. In their taxonomy, they identify a relatively small number of parameters necessary to capture a wide variety of techniques. In general, these parameters can be distilled down to the following types:

Marker The type of graphic object (e.g. point, line, etc.)

Position The location of the graphic object in 1D, 2D, or 3D space.

Shape The physical characteristics of the graphic object.

Size How large the graphic object is relative to the other objects.

Color The color of the object, including shading or transparency.

The prototype system supports a modest number of visualization techniques to provide a proof of concept for the architecture. The following list provides a listing of the techniques that we have implemented along with their associated schemas. Figure 4.13 shows examples of these techniques generated by our system. Note that

the graphical dimensionality (i.e. 1D, 2D, 3D) refers to presentation and not the specific number of input variables.

Scatterplot 2D and 3D variations. The 2D schema choices are $\{X, Y\}$ and $\{X, Y, Color\}$. The *Color* attribute is used to set the color or shade of the point being displayed. The 3D schema choices are similarly defined: $\{X, Y, Z\}$ and $\{X, Y, Z, Color\}$.

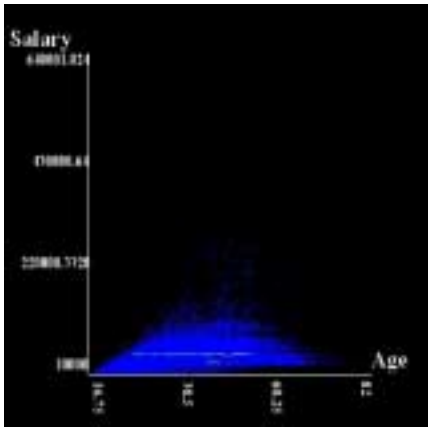
Barchart 1D and 3D variations. The 1D schema is $\{X, Height\}$ or $\{X, Color, Height\}$. The 3D schema is $\{X, Y, Height\}$ or $\{X, Y, Color, Height\}$.

Surface 2D and 3D variations. The 2D schema is $\{X, Y, Color\}$. The 3D schema is $\{X, Y, Color, Height\}$.

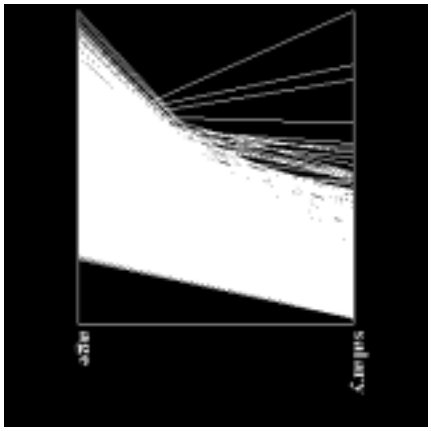
Parallel Coordinates This is a specialized technique with schema $\{A_1, \ldots, A_n\}$.[ID87]

4.6 From Visualization to Data

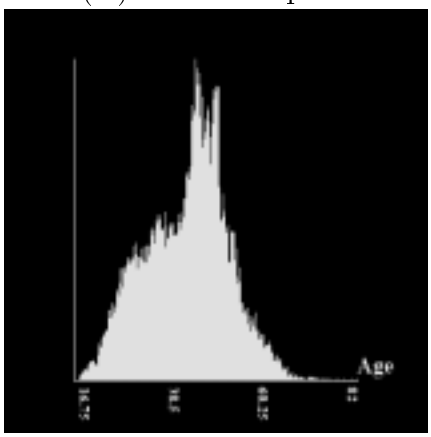
The architecture is designed to remain integrated with a database system. Because of the way that the data preparation module is architected, it is possible to access the underlying data by reversing the previously described process. In this section we outline the formalism necessary for accessing the original data.



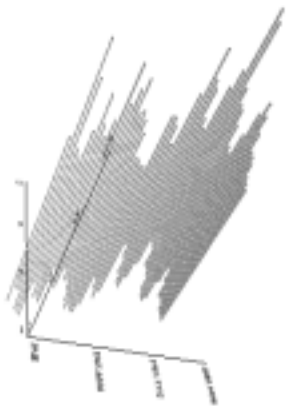
(A) 2D Scatterplot



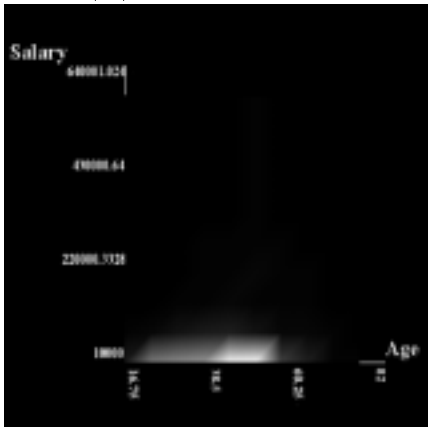
(B) Parallel Coordinates



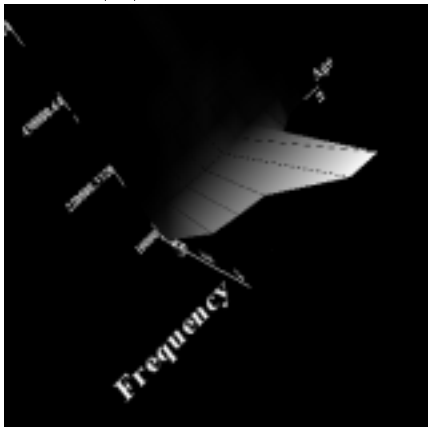
(C) 1D Histogram



(D) 3D Barchart



(E) 2D Surface



(F) 3D Surface

Figure 4.13: Gallery of supported output styles.

First, recall the definition of an aggregation $M = (R, L)$, where R is the input relation and $L = \{L_1, \ldots, L_i, L_{i+1}, \ldots, L_p\}$. The set $\{L_1, \ldots, L_i\}$ are attributes of R, while $\{L_{i+1}, \ldots, L_p\}$ are the aggregate functions applied to attributes of R.

To work from the visualization back to the data, we assume that there is some artifact of interest (i.e. point, bar, line, etc.) For instance, consider a point (x, y) of a two dimensional scatterplot display. Let $L_1 = X$ and $L_2 = Y$ be the fields chosen from the output of the aggregation to be mapped to the schema of the scatterplot. The following query performs the appropriate inverse mapping:

`Select *`

`From` R

`Where` $L_1 = x$ `AND` $L_2 = y$

Note that the resulting inverse operation (drill-down) must only consider the L_i elements of the aggregation that are not aggregate functions. The technique can be applied without actually generating the visualization. However, it makes most sense to consider the technique in the context of a user interacting with the visualization through the front end interface. In the next section we will provide details of the system we have implemented based on our architecture, in which we will describe two such applications of this ability to navigate from the visual output back to the original data.

4.7 Front End

The architecture incorporates a user interface that provides access to the components that we have previously described. As a proof of concept, we developed a system according to the architecture. In this section we provide an overview of this system.

General Interaction Techniques

The primary component of the user interface is the *Application Manager*, which is shown in Figure 4.14. The user creates an application through a visual programming interface.

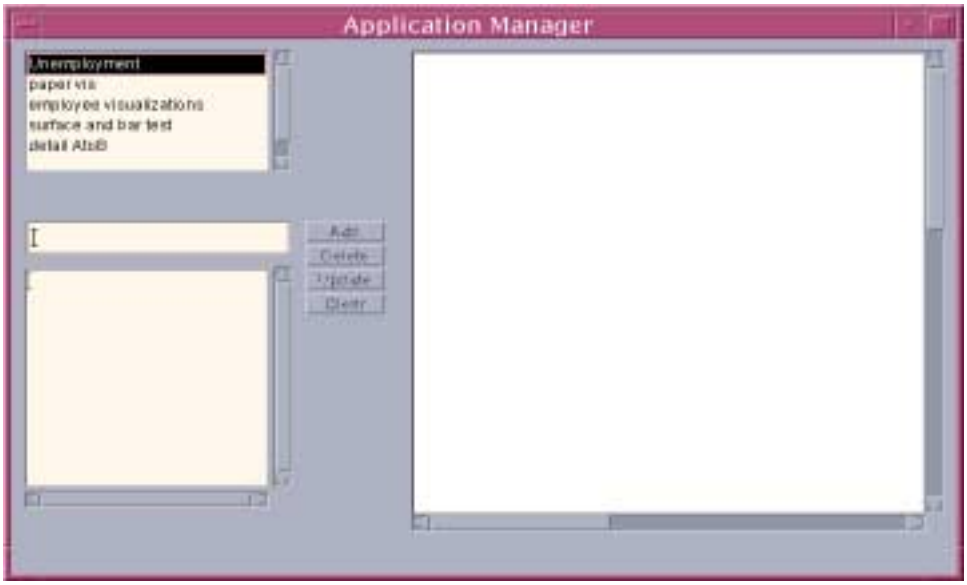


Figure 4.14: The application manager interface.

The basic steps that the user takes is as follows:

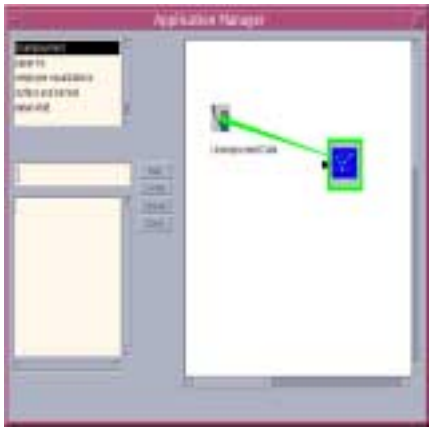
1. Select a data source from a list of tables, queries, or files.
2. Select a visualization tool - 1D, 2D, 3D, etc.
3. Connect the data source to the visualization by using the mouse.
4. Create the aggregation by selecting the attributes and aggregate functions - identify the fields from the input schema that correspond to the elements of the visualization schema.
5. Select the visualization and view the output.

The previous list is a general description of how the user performs the activities necessary to create a visualization. The system is designed to allow for the creation of queries as a separate function, which can be used to shield the users from specifying complex queries. In Figure 4.15 we show the steps that are described above.

The system is implemented in Java, using the Java 3D toolkit for rendering each of the visualization styles. The user can manipulate the visualizations in the following ways:

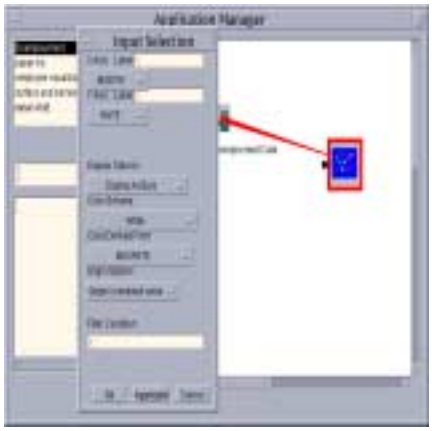
(A) Select an input data source.

(B) Select a visualization tool.

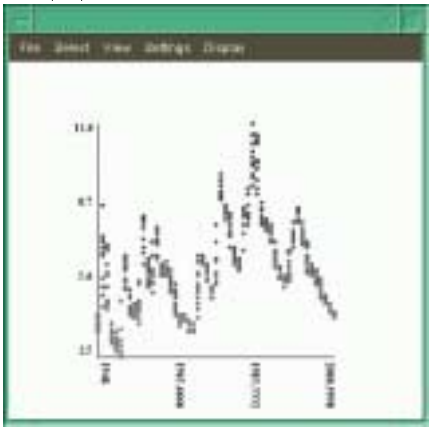


(C) Connect the data to the visualization tool.

(D) Define aggregations.



(E) Select the attributes and define aggregations.



(F) Visualize the results.

Figure 4.15: Selecting and preparing data, and visualizing the results.

Translation Moves the user's viewpoint vertically or horizontally on the display.

Rotation Moves the user's viewpoint around the display.

Zooming Moves the user's viewpoint closer to or further from the display.

These standard interactions can be useful for uncovering hidden details within the visualization. Of course, these types of interactions are not specific to database visualization. The next three that we describe in more detail are specifically related to database visualization.

Details on Demand

The interaction techniques we described previously provide some form of dynamic manipulation of the visualization. Note that in each case it is the user's state that changes and not the state of the visualization. Shneiderman, in [Shn98], provides his mantra for information visualization: "Overview first, then details on demand."

This concept is central to the differentiation between presentation graphics and information visualization. Presentation graphics are used to communicate information according to a predefined model of the user's understanding of the data. Information visualization and information visualization systems must provide methods that allow the user to interrogate the graphic display.

The architecture supports retrieval of details on demand, under the control of the user. Logically, the visualization represents the underlying data, which implies that for each graphical artifact there exists at least one tuple in the input that is used to generate the artifact. The front end enables the user to select an element of the display and retrieve the tuples from the database that are related to the selected element. For example, in Figure 4.16 the user selects a point with the mouse and views the underlying data.

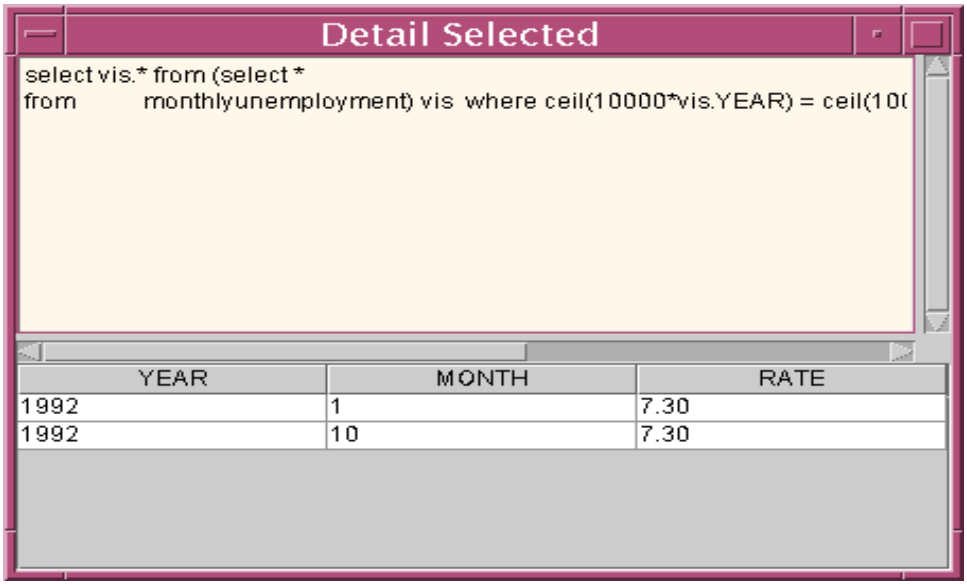


Figure 4.16: Retrieving details on demand.

This technique is accomplished by inversing the aggregation, as described in an earlier section of this chapter. Our approach is strengthened by maintaining the tight integration with the database.

Comparisons Across Visualizations

Beyond just retrieving details on demand, users are also interested in viewing a dataset in many contexts. This is especially important for high-dimensional data, since the number of possible representations exceeds the capacity of systems to easily visualize. Techniques have been developed to deal with high-dimension data, including [Fei92, BA86].

Our approach for dealing with this problem goes beyond the development of techniques to deal with this type of data and is modeled after the concept of scatterplot brushing [BC87]. The main idea behind brushing is that a user can view a dataset under multiple projections. Then, using the mouse, the user can select points in one display and have them highlighted in each of the projections.

To implement brushing for extremely large datasets may be infeasible. Instead, we allow the user to select points in the display using the mouse, which mimics the interaction achieved with brushing. Then, the user can choose to save the selected points. At this stage, the system selects out the subset of the input data using the techniques described in the previous subsection. The selected tuples are stored as a new relation in the database and added to the application manager. After that, the selected data can be shown in other visualization techniques, or can be used in a combined visualization technique that we describe in the next subsection.

Combining Visualizations

In addition to the previously described technique, the front end supports comparisons across datasets by providing a method for combining the visualizations. This method extends beyond just a simple embedding of multiple charts. Instead, the system allows the user to control the combination by specifying how the visualizations are to be combined and scaled relative to each other. There are three supported combination styles:

Overlay This style allows each visualization to occupy the same space. For example, multiple scatterplots as shown in Figure 4.17.

Offset This style allows each visualization to occupy the same space, but offset along one of the axes. For example, multiple barcharts as shown in Figure 4.18.

Tile This style displays each visualization in a separate, but commonly manipulated space. For example, multiple scatterplots for each possible attribute pairing as shown in Figure 4.19.

The system allows the user to define these combinations, and does not place any restriction on the types of visualizations that can be combined.

When the visualizations are combined, however, the data being visualized may be of differing scales. The system supports the user's control of scaling with the following

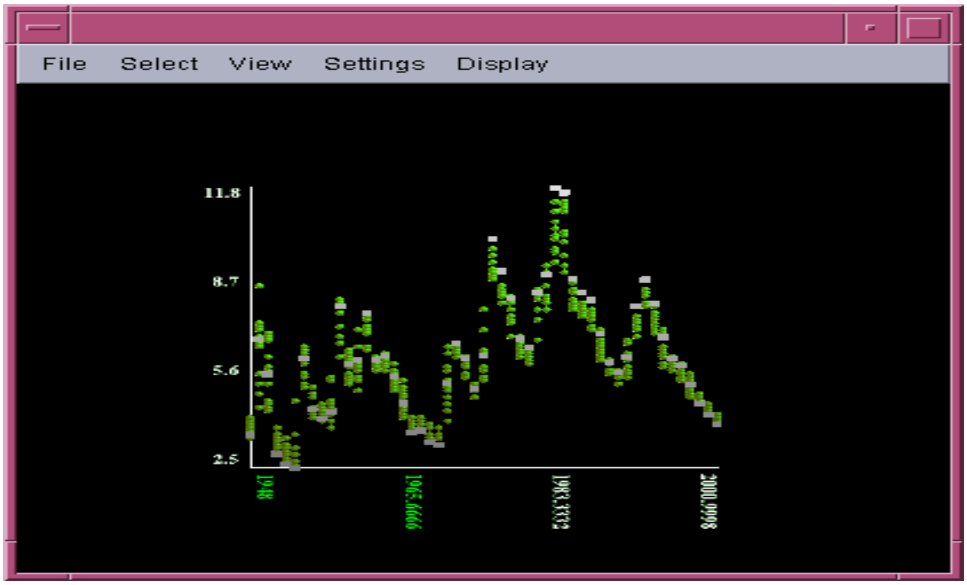


Figure 4.17: Visualizations displayed in overlay format.

options:

Independent Visualizations are scaled independent of the other visualizations in the display. Points at the minima and maxima represent a local condition.

Dependent Visualizations are scaled relative to the ranges of the totality of data being visualized. Points at the minima and maxima represent a global condition.

4.8 Summary

In this chapter we provided details of an architecture supporting visualization in a database environment. The key element of the architecture is the concept of abstracting the preparation of data under the control of the user, which extends fully

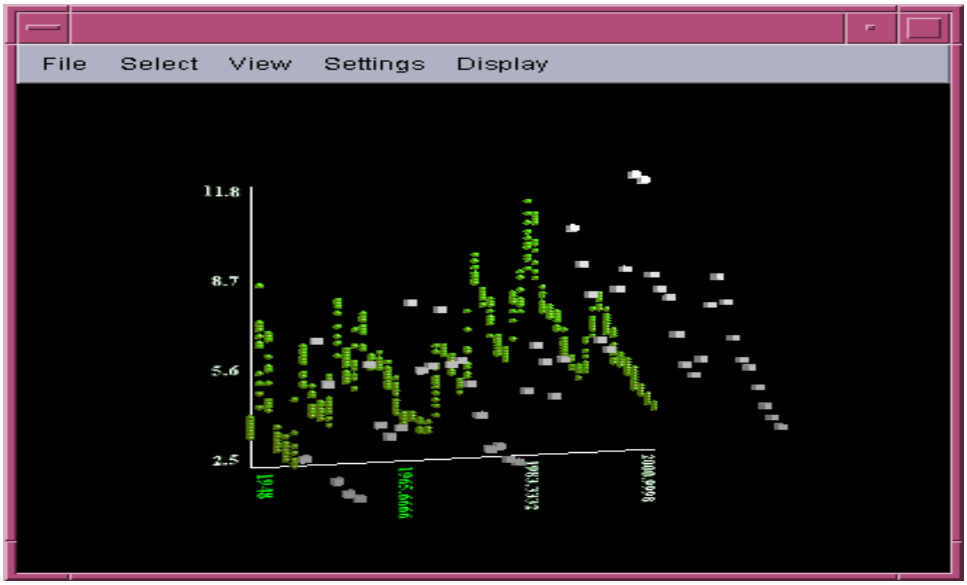


Figure 4.18: Visualizations displayed in offset format.

to a database interface for retrieving, transforming and displaying the data in a visual form.

