IMPLEMENTATION AND APPLICATIONS OF QUERY INTERFACES TO CONSTRAINT DATABASES IN A DISTRIBUTED COMPUTING ENVIRONMENT

By

CHIN-CHIH CHANG

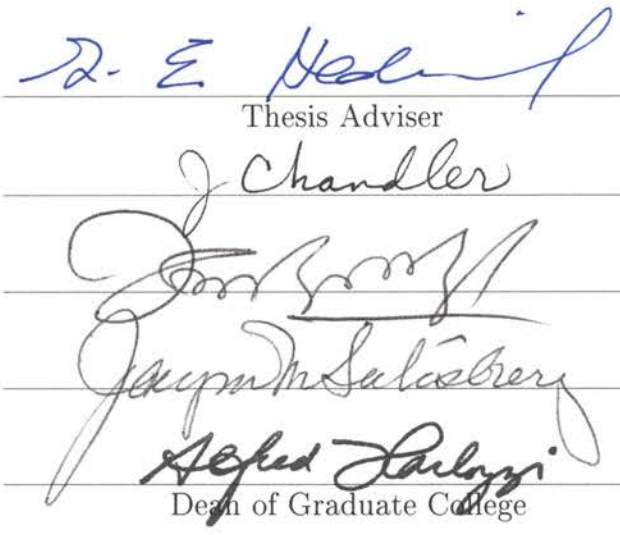
Bachelor of Engineering
Tamkang University
Taiwan, R.O.C.
1986

Master of Science
National Cheng Kung University
Taiwan, R.O.C.
1990

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial requirements for
the Degree of
DOCTOR OF PHILOSOPHY
December, 2000

IMPLEMENTATION AND APPLICATIONS OF
QUERY INTERFACES TO CONSTRAINT
DATABASES IN A DISTRIBUTED
COMPUTING ENVIRONMENT

Thesis Approved:



Thesis Adviser

Dean of Graduate College

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my advisor, Dr. G. E. Hedrick, for being a continual source of guidance, inspiration, and encouragement. Without his critical reviews and insightful comments this dissertation would not have been possible. I want to express my sincere gratitude to Dr. Jayne M. Salisbury for offering me a research assistantship at Spatial Environmental Clearing House (SEIC) and serving as my advisory committee member. My thanks also go to Dr. John P. Chandler and Dr. K. M. George for having made their time available to serve on my dissertation committee and giving me their thoughtful suggestions on my dissertation.

I want to thank my family for always believing in me. Their support, constant encouragement, and love made me continue my study throughout these years. My sisters and their children always give me a big welcome during my break in Taiwan. I would not forget to thank my late mother, Show Chen, for her blessing in heaven.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

Many database applications require more general data models and more efficient query handling than databases currently support. Some examples are geographic information systems (GIS), Computer-Aided Design (CAD), Very Large Scale Integration (VLSI) design, On-Line Analytic Processing (OLAP), robotics, and image repositories. Conventional relational database management systems (RDBMSs) are not well-suited for the storage and management of multidimensional data. This is due to the structural complexity of multidimensional data. This complexity requires both new querying facilities and specific indexing techniques. Considerable research has been devoted to database systems which can store and manage multidimensional data. One recent survey is done by Gaede and Günther [GG98].

Multidimensional data range over different domains where there can be infinitely many data points; they are spatial data in the space domain and temporal data in the time domain. These sets of potentially infinite numbers of data points make implementation of relational database systems even more difficult. To illustrate the implications of multidimensional data, specifically, spatial data are analyzed in this research. Spatial database models and prototypes proposed in the literature typically focus on one specific type of multidimensional information or a finite set of specific types. This narrow focus is justified, since it is sufficient for many applications, and it permits tuned efficient implementations [PdBG94]. However, most of these models and prototypes lack the integration between multidimensional data processing and standard data management [GW97]. Frequently, there is no high-level query facility [GRSS97]. Furthermore, these models only can be used in a fixed number of dimensions [BBC97]. The challenge is the design of data models that are both

general enough and powerful enough to handle conventional data as well as spatial data in an unified framework [Par95].