We have provided examples from a system developed according to this architecture. The system demonstrates a level of integration that is focused on leveraging the strengths of relational database systems. By maintaining this linkage with the database system, we have demonstrated the the user can gain access to information stored in a database by interrogating the graphical display.

The formalism we have presented is consistent with traditional database theory relative to query formulation. In particular the notion of a map application as an aggregate query is a long studied technique in database systems. We described the concept of mapping, which uses a map relation in the database to transform the

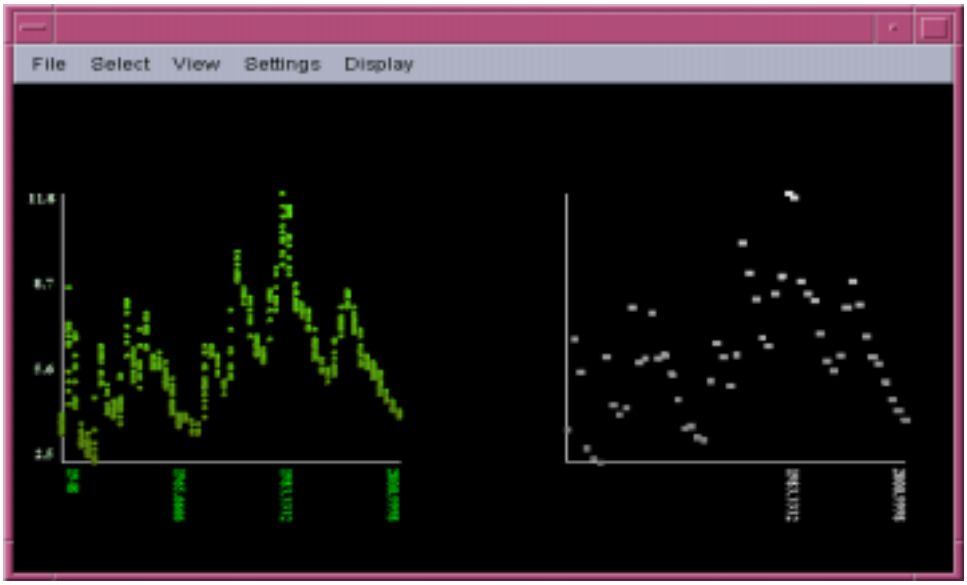


Figure 4.19: Visualizations displayed in tile format.

data, adding both order and scale. In the next chapter we will focus exclusively on a declarative approach we have developed for defining the contents of a map.

5

Mapping

In the previous chapter we outlined our architecture designed to support database visualization. While we described the process of applying a map to input data, we did not discuss specific ways in which the map could be constructed. In this chapter we describe a query language designed to facilitate the transformations that may be required in order to visualize certain datasets.

5.1 Introduction

In the context of our architecture, a map is a relation that is used to transform input data into a form that is more useful for visualization purposes. The input data to be mapped is represented by relation and is physically stored as a table in the database. The contents of the map depend on the application, as specified by the

user. Note that the map controls how data appears in the visualization, but remains separate from the data itself.

We break the mapping process into three stages: (1) specification, (2) construction, (3) application. The first two stages occurring in the Map Specification module of the architecture and the third, which occurs in the Mapping module. For each stage, the essential artifact being used is the *Map*, which defines the specific data transformations that are required for the desired visualization. Before describing the first two stages of the mapping process, we will motivate this concept by describing how maps can be applied to facilitate visualization.

A map is used to add order and scale to the data that is desired to be visualized. The mapping process is conceptualized in Figure 5.1. For example, the input data is "mapped" to a point in the display based on the contents of the map. In this case, we use a two-dimensional scatterplot only as a means to illustrate the mapping process. The approach we present here are independent of any particular choice of graphical display technique.

Formally, a map is a relation instance $\mathbf{s}$ over schema S. $\mathbf{s}$ is applied to data through the use of a relational join operator. Our approach places no restriction on the type of join being performed, allowing for the specific requirements of the visualization problem to be satisfied by a user specified join.

To describe the concepts behind the mapping process we will use an example

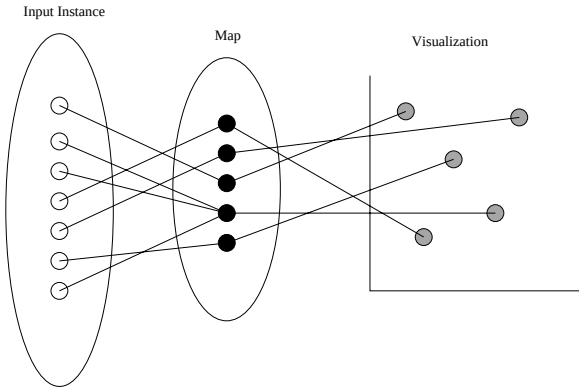


Figure 5.1: The mapping process.

based on the instance of the *Customer* relation shown in Figure 5.2. Note that the schema of *Customer* contains a mixture of categorical and numeric attributes.

CustomerId	State	AnnualSales	Employees
1	IN	1000	2500
7	IL	2000	1150
5	CA	500	6000
75	FL	6500	579
⋮	⋮	⋮	⋮

Figure 5.2: An example *Customer* relation.

The visualization we are interested in viewing is a two-dimensional scatterplot of annual sales compared to geographic region. The sales amount is a numeric value that maps naturally to our display. However, the geographic region must be derived from the state attribute. To accomplish this, consider the map shown in Figure 5.3, which we will refer to as **StateMap**.

State	StateOrder
AK	1
HI	1
CA	2
OR	2
WA	2
⋮	⋮

Figure 5.3: A map that orders states from West to East.

In this example, the map associates with each state a natural number, which adds both order and scale to the categorical value of the state. The application process combines the map with the *Customer* relation by using a natural join. In SQL this map is applied to the original data with the following query:

```
Select  Customer.*, StateMap.StateOrder
From    Customer, StateMap
Where   Customer.State = StateMap.State
```

In order to generate the desired visualization we execute the following aggregate query, which extracts the "mapped" state value and the annual sales for each customer. The resulting visualization is shown in Figure 5.4. The color for each point is derived from the *Frequency* attribute of the aggregate query.

```
Select  vis.StateOrder, vis.AnnualSales, count(*) as Frequency
From    (Select  Customer.*, StateMap.StateOrder
```

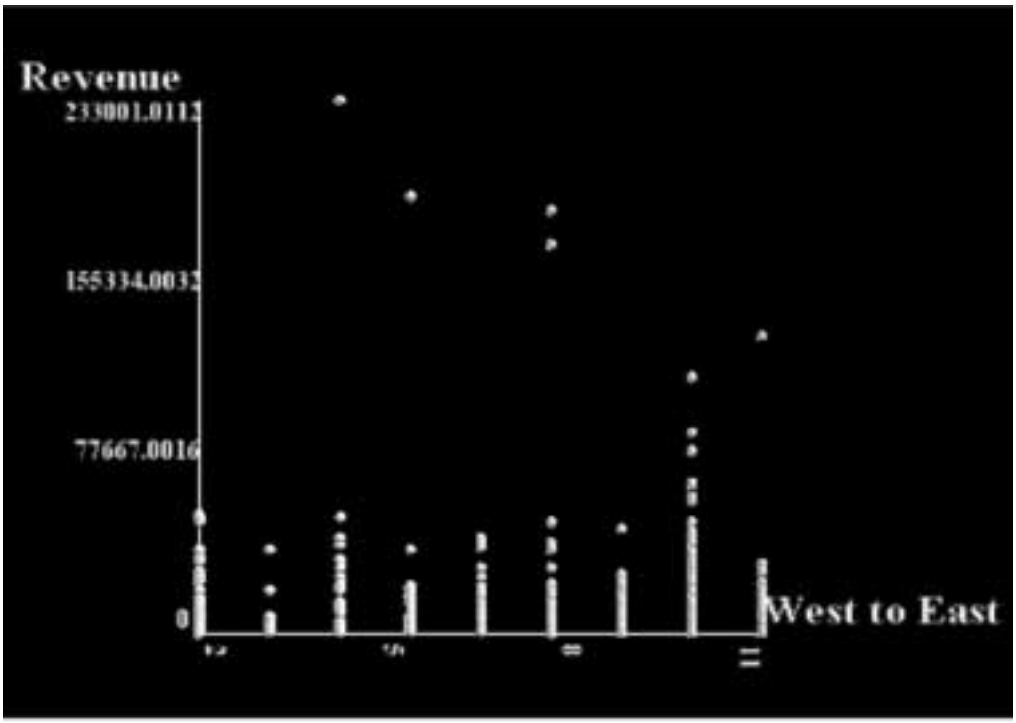


Figure 5.4: An example visualization using the mapped value.

```
        From    Customer, StateMap
        Where   Customer.State = StateMap.State) vis
Group By vis.StateOrder, vis.AnnualSales
```

This approach to mapping provides flexibility in a number of ways. First, changes to the visualization can be effected by simply changing the map, and not the base data. Second, multiple maps can be defined for the same data, allowing for reuse of data in multiple contexts.

There are many ways that a map may be constructed. For example, simple SQL

insert statements are sufficient to populate the map used previously. The following, partial sequence of SQL insert statements creates **StateMap**.

```
Insert Into StateMap Values ('AK', 1)

Insert Into StateMap Values ('HI', 1)

Insert Into StateMap Values ('CA', 2)

    . . .
```

While construction of the map can be accomplished with SQL statements, it is important to note that a user's proficiency with the formulation of queries serves as an effective upper-bound on the complexity of problems that can be addressed using this technique. To address this issue, we propose a declarative approach for specifying maps that shields the user from the SQL syntax. The remainder of this chapter focuses on the mapping language.

5.2 The Mapping Language

In this section we define the formal syntax and semantics of the mapping language - MQL. MQL is based on Datalog, which is a logic-based language that we described in Chapter 3.

We begin the formal description of the mapping language with the definition of an expression:

Definition 4 *An expression E is defined inductively as follows:*

1. A variable V is an expression.

2. A constant c is an expression.

3. If E is an expression, (E) is an expression.

4. If $E_1, \ldots, E_n$ is an expression, then $f(E_1, \ldots, E_n)$ is an expression, where f is an n-ary, supported function.

5. If E_1 and E_2 are expressions, then $E_1 \beta E_2$ is an expression, where β is a mathematical operator $(+, -, *, /)$

We refer to variables within an expression as $Var(E)$. We define a valuation of an expression E as follows:

Definition 5 *Let E be an expression with variables $Var(E) = \{V_1, \ldots, V_k\}$ and $C = \{c_1, \ldots, c_k\}$ be an assigment of values to the variables in E, where the assignment is $V_i = c_i$. We define $E(C)$ to be the valuation of E by evaluating the expression with respect to the assignment given in C.*

Definition 6 *A boolean condition B is defined inductively (using expressions) as follows:*

1. If E_1 and E_2 are expressions, then $E_1 \theta E_2$ is a condition, where θ is a boolean comparator ($=, <, \leq$, etc.)

2. If B_1 and B_2 are conditions, then B_1 AND B_2, B_1 OR B_2 are conditions.

3. If B is a condition, NOT B is a condition.

4. If B is a condition, (B) is a condition.

Let B be a boolean condition and $Expr(B) = \{E_1, \ldots, E_k\}$ be the expressions contained in B. The variables in B are $Var(B) = \cup_{E \in Expr(B)} Var(E)$. We define a valuation of an boolean condition B as follows:

Definition 7 *Let B be an condition with variables $Var(B) = \{V_1, \ldots, V_k\}$ and $C = \{c_1, \ldots, c_k\}$ be an assigment of values to the variables in B, where the assignment is $V_i = c_i$. We define $B(C)$ to be the valuation of B by first evaluating each expression according to Definition 5 and then directly evaluating B according to standard logical operations.*

In form, the mapping language is constructed from rules. Recall from Chapter 3 the definition of a Datalog rule:

$Ans(u) \leftarrow R_1(u_1), \ldots, R_n(u_n)$

The expression on the left hand side is called the head of the rule, while the expression to the right is called the body. A Datalog rule has the following, informal

interpretation. If there exists a valuation of the body of the rule that is true, then the head of the rule is inferred to be true. Relations in the head of a rule are called intensional relations. Relations in the body are called extensional relations. A Datalog program is a set of rules. In general, there is no restriction to the number of intensional or extensional relations in a program. Our approach places the following restrictions on the language:

1. Only one extensional relation is allowed in a program.
2. Only one intensional relation is allowed in a program.
3. Functions are allowed in both the head and body of a rule.
4. A limited form of negation is allowed in the body of a rule.
5. Recursion is not allowed.

Let ψ be a boolean condition with the following restrictions:

Variables $Var(\psi) \subseteq R$.

Constants Values taken from *dom*.

Let α be an arithmetic expression with the same restrictions as ψ. It is important to note that all variables are bound to R. We extend the basic construct of a datalog rule to be of the following form:

$Ans(u_1, \alpha) \leftarrow R_1(u_1) \wedge \psi$

Furthermore, the schema of the intensional relation has the following definition. Let P be a program, and R be the extensional relation in P. Let S be the intensional relation in P, with schema $R \cup \alpha$. We call α the target of the map.

Functions in rules are restricted to be only those functions that are computable, in the same spirit as a function contained in an SQL query. For example, mathematical functions $A+7$, $Floor(A/3)$, etc. are supported, where A is an attribute with domain $\mathbb{N}$. Note that for identity, we simply use the attribute name.

An α in the head of a rule is an expression using the same basic constructs and functions as allowed in the body. Each α in a program must be the same type.

Before proceeding with a description of interpretations of mapping programs we can make the following observation. The schema of the intensional relation is the extensional relation schema, augmented with a new attribute α. Since the variables in a rule are bound to R we arrive at a more concise syntax for rules:

Definition 8 *A map rule is a rule of the form $\alpha \leftarrow \psi$, where α is an expression (Definition 4) and ψ is a boolean condition (Definition 6).*

This provides a benefit from the standpoint that map programs can be defined independent of a specific extensional relation. This allows programs to be defined once and executed in multiple contexts, as required by the user's specific application

requirements. We will now make this statement more precise:

Definition 9 *A map program P is a list of map rules $\langle p_1, \ldots, p_k \rangle$, where each p_i is $\alpha_i \leftarrow \psi_i$ and each α_i is of the same datatype.*

The set of variables contained in P is called the schema of P, defined as:

$$Sch(P) = \cup_{i=1}^{k}(Var(\alpha_i) \cup Var(\psi_i))$$

The valuation of the body of a rule has the following definition:

Definition 10 *Given a tuple t and expression ψ, we define $\psi(t)$ to be the value of the expression ψ after substituting values taken from the tuple t.*

Likewise, the valuation of the head of a rule is defined:

Definition 11 *Given a tuple t and expression α, we define $\alpha(t)$ to be the value of the expression α after substituting values taken from the tuple t.*

We support two different interpretations of map programs - relational, or functional. We first define the relational interpretation:

Definition 12 *Let $\mathbf{s}$ be an instance of relation S and $P = \langle p_1, \ldots, p_k \rangle$ be a map program. Define $I_{\mathbf{R}}(P, \mathbf{s})$ to signify the relational interpretation of P given input instance $\mathbf{s}$. The semantics of $I_{\mathbf{R}}(P, \mathbf{s})$ is:*

$$\{\langle t, \alpha_i(t)\rangle : t \in s \wedge 1 \leq i \leq k \wedge \psi_i(t)\}$$

In other words, the relational interpretation generates output for every program rule whose body evaluates to true. The following definition of the functional interpretation relies upon the order of the rules:

Definition 13 *Let* $\mathbf{s}$ *be an instance of relation* S *and* $P = \langle p_1, \ldots, p_k\rangle$ *be a map program. Define* $I_{\mathbf{F}}(P, \mathbf{s})$ *to signify the functional interpretation of* P *given input instance* $\mathbf{s}$*. The semantics of* $I_{\mathbf{F}}(P, \mathbf{s})$ *is:*

$$\{\langle t, \alpha_i(t)\rangle : t \in s \wedge 1 \leq i \leq k \wedge \psi_i(t) \wedge (\forall j)[1 \leq j < i \Rightarrow \neg\psi_j(t)]\}$$

The following inclusion theorem follows from the provided semantics:

Theorem 1 *Given any program* P *and input instance* $\mathbf{s}$, $I_{\mathbf{F}}(P, \mathbf{s}) \subseteq I_{\mathbf{R}}(P, \mathbf{s})$. □

Given these different semantics for a map program P, we can consider programs as queries. However, we need to be careful that the programs are consistent schema-wise with the input relation. P is *consistent* with S if $Sch(P) \subseteq S$, otherwise it is *inconsistent*. If P is inconsistent, then both $I_{\mathbf{R}}(P, \mathbf{s})$ and $I_{\mathbf{F}}(P, \mathbf{s})$ are the empty set by definition.

Consistency in no way guarantees $I_{\mathbf{R}}(P, \mathbf{s})$ or $I_{\mathbf{F}}(P, \mathbf{s})$ is not the empty set, as shown by the following trivial example.

Example 2

Let $S = \{A\}$ be an extensional relation and P be a map program with the following rule:

$A \leftarrow A > A$

Clearly, the rule is syntactically correct, and $I_{\mathbf{R}}(P, \mathbf{s})$ is consistent. However, there can be no valuation of the body of the rule that is true, and $I_{\mathbf{R}}(P, \mathbf{s}) = \emptyset$.

The following example shows a map program that maps the example *Customer* relation based on the geographic position of the state. A similar construction using SQL insert statements was shown previously.

Example 3

$1 \leftarrow State =$ 'AK' OR $State =$ 'HI'

$2 \leftarrow State =$ 'CA' OR $State =$ 'OR' OR $State =$ 'WA'

...

Implementation

The mapping language that we described in the previous subsection has a syntax very much like Datalog. In this subsection we provide the a description of algorithms that implement the language.

Informally, a relational interpretation can be thought of in the same sense as normal Datalog. We will describe the functional implementation after we have fully

described the relational implementation.

Given a map program $P = \langle p_1, \ldots, p_k \rangle$ and an input relation instance $\mathbf{s}$, define $p_i(t)$ to be the result of evaluating rule p_i on tuple $t \in \mathbf{r}$. In particular, let $p_i(t) = \{\langle t, \alpha_i(t) \rangle\}$ where $\alpha_i(t)$ is the value derived from the α expression in the head of rule p_i, provided that $\psi_i(t)$ is true, $p_i(t) = \emptyset$ otherwise. We can think of $\langle t, \alpha_i \rangle$ in the following terms. If $|R| = n$, t is a tuple of arity n, and $\langle t, \alpha_i(t) \rangle$ is a tuple of arity $n + 1$.

Under the relational interpretation, a naive implementation can be accomplished by breaking P into k distinct programs, each program P_i having a single rule corresponding to each $p_i \in P$. It is easy to see that the following will calculate $I_{\mathbf{R}}(P, \mathbf{s})$:

$$I_{\mathbf{R}}(P, \mathbf{s}) = \cup_{i=1}^{k} \{p_i(t) | t \in \mathbf{s}\}$$

Since the language is restricted to support only total functions, with input either constants or variables bound to input instances, we can preprocess the input and replace each function with its calculated value. We make this statement more precise with the following definition:

Definition 14 *Let P be a program and S be a relation, such that $Sch(P) \subseteq S$. We define P^{FF} to be a function-free program, created by using the following transformation.*

We assume that each function contained in P has input that is either a constant, or an attribute of S. Let $F = \{f_1, \ldots, f_\ell\}$ be the functions occurring in P. Replace each occurrence of f_i in P with a new variable $A_{f_i} \notin S$. □

Note that the transformation process from P to P^{FF} results in a program that is inconsistent with S. That is, $I_{\mathbf{R}}(P^{FF}, \mathbf{s}) = \emptyset$, since $Sch(P^{FF}) \not\subseteq S$. We define the following transformation of S:

Definition 15 *Let P be a program, S be a relation, and P^{FF} be a function-free program transformed according to the previous definition. Define $S^{P^{FF}}$ to be transformed from S in the following way.*

Create $S^{P^{FF}}$ by augmenting the schema of S with the new attributes generated by the function-free transformation procedure. In particular, let $\{A_{f_1}, \ldots, A_{f_\ell}\}$ be these attributes. A valuation for each f_i is determined by substituting values from a tuple $t \in \mathbf{s}$. Create $\mathbf{s}^{P^{FF}}$ by evaluating each f_i and creating a new tuple t', augmented with the function evaluations. □

With these two transformations the following lemma is given:

Lemma 1 *For every MQL program P and input instance $\mathbf{s}$, when P^{FF} and instance $\mathbf{s}^{P^{FF}}$ are created from P and $\mathbf{s}$ using the procedures given in Definitions 14 and 15, then $I_{\mathbf{R}}(P, \mathbf{s}) = \Pi_{S,\alpha}(I_{\mathbf{R}}(P^{FF}, \mathbf{s}^{P^{FF}}))$.*

Proof: This follows immediately from the definition of the transformations. □

The lemma allows us to eliminate any concerns related to the introduction of functions in the language. The following example demonstrates an application of Lemma 1.

Example 4

Let $S = \{A_1, A_2\}$ with instance **s** as follows:

A_1	A_2
1	1
2	-2
3	2

Let P be the following program:

$$ABS(A_2) \leftarrow A_1 < (A_2 - 1)$$

In this example $F = \{ABS(A_2), (A_2 - 1)\}$, where ABS is the absolute value function. We create a new relation $S^{PFF} = \{A_1, A_2, A_3, A_4\}$. We apply the function to each tuple in **s** and populate $\mathbf{s}^{PFF}$:

A_1	A_2	A_3	A_4
1	1	1	0
2	-2	2	-3
3	2	2	1

According to the transformation procedure, P^{FF} is the following program:

$A_3 \leftarrow A_2 < A_4$

Using this construction, we claim $I_{\mathbf{R}}(P, \mathbf{s}) = \Pi_{S,\alpha}(I_{\mathbf{R}}(P^{P^{FF}}, \mathbf{s}^{P^{FF}}))$. Clearly, if P does not contain any functions, $P = P^{FF}$ and $\mathbf{s} = \mathbf{s}^{P^{FF}}$. In this case, however, $I_{\mathbf{R}}(P, \mathbf{s})$ is:

A_1	A_2	α
2	-2	2

and $I_{\mathbf{R}}(P^{FF}, \mathbf{s}^{P^{FF}})$ is:

A_1	A_2	A_3	A_4	α
2	-2	2	1	2

We immediately see that $I_{\mathbf{R}}(P, \mathbf{s}) = \Pi_{A_1,A_2,\alpha}(I_{\mathbf{R}}(P^{FF}, \mathbf{s}^{P^{FF}}))$ and the construction holds. □

Since MQL does not allow recursion, it is useful to describe the expressive power of the language by considering a language of equivalent power. First, we provide an equivalence between MQL (under the relational interpretation) and relational algebra (RA). While RA does not contain functions, it is clear that the functions contained in rules of an MQL program do not add to the expressive power of the language. Consequently, any function in MQL could be straightforwardly implemented in the RA language.

Theorem 2 *For every function-free MQL program P, there exists an equivalent RA expression.*

Proof: We need to show that a map program P can be rewritten into an equivalent RA expression. The rewriting strategy is as follows. We assume an input relation $R = \{A_1, \ldots, A_n\}$ and a map program $P = \langle p_1, \ldots p_k \rangle$. For each $p_i \in P$ construct an RA expression E_i: $\pi_{A_1,\ldots,A_n,\alpha_i}(\sigma_{\psi_i}(R))$. The complete RA expression E is simply $E_1 \cup \ldots \cup E_k$. □

It is important to note that the equivalence between MQL and RA does not involve set minus.

A simple analysis of the RA approach yields a time complexity of $O(k|\mathbf{r}|)$ where k is the number of rules in P. Note that this assumes a standard approach for database optimizers, in which each query is computed independently, and the results merged.

Note that, while each MQL program has an equivalent RA query, the reverse is not the case. For example, MQL programs cannot represent negative information. Note that the "tuple at a time" semantics of MQL disallows set difference.

Theorem 3 *MQL programs are monotonic.*

Proof: This result follows immediately from the proof of Theorem 2 and the fact that the construction of $\mathbf{s}^{PFF}$ in Lemma 1 is monotonic. □

Input :
  MQL Program P
  Input Instance $\mathbf{r}$

Output :
  Output Instance$\mathbf{s}$

MQLAlgorithm ::
  $\mathbf{s} := \emptyset$;
  For Each $t \in \mathbf{r}$ do
   For Each $p_i \in P$ do
    $\mathbf{s} := \mathbf{s} \cup p_i(t)$;
   od
  od
  return $\mathbf{s}$;

Figure 5.5: Algorithm to compute $I_{\mathbf{R}}(P, \mathbf{s})$.

Even though we can construct an RA expression equivalent to any MQL program, a more efficient implementation can be accomplished by processing the input in a "tuple at a time" fashion. The improved algorithm for computing $I_{\mathbf{R}}(P, \mathbf{s})$ is shown in Figure 5.5.

There are a number of comments to make about the algorithm. First, the order of the rules does not affect the outcome. Second, computing $p_i(t)$ is accomplished deterministically by first replacing each variable in ψ_i with the attribute values in t, then checking whether the expression evaluates to "true". If ψ_i is true, compute the $\alpha_i(t)$ value and create the tuple $\langle t, \alpha_i(t) \rangle$.

In contrast with the RA approach, this algorithm is more efficient. In particular, the complexity of the algorithm is $O(|\mathbf{r}|)$, an improvement by a factor of k, because it

reads each tuple from the input only once, while the RA approach incurs the cost of reading the tuple k times - once for each rule. This is another important reason for considering MQL as a viable language, since in practice k may be significantly large.

Implementing the Functional Interpretation

The main difference introduced by the functional interpretation is in how the rules are evaluated. A functional interpretation, as the name implies, maps input tuples to at most *one* output tuple. In order to accomplish this, we must control the order of evaluation for the rules. This order is provided by the user.

This concept of a user defined ordering of the rules was first introduced by Imielinski and Naqvi in [IN88], in which they present a rule algebra. Their algebra is simple and elegant, allowing the users to explicitly specify how rules are to be processed.

Let $P = \langle p_1, \ldots, p_k \rangle$ be an MQL program. We will describe the operators of the rule algebra with this program. The operators in the rule algebra are $\cup, \circ$ and *, with the following definitions.

Parallel Execution $p_i \cup p_j$ – compute each term separately and union the result.

Consecutive Execution $p_i \circ p_j$ – compute p_i and then compute p_j.

Iterative Execution $(p_i)^*$ – compute p_i until a fixpoint is reached.

Note that the rule algebra allows the composition of statements like a regular expression. The rule algebra provides a fine grained level of control for the user. For example, the following statement defines the normal inflationary semantics for datalog programs:

$(p_1 \cup \ldots \cup p_k)^*$

An expression in the rule algebra that is equivalent to our previously described relational interpretation would be:

$p_1 \cup \ldots \cup p_k$

Since we restrict MQL programs to not have recursive structure, the following consecutive expression also a relational interpretation:

$p_1 \circ \ldots \circ p_k$

For functional interpretations we introduce an exclusive-consecutive operator, $\oplus$. The $\oplus$ operator is described as follows. If p_i and p_j are rules in an MQL program, then $p_i \oplus p_j$ means to execute p_j only if p_i fails.

Specifying P as a list of rules provides the order of execution for an MQL program to be interpreted functionally. The following describes an implementation for the functional interpretation:

$$I_{\mathbf{F}}(P, \mathbf{s}) = \cup_{i=1}^{k} \{p_i(t) | t \in \mathbf{s} \wedge (\forall j)[1 \leq j < i \implies p_j(t) = \emptyset]\}$$

```
Input :
  MQL Program P
  Input Instance r
  Interpretation MapType

Output :
  Output Instances

MQLAlgorithm ::
  s := ∅;
  For Each t ∈ r do
   For Each p_i ∈ P in order do
    If MapType == Functional
     If p_i(t) ≠ ∅
      s := s ∪ p_i(t);
      Break;
     Fi
    Fi
    If MapType == Relational
     s := s ∪ p_i(t);
    Fi
   od
  od
  return s;
```

Figure 5.6: Algorithm to compute $I_{\mathbf{F}}(P, \mathbf{r})$ or $I_{\mathbf{R}}(P, \mathbf{r})$.

Using the order of the rules, the algorithm presented in Figure 5.6 computes the functional interpretation of an MQL program. We extend the algorithm with a flag to support either interpretation. It is evident that, with proper setting of $MapType$, this program faithfully implements $I_{\mathbf{R}}$ or $I_{\mathbf{F}}$ as defined previously.

5.3 Program Transformations

In this section we introduce an alternative method for implementing functional interpretations of MQL programs based on performing a program transformation.

Definition 16 *Given a program P, define P^τ be the original program transformed according to the following procedure:*

For each $p_i \in P$ append the statement $\wedge\neg(\psi_i)$ to every succeeding p_j in the ordered list of rules in P. This yields a program P^τ of the form:

$\alpha_1 \leftarrow \psi_1$

$\alpha_2 \leftarrow \psi_2 \wedge \neg(\psi_1)$

$\ldots$

$\alpha_k \leftarrow \psi_k \wedge \neg(\psi_1) \wedge \ldots \wedge \neg(\psi_{k-1})$

Figure 5.7 shows the context of this transformation. The black dots indicate the results of applying the transformation to one arbitrary instance. Of particular interest are the identities and containment between the results.

While the functional interpretation appears different than the relational interpretation, the following theorem shows that every program that is intended to be interpreted functionally can be transformed into an equivalent relationally interpreted program.

Theorem 4 $I_{\mathbf{F}}(P, \mathbf{s}) = I_{\mathbf{R}}(P^\tau, \mathbf{s})$

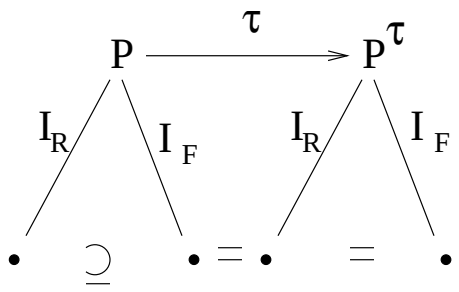


Figure 5.7: Context of the program transformation process.

Proof: Let P be an MQL program. Assume $t \in I_{\mathbf{F}}(P, \mathbf{s})$. If $t \in I_{\mathbf{F}}(P, \mathbf{s})$, then there must be a $p_i \in P$ and a tuple $u \in \mathbf{s}$, such that $p_i(u) = t$. Let p_i and p_j be two different rules in P, such that $i < j$. Since P is evaluated functionally, $p_j(u) \notin I_{\mathbf{F}}(P, \mathbf{s})$ by definition.

Create program P^τ according to the rewriting procedure given in Definition 16. Let the transformed rules of P^τ be denoted by $\{q_1, \ldots, q_k\}$. It is clear that $q_i(u) \in I_{\mathbf{R}}(P^\tau, \mathbf{s})$, assuming that p_i occurs before p_j. Obviously, $q_j(u) \notin I_{\mathbf{R}}(P^\tau, \mathbf{s})$, since q_j is of the form $\alpha_j \leftarrow \psi_j \wedge \neg(\psi_i)$. If both $q_i(u)$ and $q_j(u)$ are in $I_{\mathbf{R}}(P^\tau, \mathbf{s})$ then we have the contradiction $\psi_i \wedge \neg(\psi_i)$ and the construction holds. □

Using the transformation process and the interpretation definitions, it is easily seen that $I_{\mathbf{R}}(P^\tau, \mathbf{s}) = I_{\mathbf{F}}(P^\tau, \mathbf{s})$

5.4 Extending MQL

In this subsection we describe two additional features of MQL that we support. The first feature allows for tuples to be excluded from further processing. The second allows for special handling if every rule fails for a tuple.

Tuples may be excluded from further processing by introducing a rule of the following form:

$Except \leftarrow \psi$

The semantics of such a rule, in the context of a relational interpretation is:

$$I_{\mathbf{R}}(P, \mathbf{s}) =$$

$$\{\langle t, \alpha_i(t)\rangle : t \in s \wedge 1 \leq i \leq k \wedge \alpha_i \neq Except \wedge \psi_i(t)$$

$$\wedge(\forall j)[1 \leq j < i \Rightarrow \neg(\alpha_j = Except \wedge \psi_j(t))]\}$$

Similarly, the functional interpretation is:

$$I_{\mathbf{F}}(P, \mathbf{s}) =$$

$$\{\langle t, \alpha_i(t)\rangle : t \in s \wedge 1 \leq i \leq k \wedge \alpha_i \neq Except \wedge \psi_i(t)$$

$$\wedge(\forall j)[1 \leq j < i \Rightarrow \neg\psi_j(t)]\}$$

The following transformation procedure is used to create *Except-Free* programs.

Definition 17 *Given a program P, define P^{XF} to be the original program transformed according to the following procedure:*

First, perform the transformation given in Definition 16. Note that this initial step

ignores the heads of rules and simply propagates negated bodies to later rules. Second, simply remove all rules of the form $Except \leftarrow \psi$ from the transformed program. □

The following example demonstrates the use of this transformation:

Example 5

Let P be the following program:

$Except \leftarrow State =' IL'$

6 $\leftarrow State =$ 'IN' OR $Employees > 1000$

The transformed program, P^{XF} is shown below:

6 $\leftarrow$ ($State =$ 'IN' OR $Employees > 1000$) AND NOT ($State =$ 'IL') □

It is important to note that this transformation is defined on programs, not interpretations. The following lemma follows immediately from the definition of the transformation:

Lemma 2 *Given programs P and P^{XF}, where P^{XF} is created from P according to Definition 17, then:*

$I_{\mathbf{F}}(P, \mathbf{s}) = I_{\mathbf{F}}(P^{XF}, \mathbf{s})$, and

$I_{\mathbf{R}}(P, \mathbf{s}) = I_{\mathbf{R}}(P^{XF}, \mathbf{s})$ □

It is important to note that this transformation process holds true for relational

interpretations only in the case that the Except rules precede all other rules. Nevertheless, it is clear that the Except rules remain as a simple syntactic extension of the language and do not express queries that cannot be stated by rewriting the rules.

The second extension we describe is the allowance of a single rule of the form $\alpha \leftarrow Else$ in MQL programs. The meaning of such a rule is straightforward: if every rule preceding an Else rule fails, evaluate the head of the Else rule and place the result in the output.

Again, we provide a defined program transformation to create *Else-Free* programs.

Definition 18 *Given a program P, define P^{EF} to be the original program transformed according to the following procedure:*

We restrict MQL programs to having a single Else rule - let $p_{Else} \in P$ be the sole Else rule. First, for every non-Except, non-Else rule p_i that precedes p_{Else}, replace the body of p_{Else} with the expression $\neg(\psi_i) \wedge \ldots \wedge \neg(\psi_{Else-1})$. The Else rule p_{Else} is now of the form $\alpha_{Else} \leftarrow \wedge\neg(\psi_i) \wedge \ldots \wedge \neg(\psi_n)$. Lastly, perform the transformation given in Definition 17 to make the transformed program Except-Free. □

The following example demonstrates the transformation procedure:

Example 6

Let P be the following MQL program:

$Except \leftarrow State =$ 'IL'

$6 \leftarrow State =$ 'IN' OR $Employees > 1000$

$7 \leftarrow Else$

The program first excludes companies from Illiois, then maps companies from Indiana or companies having more than 1000 employees to the value 6. All other companies are mapped to the value 7. The transformed program, P^{EF} is shown below:

$6 \leftarrow State =$ 'IN' OR $Employees > 1000$ AND NOT ($State =$ 'IN')

$7 \leftarrow$ NOT ($State =$ 'IN' OR $Employees > 1000$) AND NOT ($State =$ 'IN')

□

Similar to the Except rules, Else rules are syntactic sugar and do not extend the power of the language. We simply state the following lemma, which follows from the defined transformation.

Lemma 3 *Given programs P and P^{EF}, where P^{EF} is created from P according to Definition 18, then:*

$I_{\mathbf{F}}(P, \mathbf{s}) = I_{\mathbf{F}}(P^{EF}, \mathbf{s})$, and

$I_{\mathbf{R}}(P, \mathbf{s}) = I_{\mathbf{R}}(P^{EF}, \mathbf{s})$ □

Lastly, we provide a complete algorithm for computing MQL queries, incorporating both Except and Else rules, as shown in Figure 5.8.

```
Input :
  MQL Program P
  Input Instance r
  Interpretation MapType

Output :
  Output Instances

MQLAlgorithm ::
  s := ∅;
  For Each t ∈ r do
   RuleSucceeded := false;
   For Each p_i ∈ P in order do
    If MapType == Functional
     If p_i(t) ≠ ∅
      If p_i(t) is not an Except rule
       s := s ∪ p_i(t);
      Fi
      RuleSucceeded := true;
      Break;
     Fi
    Fi
    If MapType == Relational
     If p_i isan Exceptrule
      If the body of p_i evaluatestotrue
       RuleSucceeded := true;
       break;
      Fi
     Fi
     s := s ∪ p_i(t);
     RuleSucceeded := true;      Fi
   od
   If RuleSucceeded == false
    s := s ∪ p_Else(t);
   Fi
  od
  return s;
```

Figure 5.8: Algorithm to compute the output of an MQL^{Relational} or MQL^{Functional} program, incorporating Ecept and Else rules.

5.5 MQL Grammar

Figure 5.9 shows the BNF grammar for the implemented mapping language.

5.6 Mapping Example

In this Section we provide a description of the prototype system's implementation of the mapping process. 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. In the left pane of Figure 5.10, 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. The right pane 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
```

```
<MAP_PROGRAM>        ::= <RULE> [<Line Feed> <RULE>]*
<RULE>               ::= <HEAD> "<-" <BODY>
<HEAD>               ::= <EXPRESSION>
                     |   "Exclude"
<BODY>               ::= <BOOLEAN_EXPR>
                     |   "Else"
                     |   Null
<BOOLEAN_EXPR>       ::= <CONDITIONAL_EXPR>
                         [<BOOLEAN_OPERATOR>
                         <CONDITIONAL_EXPR>]*
                     |   ["NOT"] "(" <BOOLEAN_EXPR>
                         [<BOOLEAN_OPERATOR>
                         <BOOLEAN_EXPR>]* ")"
<BOOLEAN_OPERATOR>   ::= "AND" | "OR"
<CONDITIONAL_EXPR>   ::= <STATEMENT> [<RELATION>
                         <STATEMENT>]
<STATEMENT>          ::= ["NOT"] <EXPRESSION>
<RELATION>           ::= "<" | "<=" | ">" | ">=" | "=" | "!="
<EXPRESSION>         ::= <EXPRESSION> [<OPERATOR>
                         <EXPRESSION>]*
                     |   "(" <EXPRESSION> ")"
                     |   <FUNCTION>
                     |   <ELEMENT>
<OPERATOR>           ::= "+" | "-" | "*" | "/" | "^" | "&" | "#" | "|"
<FUNCTION>           ::= "ABS(" <EXPRESSION> ")"
                     |   "FLOOR(" <EXPRESSION> ")"
                     |   "CEIL(" <EXPRESSION> ")"
                     |   "SQRT(" <EXPRESSION> ")"
                     |   "GRAYCODE(" <EXPRESSION> ["," 
                         <EXPRESSION>]* ")"
                     |   "LENGTH(" <EXPRESSION> ")"
                     |   "SUBSTRING(" <EXPRESSION> ","
                         <EXPRESSION> "," <EXPRESSION> ")"
<ELEMENT>            ::= <VARIABLE>
                     |   <CONSTANT>
                     |   "(" <EXPRESSION> ")"
<VARIABLE>           ::= A_i ∈ R
<CONSTANT>           ::= number
                     |   "string"
```

Figure 5.9: Grammar for the mapping language

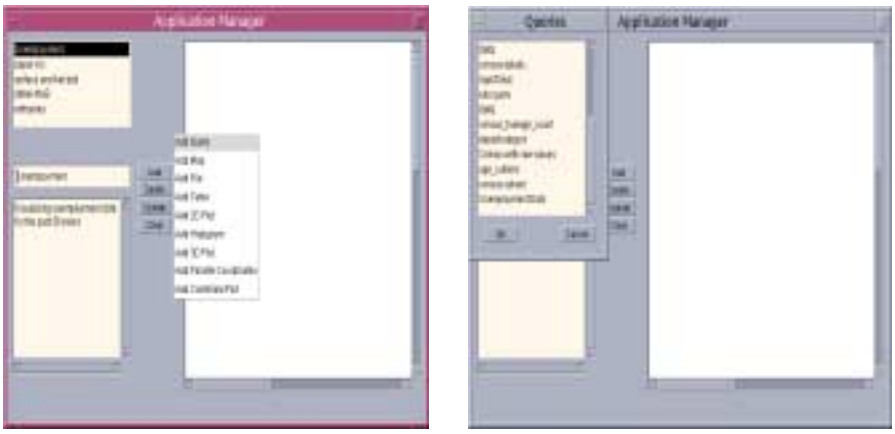


Figure 5.10: The user right-clicks with their mouse to see a menu of available application objects. Selecting "Add Query", the user is presented a list of available queries that can be visualized.