The constraint database model introduced by Kanellakis, Kuper and Revesz [KKR90] is a promising paradigm for the representation of multidimensional data in an unified framework. The basic idea extends a tuple to contain a conjunction of constraints on a predefined number of variables. For example, in two dimensions there are two variables which could correspond to two coordinates or two attributes. These two variables can be combined using different mathematical operations to form different algebraic equations. Such a tuple termed as a *generalized tuple* can actually represent infinite data. A *generalized relation* is a finite set of generalized tuples and a *generalized database* is a finite set of generalized relations. The constraint data model appears to have more expressivity than the relational data model. While infinite multidimensional data cannot be represented in the relational data model, it can be represented by equations or inequalities. Some may argue that the object-oriented data model could also describe multidimensional data, but one must remember that the object-oriented data model only can represent finite amounts of data. The operations for multidimensional data also can be defined in the constraint data model.

1.1 Thesis

There are still many problems under investigation. Until recently, very few constraint database systems have been built. Most constraint database systems are still in a very primitive stage. A prototype implementation of a constraint database system is the focus of this dissertation. Many choices must be made to build a constraint database system. In our implementation, an add-on system is built on top of a RDBMS. The graphic query interface extends the *Structured Query Language (SQL)* to incorporate constraints. The Java language is chosen mainly because of its portability and simplicity. Running the application on *Common Object Request Broker Architecture (CORBA)* allows distributed processing across different platforms. *Internet Inter-ORB Protocol (IIOP)* is essentially a *Transmission Control Proto-*

col/Internet Protocol (TCP/IP) with some CORBA-defined message exchanges that serve as a common backbone protocol to ensure the interoperability. *Hypermedia* is a multimedia system that is enriched with links to other documents. The *World Wide Web* (known as *WWW*, *Web* or *W3*) is a hypermedia information delivery system on networks [Com95, Kro94, KF95, Wig95]. This reflects a vision "**The Network is the Computer**" by Sun Microsystems, Inc. [Eco95]. The integration of distributed objects and the Web is called the *Object Web*. The Object Web is the latest paradigm for the distributed computing and the Web [OH98, OHE97b, OHE97a]. This description demonstrates that building a CORBA-based application on the Web is practical. Several applications are presented in this research. Tax calculation and stock database query are two Web-based applications. GIS applications, linear programming applications, and spatio-temporal applications are examples running on a general query interface to constraint databases.

1.2 Organization

This dissertation is organized as follows. Chapter I is an introduction. Chapter II explains the fundamental concepts of multidimensional data, constraint databases, and Object Web. This chapter lays the foundation for the later discussion. Chapter III presents the implementation of two Web-based applications which store constraint data into a relational database management system (RDMS) directly. The implementation of distributed computing applications is illustrated here. Chapter IV gives a detailed description of the implementation of a general query interface to constraint databases. The simple implementation from the chapter III is limited to one-dimensional data and insufficient in general and more complex cases. The implementation of constraint database systems for general applications is presented here. Chapter V focuses on the applications on this query interface. Chapter VI summarizes the study and the prospects for future work.

CHAPTER II

FUNDAMENTAL CONCEPTS

2.1 Multidimensional Data and and Geographical Information Systems (GIS)

We discuss issues of multidimensional data in this section. Spatial data are used as an example to illustrate the implications of multidimensional data in database design and implementation. Some current solutions are presented, and then it is shown that most of them work only on one specific type of spatial data; thereby lacking integration between multidimensional data processing and standard data management. The thought of looking for a more general and effective solution comes into our mind. The next section presents constraint databases, which are a promising solution to this issue.