```
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 5.11, 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. The right pane of the figure shows the result of the operation, in which

Figure 5.11: The user user the mouse to connect an input data source to a map. After the map has been applied to the input data, a new object is automatically added to the desktop.

a new object is added to the desktop that represents the input data transformed according to the map.

With the map applied to the data we can plot the data using similar steps. In Figure 5.12 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.

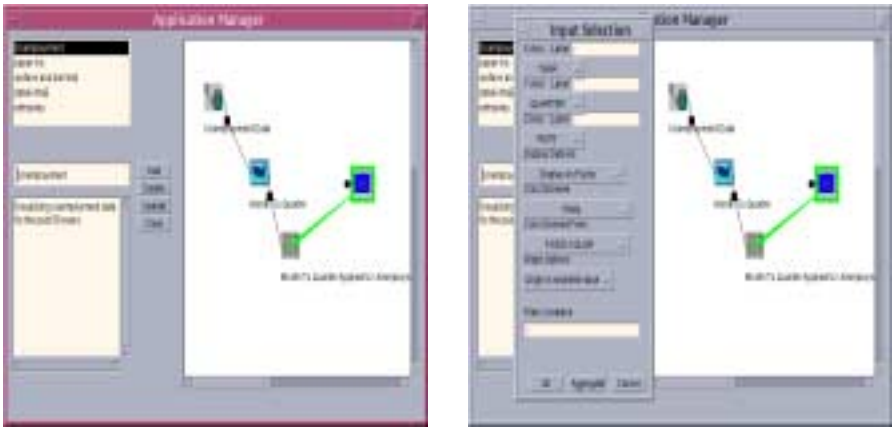


Figure 5.12: The user adds a plotting object to the desktop and connects the mapped data to the plotting object. The user is then presented with options based on the input data and the type of plot.

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 5.13 shows the unemployment data in using a 3D scatterplot display. The original, unmapped data is shown in the right pane of the figure.

5.7 Summary

This chapter introduced the mapping language MQL, which we employ in the architecture. The language allows users to add both order and scale to data. Functionally, the language is equivalent to the monotonic relational algebra, although the implementation of the language is more efficient, with all programs executable in

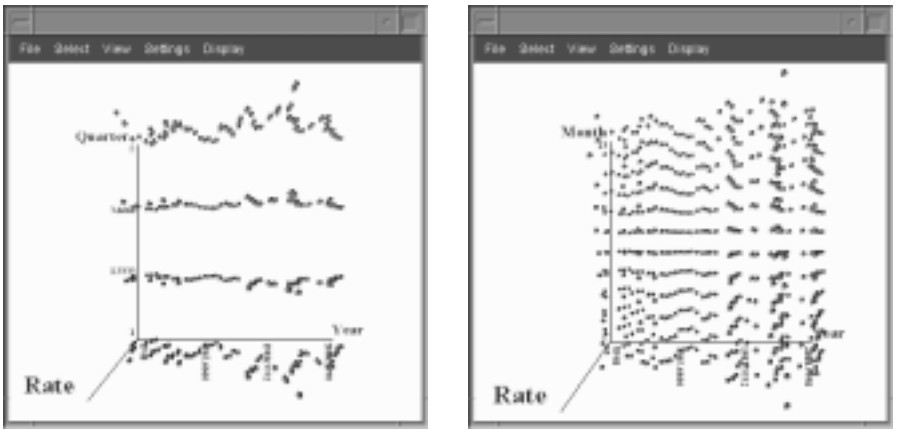


Figure 5.13: The left pane shows the mapped unemployment data within a 3D scatterplot. The right pane shows the original data in its unmapped form.

linear time with respect to the size of the input.

While the language is limited by design, it can be used to express quite complicated expressions. We provide two interpretation techniques - functional and relational. We proved that the functional interpretation was less powerful than the relational interpretation. Lastly we showed how the user interacts with the map programs with the system prototype.

6

Usability Results

This chapter provides results from a usability experiment comparing the mapping language introduced in Chapter 5 to SQL. The research experiment focuses on a specific type of query task, namely classification queries. Classification is the process of assigning input data to discrete classes according to application specific criteria. This mapping language is well suited for this type of task. In particular, for visualization purposes, we seek to discover whether the mapping language offers any advantages over SQL for transforming the data. The usability experiment measures the effectiveness, efficiency and satisfaction of novice and expert users performing a variety of classification tasks.

6.1 Introduction

Classification is the process of assigning input data to discrete classes according to application specific criteria. Simple examples abound for this type of task. For example, employees in an employee database may be classified according to their salary into "High", "Medium" and "Low" classes.

In this research we consider classifications that are expressed by the user in two ways. First, a classification is *fixed* if the class values are explicitly assigned according to a pre-defined scheme. For fixed classifications, the class values are known in advance of the actual classification task. The definition can be stated in the form of a series of *If .. Then* statements. For example, consider the following sentences that define the employee salary classification:

1. If an employee's salary is less than $30,000 then assign the employee to the "low" salary class.

2. If an employee's salary is between $30,000 and $60,000 then assign the employee to the "medium" salary class.

3. If an employee's salary is more than $60,000 then assign the employee to the "high" salary class.

A classification is *derivable* if the class values are assigned according to a calculation. In other words, the class values are not known prior to classifying the input data. For example, we can assign employees to classes using the statement $SalaryClass = floor(Salary/1000)$, which effectively "bins" employees into salary ranges.

This research considers schemes that may be either definable or derivable to the extent that a user can describe the classification in some declarable form. We do not consider algorithmic techniques encountered in data mining, such as automatic cluster detection or discretization, which are described in [HK01, HMS01].

In this research we have presented a declarative language (MQL) that supports both definable and derivable classifications. The language has a syntax that succinctly describes the classifications according to a set of rules. We compare our approach with an Structured Query Language (SQL) approach in a controlled usability experiment.

The usability experiment seeks to determine whether one approach is superior to another in a controlled fashion. Previous experiments have compared systems equivalent to each other. That is, each system was fully equivalent in expressible power. differing in syntactic form, or in interaction style. This research differs from previous approaches in that MQL is strictly weaker than SQL. Nevertheless, by focusing on a specific work task (classification) the systems are comparable.

This chapter is structured as follows. Section 6.1 provides pointers to the relevant

literature most closely related to this work. Section 6.2 provides a brief description of the rule-based language. In Section 6.3 we describe the setup for our experiment. Section 6.4 provides the results from the usability experiments. Lastly, in Section 6.5 we provide directions for future research activities and summarize our findings.

Related Work

There have been numerous examples in the literature related to usability of query languages. The approaches tend to follow a fairly standard experimental design, in which two query languages are compared. Early work by Reisner [RBC75] forms the basis for many other experiments. Surveys that provide a compilation of multiple experiments were provided by Reisner [Rei81] and Welty [WS81]. A study by Yen and Scamell [YS93] compared SQL to Query By Example (QBE) serves as the basis of our experiment design. More recently, comparisons of diagrammatic languages to SQL have been presented, including work by Catarci, et. al. [Cat00].

6.2 The Rule-Based Language

In this section we provide a brief reprise of the mapping language, focusing its application on classification as the the user's goal. Conceptually, we view the process of classifying data as shown in Figure 6.1. The class values that are assigned can be

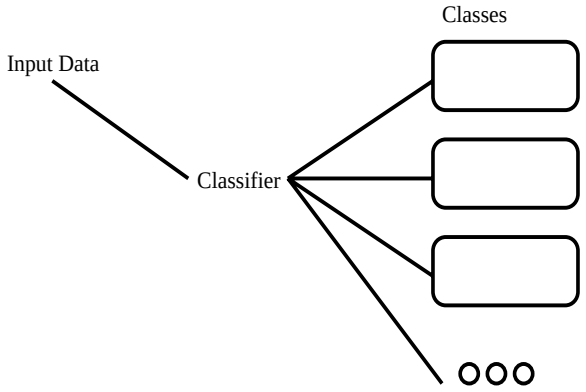


Figure 6.1: The classification process

expressed with database queries. We will continue with the employee salary example for the purposes of describing our approach.

The mapping language specifies queries in the form of a finite list of rules. Each rule is of the form $Head \leftarrow Body$, with the following semantics. The body of a rule is a boolean expression involving attribute names and constants, as well as a small number of supported functions. For example, basic mathematical functions such as addition and subtraction are supported. The head of a rule is an expression like the body, except that it is not restricted to a $\{0, 1\}$ result. Each rule head in a query must of the same data type. We refer to the set of attributes used in the rules as the schema of the query.

Let $P = \langle p_1, \ldots, p_k \rangle$ be a rule query and $\mathbf{r}$ be a table. For each $t \in \mathbf{r}$, let $Head(t)$ be the value of the expression $Head$ when given t as input. $Body(t)$ is defined in the

same manner. The output schema of P is $R \cup Class$, where R is the schema of the input table and $Class$ is the class value assigned to each tuple.[1] The following rule query defines the employee salary classification.

$'Low' \leftarrow Salary < 30000$

$'Medium' \leftarrow Salary >= 30000$ and $Salary <= 60000$

$'High' \leftarrow Salary > 60000$

Rules are evaluated in the order that they are defined. In addition, the language provides two different interpretation strategies. Queries may be interpreted *functionally* or *relationally*. Under a functional interpretation, rules are evaluated until a success is encountered. In contrast, a relational interpretation will evaluate every rule, which provides for a single input tuple to be mapped to multiple classes.

MQL provides two special types of rules to provide greater control over the process. A rule of the form $Except \leftarrow Body$ excludes input tuples from further processing. A rule of the form $Head \leftarrow Else$ allows for a tuple to be mapped if all of the preceding rules failed.

By design, the rule-based query language is restricted in terms of the types of queries that can be expressed. While it is similar to Datalog in terms of its syntax,

[1] The implementation provides the user with a mechanism for defining a unique name for $Class$.

we do not support recursive queries or negation in the head of the rule. With these restrictions, the queries can be executed in linear time by processing one tuple at a time.

Equivalence to SQL

Each rule query is equivalent to an SQL query involving set operations. The proof of this equivalence is based on a transformation from the rule-based language to SQL. Rather than providing the proof we provide a sketch of the transformation process here. First, note that the body of a rule is essentially the same as the SQL where clause. The head of a rule is the same as the SQL select clause. The following SQL query is equivalent to the rule query that classifies employee salaries.

```
Select "Low", * from employee where salary < 30000
  Union
Select "Medium", * from employee where salary >= 30000 and salary <=60000
  Union
Select "High", * from employee where salary > 60000
```

Special care must be taken to support the *Except* and *Else* rules by creating more complicated where clauses in the SQL queries. From a processing perspective, it is likely that SQL queries used for classification tasks will be less efficient than the

rule-based process. Note that the SQL queries may require multiple passes through the input data. In addition, using SQL in support of classification tasks may require complex queries to be written, which may be beyond of the end user's ability.

6.3 Experiment Design

In order to understand the usefulness of our approach we have performed a controlled usability experiment. This experiment seeks to quantify a user's ability to solve classification tasks with either SQL or the rule-based language. The focus is targeted only at classification tasks, and is not a broad comparison of fully equivalent systems. SQL was chosen as the most appropriate comparative tool due to its predominant use by both expert and novice users.

Independent Variables

The independent variables used to control the experiment were:

1. User skill level (Low, Medium, High)

2. Query language (SQL, Mapping Language)

3. Query complexity (Less complex, More complex)

Subjects for the experiment were recruited from a one-semester course in database systems as well as professional programmers. The Table 6.1 describes how user skill levels were assigned as well as the number of subjects in each skill level.

Subject Description	Skill Level	Number
Undergraduate Student	Low	27
Graduate Student	Medium	28
Professional experience	High	10

Table 6.1: Breakdown of subjects used in the experiment.

Half of the subjects performed classification tasks using either SQL or the mapping language. The use of SQL and the mapping language was balanced within the groups. The same problems were attempted by each user.

For the professional programmers, an additional within-groups experiment was performed. After completing either the mapping language or SQL problems the professionals performed the same tasks in the alternate language, which introduced an order of exposure variable. After completing both sets of problems the professional programmers completed an qualitative evaluation of both approaches.

Query complexity was based on the type of classification being performed. The complexity of the task was calculated based on a model answer for the problem as formulated in the rule-based language. A complexity score for a rule is defined according to the combined complexity of the head and the body, each of which is given a score of 1 (simple) or 2 (complex). An expression is simple if it involves

only constants or simple boolean comparisons, otherwise it is complex. The total complexity score for a task is given by summing the score for each rule. The score for our running example would be 6 - each rule involves only simple expressions and there are 3 rules.

Tasks below the mean score for all tasks are considered less complex, while tasks that score above the mean are considered more complex. It is important to note that the queries required to solve most of the tasks in this experiment would be classified as complex in the previously reported experiments due to their use of set operations.

Environment and Evaluation

While we have a fully functioning implementation of the rule-based language, we decided to administer the experiment by using a paper and pencil exam. This technique has been frequently employed in previous experiments, and benefits from being efficiently administered to multiple subjects simultaneously. Furthermore, a paper exam insulates the subjects from details of the system implementation, which provides for closer scrutiny of the language. Prior to participating in the exam, each subject filled out a short demographic questionnaire, which identified their skill level.

The exam was comprised of twelve classification problems against three different datasets. The datasets were described in terms of the input schema of the table. The problems were worded in the form of English sentences describing the desired

classification. A set of training materials and the exam are provided as an appendix to this dissertation.

Each subject was provided a training manual for the tool they were to use to solve the classification tasks. Each user had some experience with writing SQL queries, so the SQL training material focused on the writing of queries involving set operations. The rule-based language training materials were slightly longer due to syntax differences. Both training manuals contained an identical set of example classification tasks as well as solutions.

After reviewing the training materials the users were asked to complete a survey about the training materials and their impression of the proposed approach (SQL or the mapping language). They then proceeded to attempt solving the twelve classification problems. After solving the problems a post-exam survey was completed by each subject. The flow of the experiment is shown in Figure 6.2. The professional programmers evaluated both languages. They are represented in the figure by the lines connecting the Post-Exam Survey to the training module for the other language.

The examinations were administered in one sitting, limited to 45 minutes. Consequently, we expect that many of the problems will be unanswered. The mapping language was an entirely new technique for the subjects, which may yield even lower scores due to its novelty.

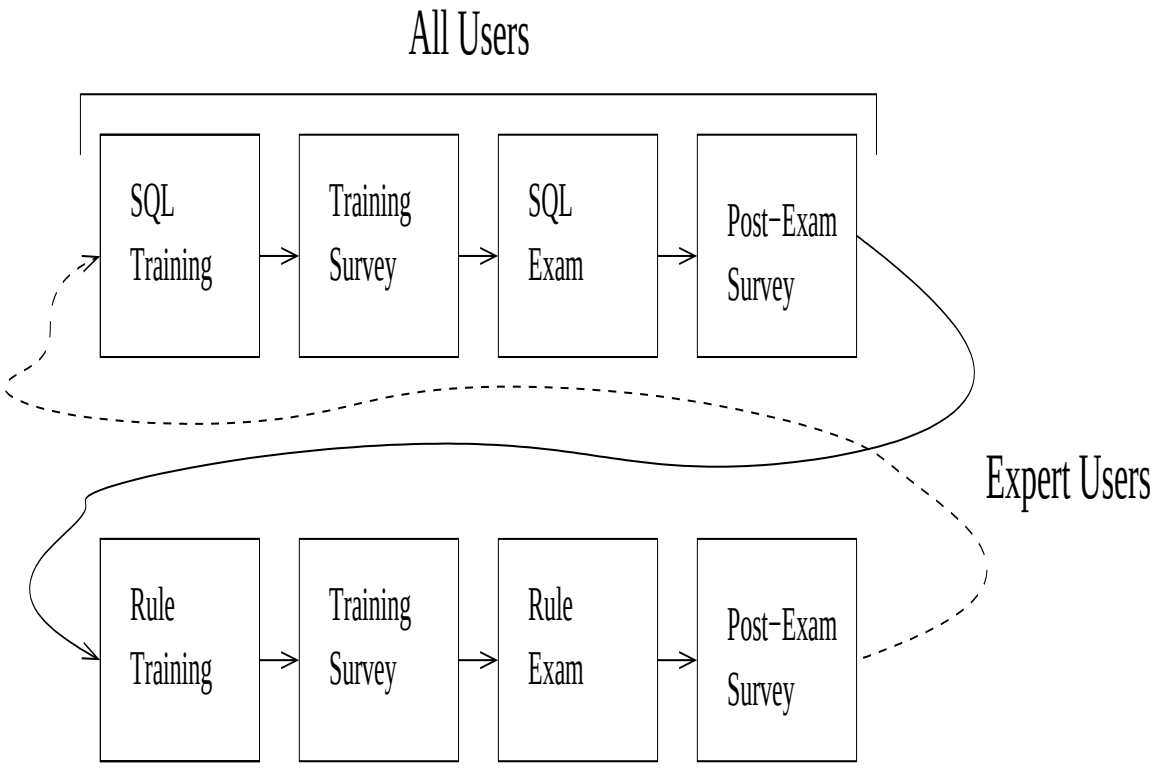


Figure 6.2: An overview of the design of the usability experiment.

Subject Group Comparison

Because of the design of the experiment, there are several groups whose characteristics need to be considered. Table 6.2 provides a summary of the information provided by the subjects that reported GPA's. A test for homogeneity of variances showed that the subject groups were comparable ($p < .10$).

The professional programmers were not asked to provide GPA's. Instead, they were asked to report their work experience (in years) with databases. As a group,

Subject Description	Average GPA (SD)	Average Experience (SD)
Undergraduate Student	3.35 (0.47)	1.18 (0.62)
Graduate Student	3.64 (0.31)	1.00 (0.85)

Table 6.2: The average GPA and number of database courses for the student subjects.

the professional programmers averaged 4.2 years of experience.

Dependent Variables

The dependent variables we measured with this experiment were:

1. Pre-Exam User Satisfaction

2. User Accuracy

3. Post-Exam User Satisfaction

For the professional programmers, we measured their satisfaction and accuracy for both languages. In addition, an overall satisfaction rating, comparing both approaches was measured for the professionals.

Accuracy is a quantitative measurement of the user's ability to solve classification problems with a specific tool. Satisfaction is a qualitative measurement of the user's feelings towards using a specific tool to solve classification problems. We did

not measure the time required to solve each problem for two reasons. First, measuring time would have required the use of either an SQL interface as well as the implemented rule-based system. The queries are more efficiently processed using the rule-based system, which we believe would unduly influence the satisfaction measurement. Second, interacting with either system introduces possible side-effects to the accuracy measurements. For example, subjects may fail to get the syntax of the query exactly correct and become frustrated with either system leading to fewer problems attempted.

We determined the accuracy of a user's solution using a technique similar to [YS93]. Each solution was assigned the lowest of the following possible scores as shown in Table 6.3.

Score	Description
3	Correct solution
2	Essentially correct solution (typographical errors)
1	Partially correct solution (missing conditions)
0	Incorrect solution, or unsolved

Table 6.3: Scoring guidelines used in the experiment.

The scoring method is intentionally coarse. Each subject's total score was computed by totaling their score for each problem and dividing by the maximum number of possible points.

User satisfaction was determined by using a qualitative assessment survey. Two surveys were completed by each student subject. The first survey was completed after reviewing the training material, which assesses the subject's feelings about the technique described in the training, prior to actually attempting to use the technique. The second survey was completed after completing the exam problems. The professional programmers completed two surveys for each language, as well as an subjective survey comparing both languages.

The relationship between the satisfaction scores for each subject is interesting, in that it allows us to determine whether the technique met their expectation. For example, a lower post-exam satisfaction score may indicate that the technique is more difficult than originally perceived. On the other hand, a higher post-exam satisfaction score would indicate that the technique was even better than the subject initially felt.

Hypotheses

The following list provides the hypotheses we seek to test with this experiment. By convention, each hypothesis is stated in its negative form.

H1: There will be no difference in accuracy based on tool selection.

H2: There will be no difference in accuracy based on user skill level.

H3: There will be no difference in accuracy based on tool and task complexity level.

H4: There will be no difference in satisfaction based on tool selection.

H5: There will be no difference in satisfaction based on user skill level.

H6: For professional programmers, there will be no interaction between the order of exposure and accuracy.

H7: For professional programmers, there will be no interaction between the order of exposure and satisfaction.

H8: For professional programmers, there will be no difference in tool preference.

Prior to executing the experiment we expect that most of the hypotheses will not be rejected. Nevertheless, we do expect to gain insight from the experiments, which will lead to further experimentation, and refinements to the design of the mapping language.

6.4 Results

In this section we report the results of the experiment. In the following subsections we report:

1. Efficiency
2. Accuracy

3. Satisfaction

4. Professional programmers

The statistical test employed for this analysis was the standard T-Test. The significance level employed for all tests was 0.10. Note that the risk associated with making a type I error with either approach is small.

Efficiency

Often, usability experiments of this type will measure the efficiency with which users can solve problems. Efficiency may be measured in terms of time; however, for this experiment time was a constraint placed upon the user. Consequently, the time involved in solving problems would have a more significant effect on effectiveness measures, since the lack of time to solve all of the problems would preclude users from fully completing the problems. This was certainly the case with the student subjects, with only two subjects providing solutions for every problem.

A different measure of effectiveness which can be reported for this experiment is the number of steps involved in solving problems with each system. For example, a smaller number of steps with one system may indicate an improvement efficiency. The problems given to the users in this experiment involved similar steps with either system:

1. Identify the classes to be generated.

2. Define a condition for each class.

3. Define the output value to be generated for each condition.

4. Construct a simple query in SQL or the Rule language.

5. Put the queries or rules in the correct order.

Syntactically, the solutions had the same number of rules or queries. For this experiment, then, the systems are not separable. In the future, more elaborate experiments could be constructed to more closely measure the cognitive impact of these languages.

It is clear that the rule language is less verbose than SQL, which allows for a comparison on this basis. For example, the following rule program classifies patients according to their age:

$'Newborns' \leftarrow Age < 3$

$'Children' \leftarrow Age \geq 3$ AND $Age \leq 12$

$'Adolescents' \leftarrow Age \geq 13$ AND $Age \leq 18$

$'Adults' \leftarrow Age \geq 19$ AND $Age \leq 65$

$'Elderly' \leftarrow Age > 65$

The equivalent SQL query is much more verbose:

```
Select "Newborns", * from Patient where Age < 3
  Union
Select "Children", * from Patient where Age >= 3 and Age <= 12
  Union
Select "Adolescents", * from Patient where Age >= 13 and Age <= 18
  Union
Select "Adults", * from Patient where Age >= 19 and Age <= 65
  Union
Select "Elderly", * from Patient where Age > 65
```

A word count (each term) of the rule language yields 37 words, while the SQL query has 61 words. Using this measure, the SQL queries were all longer than the rule programs. The average word count for the rule programs was 29.2 words, while the average for the SQL queries was 44.8 words.

Even though the rule language has the propensity to be more efficient using this measure, it is difficult to draw any direct conclusions. It is more likely that the experiment has revealed this in an indirect way. For instance, it is plausible that the lower satisfaction scores for SQL are related to the length of the solutions.

Accuracy

Table 6.4 shows the accuracy scores for each subject group.

Subject	SQL	Mapping Language
Undergraduate Student	27 (17)	37 (10)
Graduate Student	26 (19)	24 (21)
Professional	97 (0.1)	96 (0.2)

Table 6.4: Mean accuracy scores for each group, as a percent of total. (standard deviation)

The low scores for the students is related to the limited time that was provided for the exam. The difference in performance between the undergraduate and graduate student subjects is interesting. Note that the performance of graduate and undergraduate students is indistinguishable when using SQL. However, when using the mapping language, the undergraduate students performed better.

The lack of a similar relationship between accuracy scores for graduate students is a result of 4 students that did not get any problem correct. When omitting these students the average accuracy score for graduate students increases to 34% (SD 17), which tracks more accurately with the undergraduate result. The best explanation we have is based on the mix of students in the graduate class, which has a higher number of international students. It is possible that a language barrier was the primary influencer of the lower scores for these students. Since we did not control for this variable, we retained these students scores, rather than removing them from our

result.

Table 6.5 reports the statistical tests for Hypothesis 1 - 3. Each result was tested at a 0.10 significance level. When the hypothesis is rejected, we report the lowest significance level, even though our a priori test was at 0.10. For the professional programmers we report both parts of the experiment: Prof-1 refers to the first exam, Prof-2 refers to the second exam. For Hypothesis 3, we tested both SQL and MQL.

Hypothesis	t (critical value)	Result (p)
H1	-0.56 ($\pm$1.665)	Not Rejected ($p > .10$)
H2: Undergrad	-1.86 ($\pm$1.708)	Rejected ($p < .10$)
H2: Grad	0.29 ($\pm$1.706)	Not Rejected ($p > .10$)
H2: Prof - 1	0.44 ($\pm$1.86)	Not Rejected ($p > .10$)
H2: Prof - 2	0.26 ($\pm$1.86)	Not Rejected ($p > .10$)
H3: SQL	2.4 ($\pm$1.665)	Rejected ($p < .10$)
H3: MQL	1.82 ($\pm$1.665)	Rejected ($p < .10$)

Table 6.5: Statistical tests of the accuracy results (Hypothesis 1 - 3).

The undergraduate students accuracy scores were higher for the mapping language. However, the trend on accuracy as experience increases indicates that either tool can be used to solve such classification problems. This is a positive result for the mapping language, since the subjects had no prior knowledge of the language. Future work certainly needs to consider the effect of experience with the mapping language.

The results for Hypothesis 3 show that complexity of the solution does impact the accuracy. However, the actual result is counterintuitive - subjects were more accurate

with complex solutions. This was especially true with SQL. In retrospect, controlling for complexity in the way we did is probably faulty. Note that the more complex solutions were shorter, which most likely skewed the results.

Satisfaction

Table 6.6 shows the results of the satisfaction surveys for the SQL group. Table 6.7 shows the results of the satisfaction surveys for the Mapping Language group.

Subject	Pre-Exam	Post-Exam
Undergraduate Student	1.82 (0.65)	2.45 (0.67)
Graduate Student	1.92 (0.80)	2.48 (0.81)
Professional	1.90 (0.78)	2.62 (0.56)

Table 6.6: Mean satisfaction scores (1=Best, ..., 5=Worst) for the SQL group. (standard deviation)

Subject	Pre-Exam	Post-Exam
Undergraduate Student	2.09 (0.78)	2.01 (0.92)
Graduate Student	1.83 (0.57)	2.42 (0.58)
Professional	2.10 (0.60)	2.03 (0.68)

Table 6.7: Mean satisfaction scores (1=Best, ..., 5=Worst) for the Mapping Language group. (standard deviation)

What is interesting about the satisfaction scores is the relationship between the pre-exam and post-exam scores. For the SQL group, the post-exam score is higher. Again, the undergraduate students are interesting, in that their post-exam score is

actually lower than the pre-exam score. Omitting the same 4 students as we previously omitted still resulted in a higher post-exam satisfaction score for the graduate students, although not as high as the SQL, graduate student group.

We compared the satisfaction scores to a target satisfaction score of 2.0, which would indicate that the user subjectively believes that the tool is a "good" tool to use. The statistical test employed was a standard T-Test with degrees of freedom set to the size of the sample minus 1. As a group, the SQL group's post-exam satisfaction scores indicated that SQL was not as good as they initially believed. For the Mapping Language group, only the Graduate students exhibited the same behavior.

Table 6.8 reports the statistical results for Hypothesis 4 and 5. Again, we distinguish between the professional programmer's first and second exams.

Hypothesis	t (critical value)	Result (p)
H4	1.69 ($\pm$1.665)	Rejected ($p < .10$)
H5: Undergrad	-1.45 ($\pm$1.708)	Not Rejected ($p > .10$)
H5: Grad	0.22 ($\pm$1.706)	Not Rejected ($p > .10$)
H5: Prof - 1	1.53 ($\pm$1.86)	Not Rejected ($p > .10$)
H5: Prof - 2	-0.86 ($\pm$1.86)	Not Rejected ($p > .10$)

Table 6.8: Statistical tests of the satisfaction results (Hypothesis 4 and 5).

Hypothesis 4 was rejected. Note that this result compares the mean satisfaction score to 2.0 for the post-exam satisfaction survey. This result is influenced by the decreased satisfaction in SQL. For Hypothesis 5 the results do not consider the tool.

Rather, they simply indicate that either tool is subjectively considered to be a good tool to solve classification problems.

Details of the Professional Programmer Experiment

In this subsection we report the results of statistical testing of Hypothesis 6 - 8. These hypotheses consider whether there is any difference in accuracy and satisfaction for the professional programmers when given both tools to solve problems. Table 6.9 reports the statistical results for these hypotheses.

Hypothesis	t (critical value)	Result (p)
H6	-0.01 ($\pm$1.86)	Not Rejected ($p > .10$)
H7	-1.68 ($\pm$1.86)	Not Rejected ($p > .10$)
H8	6.11 ($\pm$1.833)	Rejected ($p < .10$)

Table 6.9: Statistical tests of the satisfaction results (Hypothesis 4 and 5).

Professional programmers had no problem with solving the problems with either tool, so there is no surprise about the accuracy hypothesis (H6). The satisfaction hypothesis (H7) is close to rejection. Again, the trend indicates that the subjects were less satisfied with SQL.

The significant result from Hypothesis 8 is based on a preference survey administered after using both tools. The raw data was scored on a 1 to 5 scale:

1: Strong preference for SQL.

2: Preference for SQL.

3: No preference.

4: Preference for MQL.

5: Strong preference for MQL.

The professional programmers preferred the mapping language, with a mean preference score of 3.7 (sd 0.35). The mean score indicates a somewhat weak preference for the mapping language. However, for the type of problem (classifications) no subject had a preference for SQL. This is especially positive for MQL, since the subjects had no prior knowledge of the language.

6.5 Summary

The usability experiment showed that the mapping language could be used by both experienced and lesser experienced users with about as much accuracy as SQL. This in itself is encouraging, since the mapping language was a new concept for the subjects.

The subjects were satisfied with both SQL and the mapping language. In general, the subjects were more satisfied with the mapping language.

We took advantage of the time the professional programmers made available for the study. Ideally, we would have liked many more professionals to participate in the study. The professionals had a strong preference for the mapping language for all types of classification tasks. At the same time, they tended to like SQL, since they had much more experience with it. The professionals were able to envision the benefits of the mapping language, which translated into the higher preference scores.

7

Visualizing Database Structure

This chapter explores the use of entropy for visualizing database structure. In particular, we show how visualizing the entropy of a relation provides a global perspective on the distribution of values and helps to identify areas within the relation where interesting relationships may be discovered. The type of structure we are interested in discovering is related to functional dependencies. Our approach is not dependent on the underlying domain of the data, providing a view of the dependency landscape within a relation. Using these techniques we described comparative results for a wide variety of synthetic and real data.

7.1 Introduction

Developing visualizations of database content is of extreme interest to the research community. There are numerous examples of applications developed around a graphical display of database content, including relationship discovery, outlier detection, and trend analysis.

Our approach to visualizing the structural, information content (which is distinct from the data content) of databases is motivated by the discipline of information theory. In particular, we utilize entropy as the basis for our visualizations. Within the field of information theory, entropy is the central concept, related to the encoding of messages. As such, entropy is a statistic that provides a global description of the information content of data. This research utilizes entropy to visualize the structure of a database relation, independent of the underlying domain datatype. We defer to Section 7.2 for the definition of entropy, as well as other formal notions.

When developing effective visualizations of database content there are three significant challenges that must be addressed. First, we are often faced with high-dimensional data. Second, databases are awash with categorical data, which often lack any meaningful order or scale. Thirdly, the sheer mass of data in even a moderately sized database may overwhelm the user when trying to simply display a two-dimensional scatter plot.

There has, of course, been a great deal of research effort aimed at addressing these challenges. For example, a number of techniques have been applied to high dimensional data, such as parallel coordinate displays, worlds within worlds, dimensional stacking, and grand tour methods.[ID87, ID90, LWW90, Fei92] Likewise, the field of information visualization has many techniques for dealing with abstract, categorical data.[CMS99] For techniques dealing with visualizing the contents of large databases see [KKS94, Kei96a, Kei96b].

Whereas the goal of almost all other visualization techniques is to facilitate understanding of particular values, our approach specifically remains aloof from those values in two distinct ways. First, we are interested in large-scale properties of instances – properties that are related to attributes rather than values. For example, a relation with 10 attributes requires 45 different 2D scatter plots to provide the same amount of information our technique provides in a *single* 2D scatter plot. Indeed, the axes for many of the visualizations we present are the collection of attributes in the relation, or characteristics of those attributes, rather than the values in a particular attribute. Second, even within a particular attribute, it is the distribution of values rather than the actual presented values that matter. Indeed, we typically code values as integers to simplify processing.

This second distinction is a consequence of the notion that the structure of data is independent of data values. Formally, structure is *generic*, in that it is invariant

under 1-1 substitution of data values (hence the encoding with integers). For example, functional dependencies describe a particular type of structure - independent of the actual values. The definition of the functional dependency $X \rightarrow Y$, namely "$\forall t_1 \in r, \forall t_2 \in r(t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y])$", exhibits this formal genericity. Information theory, which uses only probabilities associated with values and not the values themselves, therefore provides techniques that allow us to view value-independent structure. Because entropy-based visualizations show global properties, they are more in tune with natural uses of visualization, where global structure and detail through drill-down are most effective.

Naive attempts at global visualization often fail when our perceptual mechanisms confront the above challenges. For example, Figure 7.1 illustrates how our natural perception is sometimes misleading - a glance suggests that the left pane is more "function-like", while in fact the right pane exhibits the functional dependency $A \rightarrow B$ while the left does not. While the presence or absence of these functional dependencies is easy to evaluate in Figure 7.1, with only eight data points, this task becomes increasingly difficult as the number of data points increases. In addition, determining when the data contains an approximate dependency [KM92], in which a functional dependency holds except for a small number of violations, is equally difficult.

This paper is structured as follows. Section 7.2 provides the formal notation and definitions used throughout the paper. Section 7.3 explores the use of entropy to

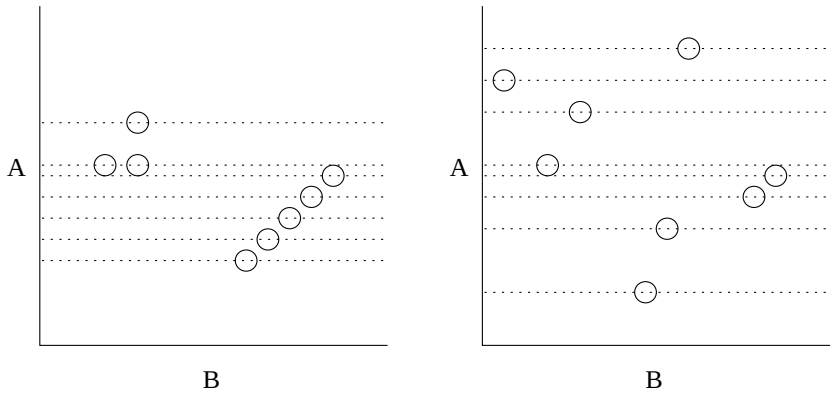


Figure 7.1: On the left, a scatter plot of data where the functional dependency $A \rightarrow B$ does not hold. On the right, a scatter plot where the functional dependency does hold. The dashed lines have been added to show the alignment of the points along the vertical dimension.

visualize frequency distributions of attribute values in database relations. Section 7.4 demonstrates the use of entropy for visualizing dependencies, or relationships between attributes. Section 7.5 provides examples of these these techniques for making broad comparisons of different datasets. Lastly, Section 7.6 provides future directions and concluding remarks.

7.2 Definitions

This section provides the formal notation and definitions used in this chapter. Our focus is on visualizing database information, so we begin with the basics of the relational model. Let $R = \{A, B, C, \ldots\}$ be a relation schema for instance $\mathbf{r}$. For

attribute $A \in R$, A denotes $\{A\}$. Sets of attributes are denoted by $X, Y, Z \subseteq R$. For X and $Y \subseteq R$, XY denotes $X \cup Y$. The notation we use for tuples is $t \in \mathbf{r}$, with $t.A$ representing the value for attribute A in tuple t.

With its genesis in message theory, entropy is defined over a set of messages $M = \{m_1, \ldots, m_n\}$, with associated probabilities $P_M = \{p_1, \ldots, p_n\}$. The entropy of M is $\mathcal{H}_M = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$. Entropy provides us with an average cost (in bits) for each message. The upper bound on the entropy of M is $\log n$, which occurs when each message has equal probability. Additional details on entropy, as well as other information theory topics, are covered in [CT91].

For databases, the message set we are interested in is taken from the relation instance $\mathbf{r}$. When projecting attributes from R, we do not eliminate duplicate values, which allows us to compute the probabilities using the counts of each value in the active domain. For example, the probability $P(A = a) = \frac{count(\sigma_{A=a}(\mathbf{r}))}{count(\mathbf{r})}$.

For any set of attributes $X \subseteq R$, we can compute $\mathcal{H}_X$ using an SQL aggregate query. The query is shown in Figure 7.2 for the case of $A \in R$. For the purposes of the visualizations generated for this research, we pre-compute $\mathcal{H}_A$ for each $A \in R$, as well as $\mathcal{H}_{AB}$ for each $A, B \in R$. Note that $\mathcal{H}_A \leq \log |\mathrm{Adom}(A)|$, where $\mathrm{Adom}(A)$ is the active domain of A. When $\mathcal{H}_A = \log |\mathbf{r}|$, A is a key.

```
Select   SUM((R1.frequency/R2.rowcount) *
             LOG(2,1/(R1.frequency/R2.rowcount)))
From    (Select    A, COUNT(*) as frequency
         From      R
         Group By  A) as R1,
        (Select    COUNT(*) as rowcount
         From      R) as R2
```

Figure 7.2: SQL query to calculate the entropy of A

Information Dependencies

Within the database research field, the concept of functional dependencies is well understood. The functional dependency $A \rightarrow B$ holds in instance $\mathbf{r}$, when for any two tuples $t_1, t_2 \in \mathbf{r}, t_1.A = t_2.A \implies t_1.B = t_2.B$. While functional dependencies always hold for an instance, a particular functional dependency may be specified as a constraint over instances of a relation and commercial database management systems have mechanisms for entering such constraints.

As is often the case, however, large, complex data rarely exhibits many functional dependencies beyond those specified as constraints. As shown in [DR00], an *Information Dependency Measure* is defined using entropy. The information dependency measure $\mathcal{H}_{X \rightarrow Y}$ provides a measure indicating the average number of bits we need to use to determine Y if we know a value for X. Another way to look at this measure is in terms of surprise. In other words, how surprising is a particular value for Y when we know X.

The information dependency $\mathcal{H}_{X \rightarrow Y}$ defined as $\mathcal{H}_{XY} - \mathcal{H}_X$. For more details on

information dependencies, as well as an equivalent definition, see [Dal00]. When $\mathcal{H}_{X\to Y} = 0$, the functional dependency $X \to Y$ holds. The upper bound on $\mathcal{H}_{X\to Y}$ is $\mathcal{H}_X + \mathcal{H}_Y$, which is the case of independence of X and Y.

Another weakness of using a traditional approach for identifying dependencies is shown in the right pane of Figure 7.1. We can verify by checking across the display that there are no violations of the dependency $A \to B$. However, as the number of datapoints increase the task becomes increasingly difficult. In addition, determining when the data contains an approximate dependency [KM92], in which a functional dependency holds except for a small number of violations, is equally difficult. Figure 7.1 may be used to illustrate the applicability if the information dependency measure: $\mathcal{H}_{A\to B}$ is 0.25 in the left pane and 0 in the right. Whereas a visual estimation of approximate functional dependencies does not scale, estimation via $\mathcal{H}_{A\to B}$ does.

7.3 Visualizing Distributions

Using the measures defined in Section 7.2, we turn to the visualization problem addressed in this research. The data we use in our visualization is drawn from a variety of sources, including the U.S. Census [cen], the U.C.I. Machine Learning Repository [BM98], and the Wisconsin Benchmark [DeW93]. The specific dataset we used for the Census was the 1990 Indiana Public Use Microdata Sample (PUMS), which has

125 attributes.

Our first application is the visualization of frequency distributions. An obvious technique for visualizing frequency distributions is to use histograms, with the height of each bar representing the frequency. Figure 7.3 shows the log of the size of the active domain for each attribute in the U.S. Census (Left) compared to the calculated entropy value for each attribute value (Right). The leftmost bar in each display corresponds to a key in the relation; otherwise, the attributes are arbitrarily ordered.

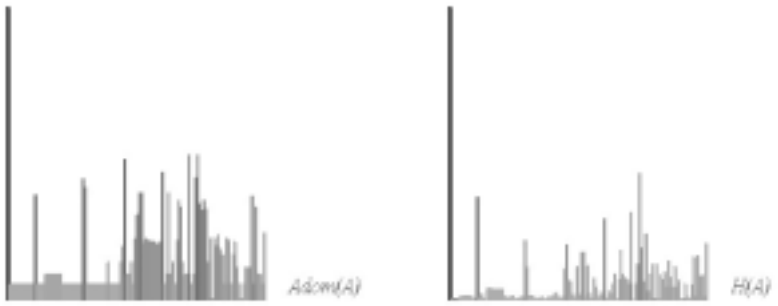


Figure 7.3: Comparing the size of the active domain for each attribute (Left) to the entropy of each attribute (Right) in the U.S. Census dataset.

Note that the height of the bars varies according to the probabilities associated with each value in the active domain, resulting in differences in the heights for the same attribute in each display. To highlight these differences, consider Figure 7.4, which shows the same information for a subset of the attribute space. The attributes displayed include: Hours Worked Per Week, Immigration Year, Income, Non-farm

Income, Farm Income, Interest and Dividend Income, Social Security Income, Public Assistance Income, and Retirement Income. In this case, we can see that certain attributes that have dominant values have their corresponding entropy values reflect this dominance.

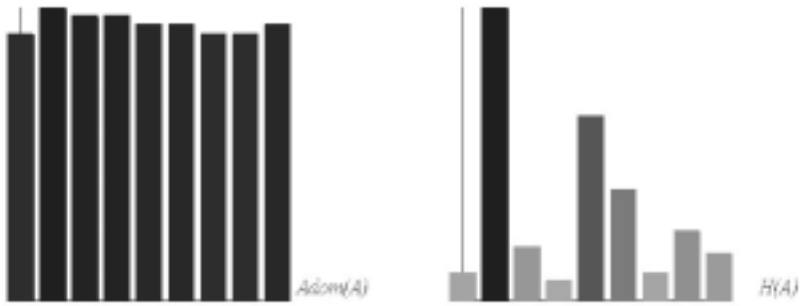


Figure 7.4: A view of the differences between the size of the active domain for (Left) compared to the entropy values for the same attributes (Right).

In order to gain an overall view of the attribute space, we can compare $\mathcal{H}_A$ to $\log |\text{adom}(A)|$ using a two-dimensional scatterplot. This visualization is shown in Figure 7.5, in which the attribute that is a key has been omitted. In the visualization, points that lie on the diagonal have an (approximately) uniform distribution. The further a point is from the diagonal, the less uniform is the associated distribution.

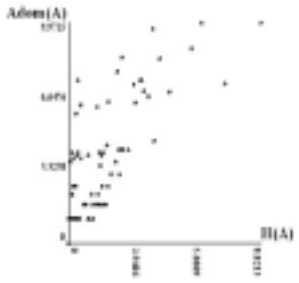


Figure 7.5: Comparing the entropy of each attribute in the census data to the log of the size of the corresponding active domain.

7.4 Visualizing Relationships

While the previous section demonstrated the use of entropy to gain insight into frequency distributions within database relations, this section extends the technique in order to explore relationships between attributes. In particular we utilize the information dependency measure described earlier to visualize these relationships.

While we have formally described the concept of an information dependency, we have not yet discussed visualizing them. The left pane of Figure 7.6 characterizes the space of $\mathcal{H}_{A\rightarrow B} \times \mathcal{H}_B$, which is encountered when visualizing the values in a 2D scatter plot. This type of visualization allows us to get an overall view of all possible attribute pairs in a compact space. A critical advantage of this approach is that the visualizations do not depend on the actual values or types of data.

The dark area in the figure represents functional dependencies in the relation.

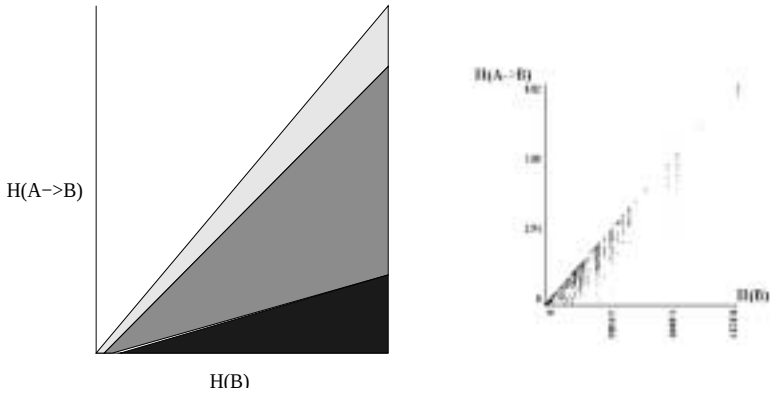


Figure 7.6: On the left, characterizing the space $\mathcal{H}_{A\to B} \times \mathcal{H}_A$. On the right, a visualization of this space for the census data.

Above the diagonal the space is empty, since the upper bound of $\mathcal{H}_{A\to B}$ is $\mathcal{H}_B$. As you move away and below the diagonal, the structure becomes more like a functional dependency. There is an area of potential interest close to the horizontal axis, in which the space represents approximate functional dependencies that are *almost* a pure functional dependency. The space closest to the diagonal contains attribute pairs where B does not significantly depend on A (we cannot say that A and B are independent since $B \to A$ is not ruled out).

The right pane of Figure 7.6 shows a scatter plot comparing $\mathcal{H}_{A\to B}$ to $\mathcal{H}_B$ for the census data. We can easily see individual attributes, which correspond to the vertical bands (since of course $\mathcal{H}_B$ is determined only by B. Unfortunately, a black-and-white rendition of this image does not indicate points that lie exactly on the horizontal axis - points that correspond to true functional dependencies. Certain points close to that

axis are of obvious interest. In addition, one isolated point about 1/3 up and 2/3 right begs investigation in detail. However, we cannot tell whether points on the upper right merely correspond to A's with small entropy.

This suggests a more detailed examination using three dimensions, comparing $\mathcal{H}_{A\rightarrow B}$, $\mathcal{H}_A$ and $\mathcal{H}_B$. Figure 7.7 shows two perspectives on this visualization - the left image looks out along the $\mathcal{H}_A$ axis with $\mathcal{H}_B$ vertical and $\mathcal{H}_{A\rightarrow B}$ going off to the right and the right image rotates the left around the vertical axis. Both images are zoomed somewhat, as is evident by the axis labels. The origin is zero for each axis. The fact that $\mathcal{H}_{A\rightarrow B} \leq \mathcal{H}_B$ is clearly shown by the empty space in the lower right of the image.

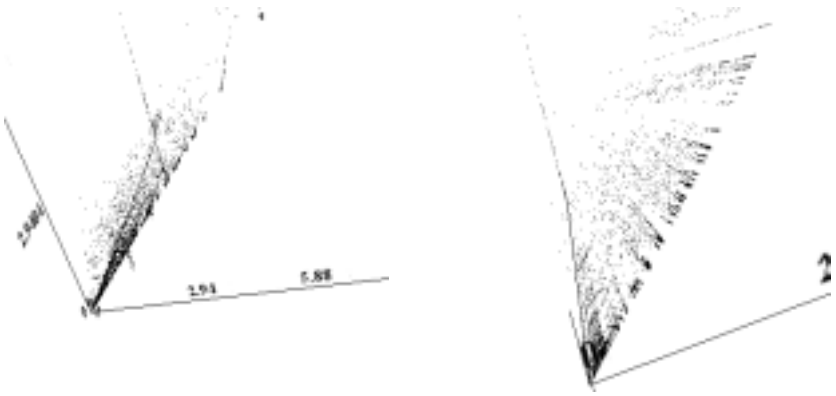


Figure 7.7: 3D plot comparing $\mathcal{H}_{A\rightarrow B}$, $\mathcal{H}_A$ and $\mathcal{H}_B$.

We can observe several properties of the space. The vertical line of data points at the far end of the $\mathcal{H}_A$ axis (Z dimension) arises when A corresponds to the key of the relation, which of course functionally determines every attribute. The rotation

from the left to right image also shows that most points lie near the $\mathcal{H}_{A\rightarrow B} = \mathcal{H}_B$ plane, althought this is less evident with still images than when image manipulation is possible.

Two bands along the edge of the $\mathcal{H}_B = \mathcal{H}_{A\rightarrow B}$ plane, corresponding roughly to $\mathcal{H}A \leq 0.5$ or $\mathcal{H}_B \leq 0.5$, line very close to the plane. This indicates the surprising fact that each low entropy attribute is independent of other attributes except each other

In addition to identifying potentially interesting relationships between attributes, the visualizations also highlight additional information. For example, when $\mathcal{H}_A$ is low and $\mathcal{H}_{AB} - \mathcal{H}_A = 0$, it is possible to decompose the original relation into smaller sub-relations, taking advantage of space savings. When the difference is very near to zero, you may decide to ignore the noise entirely and clean the data by removing the noisy data.

Drilling Down

The discussion thus far has involved global characterizations of attributes, but information-based visualization can also drill-down to reveal local structures. This makes use of the fact that the functional dependency $A \rightarrow B$ holds iff $\mathcal{H}_{A\rightarrow B} = 0$ and thus the quantity $\mathcal{H}_{A\rightarrow B}$ is a measure of how close $A \rightarrow B$ is to holding in an instance. The characterization of $\mathcal{H}_{A\rightarrow B}$ as $\sum_{a\in A} p(a) \times \mathcal{H}_B(\sigma_{A=a}(r))$ suggests that the "landscape" of $p(a)$ and $\mathcal{H}_B(\sigma_{A=a}(r))$ might reveal something about local

structure related to $A \rightarrow B$. Indeed this is the case, as we see in examples from the census data.

The first example examines `AGE` $\rightarrow$ `DEPART` (with `AGE` as A and `DEPART` as B). The plot of $p(a)$ versus $\mathcal{H}_B(\sigma_{A=a}(r))$, shown in the first panel of Figure 7.8, has several interesting features:

1. `AGE` values with low probability have low diversity of associated `DEPART`, and this holds uniformly

2. the relationship of $\mathcal{H}_B(\sigma_{A=a}(r))$ versus $p(a)$ is essentially a smooth function for low $p(a)$ values

3. when $p(a)$ exceeds a certain value, the corresponding $\mathcal{H}_B(\sigma_{A=a}(r))$ is typically close to the maximum; this cutoff is surprisingly sharp

4. there are a few higher probability `AGE`s which differ from the typical by having $\mathcal{H}_B(\sigma_{A=a}(r))$ values that are lower or 0; these `AGE`s are interesting in themselves. Indeed, further investigation of these values seems to indicate anomalies in the way the census data was collected.

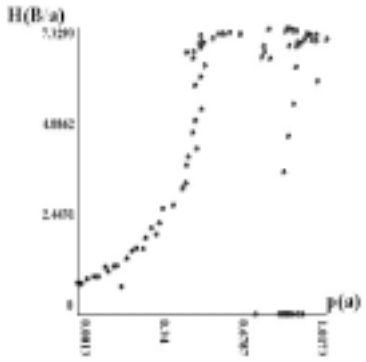


Figure 7.8: Comparing $p(a)$ to $\mathcal{H}_B(\sigma_{A=a}(r))$ for census data. In this example, A is `AGE` and B is `DEPART`.

7.5 Visual Comparisons Of Datasets

In previous sections we have demonstrated the use of entropy to visualize the information content of database relations. In this section we show how multiple, diverse datasets can be compared within the same display in order to understand the degree to which the datasets might be similar in terms of their structure.

We have used this particular technique to compare various benchmark datasets in order to evaluate their structure. Although benchmark datasets are used for a variety of applications, a primary use is the performance evaluation of new algorithms. For example, the Wisconsin benchmark [DeW93] has been used to test various join algorithms. Within the machine learning community a large number of benchmark datasets are available.[BM98] Many of these datasets have been used for evaluating various data mining techniques.

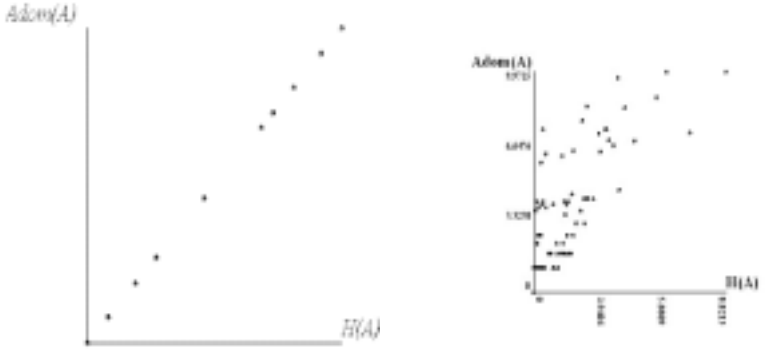


Figure 7.9: $\mathcal{H}_A$ compared to log |adom(A)| for the Wisconsin benchmark data (Left). The same comparison from the census data (Right).

Figure 7.9 (Left) shows $\mathcal{H}_A$ compared to log |adom(A)| for the Wisconsin benchmark data. The Wisconsin data can be seen to have a nearly perfect uniform distribution within each attribute. When compared on the census data, seen in Figure 7.9 (Right), it is clear that this synthetically generated data demonstrates significant differences from real data, which has much more complexity to its structure.

As another example, Figure 7.10 shows a number of datasets from the machine learning repository displayed for comparison. We can see in this visualization that these datasets have different structure as well, although the sparseness of the data does have an effect. In addition, these datasets tend to have a large number of boolean valued attributes.

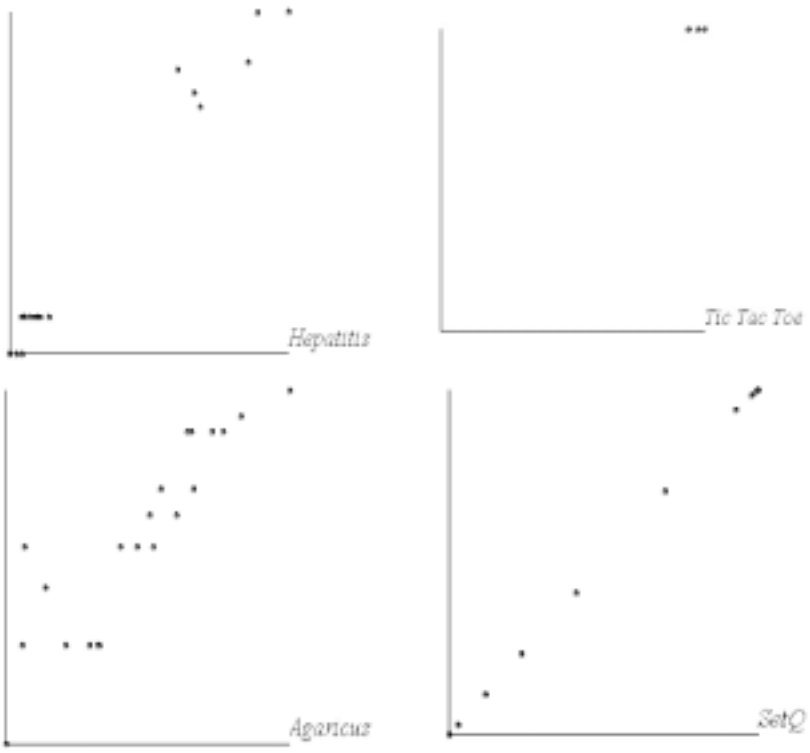


Figure 7.10: $\mathcal{H}_A$ compared to log $|\text{adom}(A)|$ for datasets taken from the machine learning repository. Clockwise from top left - Hepatitis, Tic Tac Toe, Agaricus, SetQ.

7.6 Conclusion

In this chapter we have shown how entropy, a central concept in information theory, can be used for visualizing the structure of information within database relations. The technique simplifies the display of complex relationships, allowing for dependencies to be spotted. Our use of entropy is independent of the underlying datatypes, handling all in a consistent fashion. Furthermore, we have demonstrated the technique on a wide variety of data, some of which are quite large. The census dataset,

for instance, contains 125 attributes and approximately 300,000 rows of data.

While this particular research is reported in terms of database visualization problems, the techniques we have employed are applicable to several areas. Within data mining we envision that these techniques can be used to assist an expert in exploring their particular problem space. In addition, database designers can use the visualization to assist in the construction of decompositions, either for OLTP systems, or for OLAP data warehouses.

8

Conclusion and Future Work

In this chapter we summarize the main findings presented in this research. In addition, we outline extensions to the architecture, mapping language and applications related to database visualization.

8.1 Contributions of This Research

Architecture

The architecture is the primary contribution of this research. By abstracting the concept of a map, we highlight the benefits of supporting visualization with database systems. In particular, maps can be constructed to support multiple visualizations of the same data without changing the original data. This abstraction is consistent with the traditional, layered approach embodied in the design of database systems.

The concept of a map application, which formally defines the input data in database terms, is one of the principle elements of the architecture. This simplified approach allowed for mapping from the data to the graphical output form in the same way that database queries are supported.

An implementation of the architecture serves as a proof-of-concept for our approach. In this system we provided a number of basic visualization formats, demonstrating the ability for the mapping formalism to be applied to both generate the graphics, as well as probing the visualization to retrieve the underlying data in a seamless fashion.

Mapping Language

The architecture provides the capability for users to generate a wide varety of visualizatins. However, when the data lacks an appropriate order and/or scale, it must be transformed prior to visualizing the data. Though there may be many ways in which these transformations can be implemented, this research presents a mapping language to accomplish this process.

The mapping language supports database visualization by providing an automated, controlled mechanism for defining the transformations. It can be used to convert categorical values into numeric, or for simplifying a more complex numeric scale. For example, continuous values can be binned using a simple calculation.

The language is equivalent to the relational algebra, restricted to select, project and union operations. At this time, the language does not support negation, except as a term within a conjunctive clause.

We also described different ways in which map programs could be interpreted: relational and functional. Under the control of the user, a tuple in the input can be mapped to either a single output tuple (functional) or multiple tuples (relational). Formally, we showed that functionally interpreted programs can be simulated with an equivalent relationally interpreted program. This equivalence yields a single, straightforward algorithm for processing either interpretation. Furthermore, map programs can be executed more efficiently than equivalent relational algebra programs, when RA programs are directly implemented in SQL.

Usability Evaluation

We focused the use of the mapping language on a specific user task in order to better understand the strengths and weaknesses of our approach. For comparison purposes, users solved problems using either the mapping language or SQL. The tasks were designed to solve classification problems.

The results of the experiments were instructive in a number of ways. First, users were more successful solving problem with the mapping language. While the difference in performance between the subjects was not significant when considering the

complete set of subjects, undergraduate students performed better with the mapping language than with SQL ($p < 0.10$).

Overall, the experiment showed that users were satisfied with the mapping language. The subjects' limited time to perform the experiment leads us to believe that with prolonged exposure to the language, the two approaches would be more clearly separated.

The professional programmers provided preference data, comparing the relative strengths of each approach. For this class of user, the mapping language was preferred for all tasks ($p < 0.001$). These results are quite encouraging for our proposed approach.

Visualizing Database Structure

We demonstrated an application of the architecture applied to a problem closely related to database systems. Expanding on previous work in [Dal00], we investigated methods for visualizing the structure of data in databases. In particular, we portrayed the strength of dependency between attribute pairs using scatterplots. This technique allowed for all of the dependencies to be viewed in one display.

We used the same technique as a way to compare multiple datasets. Using this

visualization technique we were able to easily discern the differences between synthetically generated datasets when compared to real data.

8.2 Future Work

To a great extent, the goal of this research has always been focused on enabling future capabilities. Indeed, the sole reason for proposing an architecture to address the problem of visualization is predicated on a requirement to flexibly support new applications. In this section we outline a few areas that are interesting extensions of this work.

Language Extensions

In its current design, the mapping language has a number of restrictions. The logic based language design has succinct methods for expressing recursive queries, as well as incorporating negation. Both of these areas would be interesting extensions to our approach. Supporting recursive queries would support the generation of visualizations of graph-based data, which has application in a number of areas. Negation, allows interesting views of what is missing from data that would currently be seen as gaps in a plot, for example.

Applications

We envision expanding the architecture by exploring targeted applications related to data mining and knowledge discovery. One area of particular interest is related to association rule management, in which the number of rules generated from a dataset exceeds the capabilities for users to understand and find the most interesting and important rules.

Usability of Visualizations

Very little work has been done in evaluating the effectiveness of visualization systems. Principally, most information visualization systems are constructed to support a specific problem. The specific designs yield systems that are not comparable from a traditional HCI perspective. At the heart of this problem is the need to develop a more comprehensive understanding of how users perceive graphic representations of data.

Bibliography

[ACS90a] M. Angelaccio, T. Catarci, and G. Santucci. QBD*: A fully visual query system. *Journal on Visual Languages and Computing*, 1(2):255–273, 1990.

[ACS90b] M. Angelaccio, T. Catarci, and G. Santucci. QBD*: A graphical query language with recursion. *IEEE Transactions on Software Engineering*, 16(10):1150–1163, 1990.

[ACSW96] Alexander Aiken, Jolly Chen, Michael Stonebraker, and Allison Woodruff. Tioga-2 : A direct manipulation database visualization environment. In *Proceedings of the 12th International Conference on Data Engineering, New Orleans, LA, February 1996*, pages 208–217, 1996.

[Ahl96] Christopher Ahlberg. Spotfire: An information exploration environment. *SIGMOD Record*, 25(4):25–29, 1996.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[ALS97] J. Allan, A. Leouski, and R. Swan. Interactive cluster visualization for information retrieval. Technical Report Technical Report IR-116, Center for Intelligent Information Retrieval, University of Massachusetts, Amherst, 1997.

[Ank00] Mihael Ankerst. *Visual Data Mining*. PhD thesis, University of Munich, 2000.

[AS92] C. Ahlberg and B. Shneiderman. Dynamic queries for information exploration: an implementation and evaluation. In *Proceedings of the ACM CHI'92 Conference*, pages 619–626, 1992.

[AS94a] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.

[AS94b] C. Ahlberg and B. Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfields displays. In *Proceedings of the ACM CHI'94 Conference*, pages 313–317, 1994.

[AW95] Christopher Ahlberg and Erik Wistrand. IVEE: An environment for automatic creation of dynamic queries applications. In *Human Factors in Computing Systems. Conference Proceedings CHI'95*, 1995.

[BA86] Andreas Buja and Daniel Asimov. Grand tour methods: An outline. In *Proceedings of the 18th Symposium on the Interface*, pages 63–67, 1986.

[BA96] Ronald Brachman and Tej Anand. The process of knowledge discovery in databases. In *Advances in Knowledge Discovery and Data Mining*, pages 37–57. 1996.

[Bay98] Roberto J. Bayardo Jr. Efficiently mining long patterns from databases. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 85–93. ACM Press, 1998.

[BC87] R. A. Becker and W. S. Cleveland. Brushing scatterplots. *Technometrics*, pages 127–142, 1987.

[BKK97] C. Brunk, J. Kelly, and R. Kohavi. Mineset: An integrated system for data access, visual data mining, and analytical data mining. In *Proceedings of KDD '97*, 1997.

[BM94] Nigel Bevan and Miles Macleod. Usability measurement in context. *Behaviour & Information Technology*, 13(1):132–145, 1994.

[BM98] C.L. Blake and C.J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 255–264. ACM Press, 1997.

[BSP+93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: The see-through interface. In *Proceedings of Siggraph*, Computer Graphics Annual Conference Series, pages 73–80. ACM, 1993.

[CAL+97] Isabel F. Cruz, Michael Averbuch, Wendy T. Lucas, Melissa Radzyminski, and Kirby Zhang. Delaunay: A database visualization system. In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 510–513. ACM Press, 1997.

[Cat00] Tiziana Catarci. What happened when database researchers met usability. *Information Systems*, 25(3):177–212, 2000.

[CC92] Matthew Chalmers and Paul Chitson. Bead: Explorations in information visualization. In *Research and Development in Information Retrieval*, pages 330–337, 1992.

[CCLB97] Tiziana Catarci, Maria Francesca Costabile, Stefano Levialdi, and Carlo Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.

[cen] www.census.gov. On The Web.

[Che75] Peter P. S. Chen. The entity-relationship model — toward a unified view of data. *Proceedings of the 1th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Kerr(ed), pp.173*, 1975.

[Chi00] Ed Chi. A taxonomy of visualization techniques using the data state reference model. In *Proceedings of InfoVis 2000 (Salt Lake City UT, October 2000)*, pages 69–75, 2000.

[CL00] Isabel F. Cruz and Peter S. Leveille. Implementation of a constraint-based visualization system. In *Proceedings of the 2000 IEEE International Symposium on Visual Languages*, 2000.

[CM97] S. Card and J. MacKinlay. The structure of the information visualization design space. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '97)*, pages 92–99, 1997.

[CMS99] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization:Using Vision to Think*. Morgan Kaufmann Publishers, Inc., 1999.

[Cod70] E. F. Codd. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Cru92] Isabel F. Cruz. DOODLE: a visual language for object-oriented databases. In *SIGMOD 1992, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 71–80, 1992.

[Cru93a] Isabel F. Cruz. Expressing constraints for data display specification: A visual approach. Technical Report CS-93-57, Brown University, Computer Science, 1993.

[Cru93b] Isabel F. Cruz. User-defined visual languages for querying data. Technical Report CS-93-58, Brown University, Computer Science, 1993.

[CS95] T. Catarci and G. Santucci. Textual query languages: A comparative experiment. In *IFIP 2.6: Third Working Conference on Visual Database Systems (VDB-3)*, pages 57–74, 1995.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, New York, NY, USA, 1991.

[Dal00] Mehmet M. Dalkilic. *Foundations of Data Mining.* PhD thesis, Indiana University, Computer Science, 2000.

[DeW93] David J. DeWitt. The wisconsin benchmark: Past, present, and future. In Jim Gray, editor, *The Benchmark Handbook for Database and Transaction Systems (2nd Edition).* Morgan Kaufmann, 1993.

[DR00] Mehmet M. Dalkilic and Edward L. Robertson. Information dependencies. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA*, pages 245–253. ACM, 2000.

[Fei92] Steven Feiner. Virtual worlds for visualizing information. In *Advanced Visual Interfaces*, pages 3–11, 1992.

[FMMT96] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Data mining using two-dimensional optimized accociation rules: Scheme, algorithms, and visualization. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 13–23. ACM Press, 1996.

[FPSS96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. 1996.

[GP88] Georges R. Grinstein and Ronald M. Pickett. Iconographic displays for visualizing multidemensional data. In *Proceedings IEEE Conference on Systems, Man, and Cybernetics*, pages 514–519, May 1988.

[GPW89] Georges R. Grinstein, Ronald M. Pickett, and Marian G. William. EXVIS: An exploratory visualization environment. In *Proceedings of Graphics Interface '89*, pages 254–261, 1989.

[GR98] Dennis P. Groth and Edward L. Robertson. Architectural support for database visualization. In *Proceedings of the Workshop on New Paradigms in Information Visualization and Manipulation*, 1998.

[GR02a] Dennis P. Groth and Edward L. Robertson. An entropy-based approach for visualizing database structure. In *Proceedings of the Sixth IFIP Conference on Visual Database Systems (VDB6)*, 2002.

[GR02b] Dennis P. Groth and Edward L. Robertson. An integrated approach to database visualization. In *Proceedings of the Conference on Advanced Visual Interfaces (AVI 2002)*, 2002.

[GR02c] Dennis P. Groth and Edward L. Robertson. An integrated system database visualization. In *Proceedings of the Sixth International Conference on Information Visualization (IV02)*, 2002.

[GSB91] Georges R. Grinstein, Stuart Smith, and R. Daniel Bergeron. Interactive data exploration with a supercomputer. In *Proceedings of IEEE Visualization '91*, Los Alamitos, CA, October 1991. IEEE Computer Society Press.

[HHW00] Arno P.J.M. Siebes Heike Hofmann and Adalbert F.X. Wilhelm. Visualizing association rules with interactive mosaic plots. In *SIGKDD 2000, Proceedings ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 20-23, 2000, Boston, MA,USA*, pages 227–235. ACM Press, 2000.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[HMS01] David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. The MIT Press, 2001.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In Weidong Chen, Jeffrey F. Naughton, and

Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, volume 29, pages 1–12. ACM, 2000.

[HYL00] Kent Wenger Hongyu Yao and Miron Livny. DEVise and the JavaScreen: Visualization on the web. In *Proceedings of the SPIE Conference on Visual Data Exploration and Analysis*, pages 375–384, January 2000.

[IBM] www.research.ibm.com/dx/. On the Web.

[ID87] Alfred Inselberg and Bernard Dimsdale. Parallel coordinates for visualizing multi-dimensional geometry. In *Proceedings of Computer Graphics International '87*, Tokyo, 1987. Springer-Verlag.

[ID90] Alfred Inselberg and Bernard Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *Proceedings of IEEE Visualization '90*, pages 361–375, Los Alamitos, CA, October 1990. IEEE Computer Society Press.

[IN88] Tomasz Imielinski and Shamim A. Naqvi. Explicit control of logic programs through rule algebra. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 21-23, 1988, Austin, Texas*, pages 103–116. ACM Press, 1988.

[Int89] International Organization for Standardization (ISO). Database Language SQL, 1989.

[Int92] International Organization for Standardization (ISO). Database Language SQL (SQL2), 1992.

[Int99] International Organization for Standardization (ISO). Database Language SQL (SQL3), 1999.

[KAK94] Daniel A. Keim, Mihael Ankerst, and Hans-Peter Kriegel. Visdb: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, 14(5):40–49, September 1994.

[Kea99] T. Alan Keahey. Visualization of high-dimensional clusters using nonlinear magnification. In *Available on the Web at http://www.acl.lanl.gov/Viz/abstracts/spie99_abstract.html*, 1999.

[Kei95] Daniel A. Keim. *Visual Support for Query Specification and Data Mining*. PhD thesis, Institute for Computer Science, Ludwig-Maximilians-University Munich, 1995.

[Kei96a] Daniel A. Keim. Databases and visualization. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, page 543. ACM Press, 1996.

[Kei96b] Daniel A. Keim. Pixel-oriented database visualizations. *SIGMOD Record*, 25(4):35–39, 1996.

[Kei97] D. Keim. Visual data mining. In *Proceedings of VLDB 1997*, 1997.

[KK95a] Daniel A. Keim and Hans-Peter Kriegel. Possibilities and limits in visualizing large databases. In *Visual Database Systems 3, Visual Information Management, Proceedings of the third IFIP 2.6 working conference on visual database systems*, pages 203–214, 1995.

[KK95b] Daniel A. Keim and Hans-Peter Kriegel. Recursive pattern: A technique for visualizing very large amounts of data. In *Proceedings of IEEE Visualization '95*, pages 279–286, Los Alamitos, CA, October 1995. IEEE Computer Society Press.

[KK95c] Daniel A. Keim and Hans-Peter Kriegel. Visdb: A system for visualizing large databases. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, page 482. ACM Press, 1995.

[KK96] Daniel A. Keim and Hans-Peter Kriegel. Visualization techniques for mining large databases: A comparison. *TKDE*, 8(6):923–938, 1996.

[KKS94] Daniel A. Keim, Hans-Peter Kriegel, and Thomas Seidl. Supporting data mining of large databases by visual feedback queries. In *Proceedings of the Tenth International Conference on Data Engineering, February 14-18, 1994, Houston, Texas, USA*, pages 302–313. IEEE Computer Society, 1994.

[KM92] Jyrki Kivinen and Heikki Mannila. Approximate dependency inference from relations. In Joachim Biskup and Richard Hull, editors, *Database Theory - ICDT'92, 4th International Conference, Berlin, Germany, October 14-16, 1992, Proceedings*, volume 646 of *Lecture Notes in Computer Science*, pages 86–98. Springer, 1992.

[Lee] J. Lee. Views, visualization and databases. In *Proceedings of the 2nd Workshop on Database Issues for Data Visualization, A. Wierse and G. G. Grinstein, eds., Springer Verlag Lecture Notes in Computer Science.*

[LGa] J. Lee and G. Grinstein. An architecture for retaining and analyzing visual explorations of databases. In *Proceedings of IEEE Visualization '95, Atlanta GA.*

[LGb] J. Lee and G. Grinstein. Describing visual interactions to the database: Closing the loop between user and data. In *Proceedings of SPIE, Visual Data Exploration and Analysis III, San Jose CA, volume 2565.*

[LG94] J. Lee and G. Grinstein. Data exploration interactions and the Exbase system. In J.P. Lee and G.G. Grinstein, editors, *Database Issues for Data Visualization*, volume 871 of *Lecture Notes in Computer Science*, pages 118–137. Springer Verlag, 1994.

[LRB+97] Miron Livny, Raghu Ramakrishnan, Kevin Beyer, Guangshun Chen, Donko Donjerkovic, Shilpa Lawande, Jussi Myllymaki, and Kent Wenger. Devise: Integrated querying and visual exploration of large datasets. In *Proceedings of ACM SIGMOD*, May 1997.

[LRM96] M. Livny, R. Ramakrishnan, and J. Myllymaki. Visual exploration of large data sets. In *Proc. of SPIE – Int. Soc. Opt. Eng.*, volume 2657, 1996.

[LWW90] Jeffrey LeBlanc, Matthew O. Ward, and Norman Wittels. Exploring n-dimensional databases. In *Proceedings of IEEE Visualization '90*, pages 230–237, Los Alamitos, CA, October 1990. IEEE Computer Society Press.

[mic] www.microsoft.com. On the Web.

[MR97] Dieter Merkl and Andreas Rauber. Alternative ways for cluster visualization in self-organizing maps. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6*, pages 106–111. Helsinki

University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

[NHM97] Gregory M. Nielson, Hans Hagen, and Heinrich Muller. *Scientific Visualization: Overviews, Methodologies and Techniques*. IEEE Computer Society, 1997.

[PdBA+92] Jan Paredaens, Jan Van den Bussche, Marc Andries, Marc Gemis, Marc Gyssens, Inge Thyssens, Dirk Van Gucht, Vijay Sarathy, and Lawrence V. Saxton. An overview of GOOD. *SIGMOD Record*, 21(1):25–31, 1992.

[PK94] A. Papantonakis and P.J.H. King. GQL, a declarative graphical query language based on the functional data model. In *Proceedings of the Workshop on Advanced Visual Interfaces*, June 1994.

[qua] www.quadbase.com. On the Web.

[RBC75] P. Reisner, R. Boyce, and D. Chamberlin. Human factors evaluation of two database query languages - square and sequel. In *Proceedings of the National Computer Conference*, pages 447–452, 1975.

[Rei81] Phyllis Reisner. Human factors studies of database query languages: A survey and assessment. *ACM Computing Surveys*, 13(1):13–31, 1981.

[SCN+93a] Michael Stonebraker, Jolly Chen, Nobuko Nathan, Caroline Paxson, Alan Su, and Jiang Wu. Tioga : A database oriented visualization tool. In *Proceedings of the IEEE '93 Visualization Conference, San Jose, CA, USA, October 1993*, pages 86–93, 1993.

[SCN+93b] Michael Stonebraker, Jolly Chen, Nobuko Nathan, Caroline Paxson, and Jiang Wu. Tioga: Providing data management support for scientific visualization applications. In Rakesh Agrawal, Seán Baker, and David A. Bell, editors, *19th International Conference on Very Large Data Bases, August 24-27, 1993, Dublin, Ireland, Proceedings*, pages 25–38. Morgan Kaufmann, 1993.

[Shn94] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11:70–77, November 1994.

[Shn96] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. IEEE Symp. Visual Languages, VL*, pages 336–343, 3–6 1996.

[Shn98] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison Wesley Longman, Inc., 3rd edition, 1998.

[SON95] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, pages 432–444. Morgan Kaufmann, 1995.

[STA98] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. Integrating mining with relational database systems: Alternatives and implications. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 343–354. ACM Press, 1998.

[UFW01] Georges G. Grinstein Usama Fayyad and Andreas Wierse. *Information Visualization in Data Mining and Knowledge Discovery.* Morgan Kaufmann Publishers, 2001.

[web01] Scientific visualization examples. On the Web at: http://www.wes.hpc.mil/scivis/examples.htm, May 2001.

[WL00] Christopher E. Weaver and Miron Livny. Improving visualization activity in java. In *Proceedings of the SPIE Conference on Visual Data*

Exploration and Analysis, pages 62–72, January 2000.

[WS81] C. Welty and D. Stemple. Human factors comparison of a procedural and a nonprocedural query language. *ACM Transactions on Database Systems*, 6(4):626–649, 1981.

[WS92] Christopher Williamson and Ben Shneiderman. The dynamic homefinder: Evaluating dynamic queries in a real-estate information exploration system. In Nicholas J. Belkin, Peter Ingwersen, and Annelise Mark Pejtersen, editors, *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Copenhagen, Denmark, June 21-24, 1992*, pages 338–346. ACM, 1992.

[WS97] Allison Woodruff and Michael Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In Alex Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 91–102. IEEE Computer Society, 1997.

[WWT99] P. Wong, P. Whitney, and J. Thomas. Visualizing association rules for text mining, 1999.

[YS93] M. Yen and R. Scamell. A human factors experimental comparison of SQL and QBE. *IEEE Transactions on Software Engineering*, 19(4):390–402, 1993.

[Zlo77] M Zloof. Query-by-example: A data base language. *IBM Systems Journal*, 16:324–343, Fall 1977.

A

Usability Experiment

This appendix describes the contents of the exam taken by subjects of the usability experiment. Each student subject received one of the exams, while the professionals completed both parts as well as a final preference survey.

A.1 Training Materials

In this section we provide the training materials that were used in the experiment. We present them here in a more compact form to conserve on space. In particular, when presenting the example queries we provide both solutions.

Training: SQL

SQL classification queries may be constructed in a variety of ways. For the purposes of this training, we assume that you are knowledgeable in the basic syntax of an SQL query. The examples we use to demonstrate classification tasks in SQL utilize the UNION set operation to combine the results of multiple queries. Queries may be

combined using the union operator as long as the output of each query has the same number of attributes, and that each attribute in the queries has the same datatype.

For example, the following SQL query is invalid, since the name of the employee is a string and the salary of employees is a numeric value.

```
Select Name   From Employee
   Union
Select Salary From Employee
```

The following query is syntactically valid:

```
Select Salary From Employee
   Union
Select Age    From Employee
```

The union operator removes duplicate rows. If you do not want to remove any duplicate rows use the UNION ALL operator.

Training: Mapping Language

Rule-based classification queries are specified as a list of rules. Each rule has the form:

```
Answer <- Condition
```

A natural way to understand the meaning of a rule is as follows. Given an input table, test each record in the input with the condition. If the condition evaluates to "True", then evaluate the answer expression and place the result in the output. The output contains all the attributes in the input table plus the answer. The system we have developed allows the user to name the answer attribute. For the purposes of this instructional paper we will use Qvalue as the name of the answer attribute. Each rule is equivalent to an SQL query of the form:

```
Select Answer as Qvalue, *
From   InputTable
Where  Condition
```

Rule-based queries are parameterized by an input table, which is significantly different than SQL queries. The restriction on the tables that can be queried for rule-based queries is simply based on the attributes of the input table. So long as each attribute referred to by a rule in the query is an attribute of the input table the query can be executed.

The purpose of each rule-based query is to classify records in the input table, which places a restriction on the data type of the answer attribute. We use the queries in a visualization system, so we require that the answers by numeric data.

Here is a simple example, using the following Employee table as input:

EmpId	LastName	Sex	Salary	Age
1	Smith	F	23,500	27
2	Jones	M	27,000	31
3	Thompson	F	48,950	25
4	Jackson	F	18,750	22
5	Johnson	M	38,000	40

Consider the following rule-based query, which selects employees from the table and assigns each Employee to a numeric class (1 or 2), based on their sex.

```
1 <- Sex = 'F'

2 <- Sex = 'M'
```

This query is evaluated against each record in the input and creates the following output:

Qvalue	EmpId	LastName	Sex	Salary	Age
1	1	Smith	F	23,500	27
2	2	Jones	M	27,000	31
1	3	Thompson	F	48,950	25
1	4	Jackson	F	18,750	22
2	5	Johnson	M	38,000	40

An equivalent SQL query can be issued to get the same results:

```
Select 1 as Qvalue, * From Employee Where Sex = 'F'
Union
Select 2 as Qvalue, * From Employee Where Sex = 'M'
```

The rule-based query language is evaluated in linear fashion. The user may specify at the time the query is executed whether they would like the rules to be interpreted functionally or relationally. A functional interpretation ensures that each row in the input maps to at most one row in the output, only checking as rules until a successful condition is found. In contrast, a relational interpretation allows each row in the input to be mapped to several rows in the output. The following example demonstrates this concept. We will use the same input table as the previous example:

EmpId	LastName	Sex	Salary	Age
1	Smith	F	23,500	27
2	Jones	M	27,000	31
3	Thompson	F	48,950	25
4	Jackson	F	18,750	22
5	Johnson	M	38,000	40

Consider the following rule-based query:

```
1 <- Age <= 25

2 <- Sex = 'F'

3 <- Sex = 'M'
```

Under a functional interpretation the output for this query would be:

Qvalue	EmpId	LastName	Sex	Salary	Age
2	1	Smith	F	23,500	27
3	2	Jones	M	27,000	31
1	3	Thompson	F	48,950	25
1	4	Jackson	F	18,750	22
3	5	Johnson	M	38,000	40
3	6	Smythe	M	66,000	41
2	7	Edwards	F	52,000	33
2	8	Adams	F	85,000	47
1	9	Benson	M	18,000	19
1	10	Davis	M	23,800	25
2	11	Flynn	F	41,250	37

The same query using a relational interpretation yields the following output:

Qvalue	EmpId	LastName	Sex	Salary	Age
2	1	Smith	F	23,500	27
3	2	Jones	M	27,000	31
1	3	Thompson	F	48,950	25
2**	3	Thompson	F	48,950	25
1	4	Jackson	F	18,750	22
2**	4	Jackson	F	18,750	22
3	5	Johnson	M	38,000	40
3	6	Smythe	M	66,000	41
2	7	Edwards	F	52,000	33
2	8	Adams	F	85,000	47
1	9	Benson	M	18,000	19
3**	9	Benson	M	18,000	19
1	10	Davis	M	23,800	25
3**	10	Davis	M	23,800	25
2	11	Flynn	F	41,250	37

The marked rows indicate rows in this output that do not exist in the functional interpretation.

There are two special types of rules that can be specified within a rule-based query. The following query excludes records from further processing:

```
Except <- Sex = 'M'
```

There is no restriction to the number or location of Except rules within the list. However, the location is an important design consideration. For example, the following query does not exclude males from further processing since the Except rule occurs after males are already classified.

```
2 <- Sex = 'M'
Except <- Sex = 'M'
```

This does not imply that the Except rules must occur before all other types of rules. For example, when rule-based queries are evaluated relationally, Except rules can be used in interesting ways. The following query classifies males, and then further classifies females based on their age.

```
2 <- Sex = 'M'
Except <- Sex = 'M'
3 <- Age < 30
4 <- Age >= 30
```

There is an equivalent SQL query that generates the same output.

```
Select 2 as Qvalue, * From Employee Where Sex = 'M'
```

```
Union
Select 3 as Qvalue, * From Employee Where Age < 30 and Sex != 'M'
Union
Select 4 as Qvalue, * From Employee Where Age >= 30 and Sex != 'M'
```

The second special type of rule is called an Else rule. Queries may contain a single Else rule that is evaluated only if all of the other rules are unsuccessful. For example, the following query is equivalent to our first example.

```
1 <- Sex = 'F'
2 <- Else
```

Again, there is an equivalent SQL query that generates the same output. In fact, there is an equivalent SQL query for every rule-based query.

```
Select 1 as Qvalue, * From Employee Where Sex = 'F'
Union
Select 2 as Qvalue, * From Employee Where Sex != 'F'
```

Sample Queries. **Query 1**: Assign every employee to class "1".

Input: The following table is used for each of the example queries.

EmpId	LastName	Sex	Salary	Age
1	Smith	F	23,500	27
2	Jones	M	27,000	31
3	Thompson	F	48,950	25
4	Jackson	F	18,750	22
5	Johnson	M	38,000	40
6	Smythe	M	66,000	41
7	Edwards	F	52,000	33
8	Adams	F	85,000	47
9	Benson	M	18,000	19
10	Davis	M	23,800	25
11	Flynn	F	41,250	37

```
Select 1 as Qvalue, *

From   Employee



1 <-
```

Output:

Qvalue	EmpId	LastName	Sex	Salary	Age
1	1	Smith	F	23,500	27
1	2	Jones	M	27,000	31
1	3	Thompson	F	48,950	25
1	4	Jackson	F	18,750	22
1	5	Johnson	M	38,000	40
1	6	Smythe	M	66,000	41
1	7	Edwards	F	52,000	33
1	8	Adams	F	85,000	47
1	9	Benson	M	18,000	19
1	10	Davis	M	23,800	25
1	11	Flynn	F	41,250	37

Query 2: Assign every female employee to class "1".

```
Select 1 as Qvalue, *

From   Employee

Where   Sex = 'F'
```

```
1 <- Sex = 'F'
```

Qvalue	EmpId	LastName	Sex	Salary	Age
1	1	Smith	F	23,500	27
1	3	Thompson	F	48,950	25
1	4	Jackson	F	18,750	22
1	7	Edwards	F	52,000	33
1	8	Adams	F	85,000	47
1	11	Flynn	F	41,250	37

Query 3: Assign every female employee to class "1" and every male employee to class "2".

```
Select 1 as Qvalue, *
From   Employee
Where   Sex = 'F'
Union
Select 2 as Qvalue, *
From   Employee
Where   Sex = 'M'

1 <- Sex = 'F'
2 <- Sex = 'M'
```

Qvalue	EmpId	LastName	Sex	Salary	Age
1	1	Smith	F	23,500	27
2	2	Jones	M	27,000	31
1	3	Thompson	F	48,950	25
1	4	Jackson	F	18,750	22
2	5	Johnson	M	38,000	40
2	6	Smythe	M	66,000	41
1	7	Edwards	F	52,000	33
1	8	Adams	F	85,000	47
2	9	Benson	M	18,000	19
2	10	Davis	M	23,800	25
1	11	Flynn	F	41,250	37

Query 4: Assign every employee to a class based on their salary. The class is calculated by dividing the salary by 1000 and then taking the floor of the resulting value. If your are not familiar with the floor function the floor function rounds down to the next smallest whole number. For example, floor(23.5) = 23, floor(1) = 1.

```
Select Floor(Salary/1000) as Qvalue, *

From   Employee

Floor(Salary/1000) <-
```

Qvalue	EmpId	LastName	Sex	Salary	Age
23	1	Smith	F	23,500	27
27	2	Jones	M	27,000	31
48	3	Thompson	F	48,950	25
18	4	Jackson	F	18,750	22
38	5	Johnson	M	38,000	40
66	6	Smythe	M	66,000	41
52	7	Edwards	F	52,000	33
85	8	Adams	F	85,000	47
18	9	Benson	M	18,000	19
23	10	Davis	M	23,800	25
41	11	Flynn	F	41,250	37

Query 5: Assign every female employee to a class based on their age.

```
Select Age as Qvalue, *

From   Employee

Where  Sex = 'F'
```

```
Age <- Sex = 'F'
```

Qvalue	EmpId	LastName	Sex	Salary	Age
27	1	Smith	F	23,500	27
25	3	Thompson	F	48,950	25
22	4	Jackson	F	18,750	22
33	7	Edwards	F	52,000	33
47	8	Adams	F	85,000	47
37	11	Flynn	F	41,250	37

Query 6: Assign every employee to class "1" if they earn less than 30000. Assign them to class "2" if they earn at least 30000 but less than 38000. If they earn at least 38000 and are male assign them to class "3". Otherwise assign them to class "4".

```
Select 1 as Qvalue, * From Employee Where Salary < 30000
Union
Select 2 as Qvalue, * From Employee Where Salary < 38000 and Salary >=
30000
Union
Select 3 as Qvalue, * From Employee Where Salary >= 38000 and Sex = 'M'
Union
Select 4 as Qvalue, * From Employee Where Salary >= 38000 and Sex = 'F'

1 <- Salary < 30000
```

```
2 <- Salary < 38000

3 <- Sex = 'M'

4 <- Else
```

Qvalue	EmpId	LastName	Sex	Salary	Age
1	1	Smith	F	23,500	27
1	2	Jones	M	27,000	31
4	3	Thompson	F	48,950	25
1	4	Jackson	F	18,750	22
3	5	Johnson	M	38,000	40
3	6	Smythe	M	66,000	41
4	7	Edwards	F	52,000	33
4	8	Adams	F	85,000	47
1	9	Benson	M	18,000	19
1	10	Davis	M	23,800	25
4	11	Flynn	F	41,250	37

Query 7: Assign every female employee that makes more then 30000 and is not 33 or 25 years old to a class based on their salary. Assign all other employees that earn 30000 or less and are not 33 or 25 years old to a class based on their age.

```
Select Salary as Qvalue, *
```

```
From   Employee

Where  Sex = 'F' and Salary > 30000 and Not (Age=33 or Age=25)

Union

Select Age as Qvalue, *

From   Employee

Where  Salary <= 30000 and Not (Age=33 or Age=25)


Except <- Age=33 or Age=25

Salary <- Sex = 'F' and Salary > 30000

Age    <- Salary <=30000
```

Qvalue	EmpId	LastName	Sex	Salary	Age
27	1	Smith	F	23,500	27
31	2	Jones	M	27,000	31
22	4	Jackson	F	18,750	22
40	5	Johnson	M	38,000	40
41	6	Smythe	M	66,000	41
85,000	8	Adams	F	85,000	47
19	9	Benson	M	18,000	19
41,250	11	Flynn	F	41,250	37

Query 8: Assign every employee to class "1" if they earn less than 30000. Assign them to class "3" if they earn more than 50000. Otherwise assign them to class "2".

```
Select 1 as Qvalue, * From Employee Where Salary < 30000

Union

Select 2 as Qvalue, * From Employee Where Salary >= 30000 and Salary <=

50000

Union

Select 3 as Qvalue, * From Employee Where Salary > 50000
```

```
1 <- Salary < 30000

2 <- Salary < 50000

3 <- Else
```

Qvalue	EmpId	LastName	Sex	Salary	Age
1	1	Smith	F	23,500	27
1	2	Jones	M	27,000	31
2	3	Thompson	F	48,950	25
1	4	Jackson	F	18,750	22
2	5	Johnson	M	38,000	40
3	6	Smythe	M	66,000	41
3	7	Edwards	F	52,000	33
3	8	Adams	F	85,000	47
1	9	Benson	M	18,000	19
1	10	Davis	M	23,800	25
2	11	Flynn	F	41,250	37

Pre-Exam Survey

The following information will be used for statistical analysis purposes. Please do not write your name on this or any other part of this set of pages.

Please circle the number that best represents your answer for the following questions.

1) The training materials were:

Clear 1 2 3 4 5 Unclear

2) Using SQL (Rule-Based) queries to perform classification tasks seems:

Simple	1	2	3	4	5	Complex
Powerful	1	2	3	4	5	Weak
Fast to write	1	2	3	4	5	Slow to write
Easy to learn	1	2	3	4	5	Difficult to learn
Easy to use	1	2	3	4	5	Difficult to use

A.2 Exam Problems

There are three input tables used for the exam. The first, a patient table with schema $\{PatientId, LastName, Sex, Age, PulseRate, SystolicBP, DiastolicBP\}$. The following problems involve the patient table:

Problem 1) We want to develop an understanding of the patients in the database. We propose to use the following age ranges to create an appropriate classification of the patients:

Less than 3 years old (Newborns)

Between 3 and 12 years old (Children)

Between 13 and 18 years old (Adolescents)

Between 19 and 65 years old (Adults)

More than 65 years old (Elderly)

Problem 2) We have received new national guidelines for analyzing pulse rates. These guidelines are identified in the following table:

Class	Average Rate
Adult Males	70 - 74
Adult Females	76 - 80
Adolescents	70 - 80
Newborns	100 - 140
Children	86 - 92
Elderly	50 - 65

Write a query that places each adult male patient into a "Low", "Average", or "High" class.

Problem 3) We have received new national guidelines for analyzing children's blood pressure rates. These guidelines are identified as follows (SystolicBP / DiastolicBP):

Ages three to five: 116/76

Ages six to nine: 122/78

Ages 10 to 12: 126/82

Ages 13 to 15: 136/86

Write a query that separates children into two risk categories - "At Risk" and "Not At Risk", using these guidelines.

Problem 4) For all patients older than 40 or younger than 20, create 4 classes - "Older Men", "Older Women", "Younger Men" and "Younger Women"

The student table had the schema $\{StudentId, LastName, GPA, HoursCompleted, Major, LiveOnCampus, International\}$. The following four problems used this student table:

Problem 5) Separate international students into classes based on whether they live on campus or off campus.

Problem 6) A student requires 128 hours to graduate. After a student is halfway through with their hour requirements we assess each student's GPA, depending on their major. If the major is "Computer Science", "Mathematics", or "Physics", we increase their GPA by 5%. Separate students based on their adjusted GPA.

Problem 7) Separate students into classes based on their GPA: "Above 3.5", "Below 2.5", and "Everyone Else".

Problem 8) Separate students into classes based on the hours completed. If a student has more than 100 hours they are "Almost Graduated". All other students are "Not Close".

The final table used in the exam was the stockmarket table with schema $\{StockSymbol,$

$CompanyName$, $YearlyHigh$, $YearlyLow$, $CurrentPrice$, $LastMonth$, $LastWeek$, $OwnStock$}. The problems associated with this table were:

Problem 9) We want to know whether stock's price is "trending down", "trending up", or "stable". A stock is trending up if its current price is more than last week's price and last week's price is more than last month's price. A stock is trending down if its current price is less than last week's price and last week's price is less than last month's price. Otherwise the price is stable.

Problem 10) We want to know whether we should "buy", "sell" a stock. We buy stocks when we don't own them, their price is trending up and they are nearer their yearly low than their yearly high. If we own a stock that that is trending down and is nearer its yearly high we consider selling it.

Problem 11) Classify stocks as being "Closer to Yearly Low" or "Closer to Yearly High" based on a comparison of the current price to the yearly high and low prices.

Problem 12) Classify stocks by the difference between the current price and last week's price.

A.3 Post-Exam Survey

1) Using SQL (Rule-Based) queries to perform classification tasks was:

Simple	1	2	3	4	5	Complex
Powerful	1	2	3	4	5	Weak
Fast to write	1	2	3	4	5	Slow to write
Easy to learn	1	2	3	4	5	Difficult to learn
Easy to use	1	2	3	4	5	Difficult to use

1) The solutions I gave for the exercises were::

Correct	1	2	3	4	5	Incorrect
Efficient	1	2	3	4	5	Inefficient
Neat	1	2	3	4	5	Sloppy

A.4 Preference Survey

This survey was completed by the professional programmers.

1) As far as classification tasks are concerned:

SQL Queries						Rule-Based Queries
Faster to write	1	2	3	4	5	Faster to write
Powerful	1	2	3	4	5	Powerful
Awkward	1	2	3	4	5	Awkward
Understandable	1	2	3	4	5	Understandable
Easy to learn	1	2	3	4	5	Easy to learn
Easy to use	1	2	3	4	5	Easy to use

1) As far as classification tasks that you think are simple are concerned:

SQL Queries						Rule-Based Queries
Faster to write	1	2	3	4	5	Faster to write
Powerful	1	2	3	4	5	Powerful
Awkward	1	2	3	4	5	Awkward
Understandable	1	2	3	4	5	Understandable
Easy to learn	1	2	3	4	5	Easy to learn
Easy to use	1	2	3	4	5	Easy to use

3) As far as classification tasks that you think are complex are concerned:

SQL Queries						Rule-Based Queries
Faster to write	1	2	3	4	5	Faster to write
Powerful	1	2	3	4	5	Powerful
Awkward	1	2	3	4	5	Awkward
Understandable	1	2	3	4	5	Understandable
Easy to learn	1	2	3	4	5	Easy to learn
Easy to use	1	2	3	4	5	Easy to use

Curriculum Vitae

Dennis Groth

701 E. Maxwell Ln.

Bloomington, IN 47401

Home phone: 812-331-2296

Email: dgroth@cs.indiana.edu

Education

- Ph.D., Computer Science, Indiana University, May, 2002
- B.S., Computer Science, Loyola University of Chicago, 1983

Research Interests

- Information visualization for databases
- Database query formulation and processing
- Human computer interaction
- Software engineering education
- Data mining
- Data modeling

Publications

Conference/Workshop Papers

- Dennis P. Groth and Edward L. Robertson: *An Integrated System for Database Visualization*, The Sixth International Conference on Information Visualization (IV02), July 2002.

- Chris M. Giannella and Mehmet M. Dalkilic and Dennis P. Groth and Edward L. Robertson: *Improving Query Evaluation with Approximate Functional Dependency Based Decompositions*, Proceedings of the 19th British National Conference on Databases (BNCOD2002), July, 2002.

- Dennis P. Groth and Edward L. Robertson: *An Entropy-Based Approach to Visualizing Database Structure*, The Sixth IFIP Working Conference on Visual Database Systems (VDB6), May 2002.

- Dennis P. Groth and Edward L. Robertson: *Discovering Frequent Itemsets in the Presence of Highly Frequent Items*, Rule Based Data Mining 2001.

- Dennis P. Groth and Edward L. Robertson: *It's All About Process: Project-Oriented Teaching of Software Engineering*, 14th Conference on Software Engineering Education and Training (CSEE&T), February, 2001.

- Dennis P. Groth and Edward L. Robertson: *Architectural Support for Database*

Visualization, New Paradigms for Information Visualization and Manipulation 1998.

Posters

- Dennis P. Groth and Edward L. Robertson: *An Integrated Approach to Database Visualization*, Advanced Visual Interfaces 2002, May, 2002

Technical Reports

- Chris M. Giannella, Mehmet M. Dalkilic, Dennis P. Groth, Edward L. Robertson: *Using Horizontal-Vertical Decompositions to Improve Query Evaluation*, Indiana University Computer Science, Technical Report 558, 2002.
- Dennis P. Groth and Edward L. Robertson: *It's All About Process: Project-Oriented Teaching of Software Engineering*, Indiana University Computer Science, Technical Report 532, 1999.

Textbooks

- Dennis P. Groth and Arijit Sengupta: *Introduction to Database Applications: A Problem-Solving Approach*, McGraw-Hill, 1998.

Teaching Experience

Lecturer – I450/I451 Designing and Developing an Information System. Developed a new course for Informatics majors, in which student teams propose, and implement an information system.

Instructor – P465/P565 Software Engineering for Information Systems I. Prepared and provided lectures, assignments and exams. The course focuses on the design of software systems for a real customer in a team environment.

Instructor – P466/P566 Software Engineering for Information Systems II. Prepared and provided lectures, assignments and exams. The course focuses on the implementation of software systems based on the design developed in the previous course.

Instructor – B665/B666 Software Engineering Management. Led, in a seminar format, the discussion of topics related to software engineering management and directed the supervisors of the project teams.

Instructor – A114 Introduction to Database Applications. Prepared and delivered lectures, assignments and exams.

Associate Instructor – I400/I590 Technology and Business. The course focused on entrepeneurship and business plan development for student proposed high-tech start-ups.

Associate Instructor – P465/P565 Software Engineering for Information Systems

I. Led discussion sections, graded exams and assignments, reviewed and assisted the project teams design documents.

Associate Instructor – P466/P566 Software Engineering for Information Systems II. Led discussion sections, graded exams and assignments, assisted the project teams with implementation details and systems support.

Teaching Awards

Teaching Excellence Recognition Award, 1999.

Work Experience

Prior to entering graduate school, I had substantial industry experience. Highlights from this experience are below:

1995 - 1996 Independent Software Development Consultant

Design and develop custom client/server applications, primarily within the pharmaceutical industry.

1994 - 1995 Director, Crawford Consulting, Inc.

Responsible for Chicago regional business activities, including: Customer and Proposal management; Hiring and staff development.

Researched and performed due diligence review for acquisition of a software company.

1990 - 1994 Manager, Systems Development, Moore Advanced Services, a Unit of Moore Business Forms

Designed and developed electronic form based systems as the major component of a strategy to assist customers of Moore Business Forms decrease their reliance on pre-printed business forms. Managed five separate development groups in different geographical locations simultaneously. Provided presentations to customers and prospective business partners in support of sales activity. Presented technical and strategic topics at several professional seminars, including, XPLOR and Uniface user group. Developed annual strategic plan for system development, including financial, resource and product planning. Designed and developed an EDI ANSI X.12 based supply chain management system. With two coworkers, received U.S. Patent # 5694551 for the system process.

1984 - 1990 Project Manager, Spectrum Healthcare Solutions, a Partnership of Baxter Healthcare and IBM Corporations

Managed a team of programmers and analysts developing remote electronic linkages between physician offices and hospital systems. Implemented solutions at more than 75 major health care facilities. Managed a team of programmers developing electronic insurance claim processing software. Developed the IBM Doctor's Office Manager System physician practice management system. Systems were sold to more than 10,000 offices.

1983 - 1984 Consultant, Andrea Data Systems

Developed custom applications for the wholesale distribution industry.