2.1.1 Properties of Spatial Data

To understand the complication of the spatial database design, we must know the properties of spatial data. Gaede and Günther summarized the properties of spatial data in [GG98]:

- Spatial data have complex structures.
- Spatial data are often dynamic.
- Spatial databases tend to be large.
- It is difficult to define a standard algebra for a wide variety of spatial data.
- Many spatial operators are not closed in the domain of operands.

- The computation costs of spatial operators are generally more expensive than standard relational operators.

A spatial database query involves both alphanumeric attributes and spatial location of a data object. Retrieval and update in a spatial database require a fast spatial search operation such as point and region query. Both operations demand fast access to those spatial objects in the database.

Specific multidimensional access methods are required for such search operations. The primary issue for the design of such methods, nonetheless, is that there is no total ordering among spatial objects that preserves spatial proximity [GG98]. This makes the design of efficient access methods in spatial databases much harder than in traditional databases. Generally, multidimensional access methods should meet the criteria as described in [Rob81, LS89, NHS84, GG98]:

1. *Dynamics.* Access methods should keep track of the database changes continuously.
2. *Secondary/tertiary storage management.* Access methods must consider secondary and tertiary storage management.
3. *Broad range of supported operations.* Access methods should support various operations.
4. *Independence of the input data and insertion sequence.* Access methods should be efficient, irrespective of the input data and insertion sequence.
5. *Simplicity.* Access methods should be simple.
6. *Scalability.* Access methods should adjust well to database growth.
7. *Time efficiency.* Spatial searches should be fast.
8. *Space efficiency.* An index should occupy small space.
9. *Concurrency and recovery.* Access methods can operate concurrently and perform recovery in case of database crash.

10. *Minimum impact.* An access method should be incorporated into a database system with minimum impact on the existing system.

2.1.2 Definitions and Queries

Multidimensional access methods indicate any access method that support searches in spatial databases. Further, they can be categorized into *point access methods (PAMs)* and *spatial access methods (SAMs)* [GG98]. While the point access methods mainly manage spatial searches on point databases, spatial access methods process extended objects, such as lines, polygons, or even higher-dimensional polyhedra. *Spatial access methods* are also denoted as *spatial index* or *spatial index structure.*

Since spatial data have complex structures, it is inefficient to index them directly. Usually, the *minimum bounding box (MBB)* or *decomposition* is used to approximate the original data. An MBB is formed by the bounding interval in each dimension of the data object. Decompositions are formed by dividing the data object into simple components such as triangles, trapezoids, or convex polygons. Then the search becomes a multi-step process [BKSS94]. First, based on the approximation, candidate solutions are found. This is the *filter* step. Then the exact solutions are searched according to the exact shape information. This is the *refinement* step.

Space and time efficiency are the main concerns for an efficient spatial access method. In the case of space efficiency, the goal is to minimize the number of bytes occupied by the index. For time efficiency the number of disk accesses is used as the measure since most spatial searches are I/O-bound rather than CPU-bound.

As noted above, it is difficult to define a standard spatial algebra or a standard spatial query language. The set of operators strongly depends on the specific application domain. The previous work [RL84, Güt89, SV89, AS91, HSH92, GS93, Ege94] usually extends the standard SQL to express the spatial query. *OpenGIS Simple Features Specification For SQL* [Ope99] from Open GIS Consortium, Inc. and SQL3/MM Spatial Standard [Int99] of International Standard Organization (ISO) are two current standards. They represent two different approaches. Open GIS Consortium's

approach conceptually stores geospatial feature collections as tables while ISO's defines abstract data types to represent spatial objects and their associated operations. These operations fall into three categories:

1. **Update operations**: *insert*, *delete* or *modify*.

2. **Spatial selections** are fundamental spatial queries.

   - **Point Query** yields all objects overlapping a given point.
   - **Region Query** yields all objects sharing points with a query polygon. A special case of the region query is the **window query** where the query polygon is given by a rectilinear rectangle.
   - **Nearest Neighbor Query** yields all objects having a minimum distance from a given query object.

3. **Spatial Join** computes all pairs of objects that satisfy the specified spatial predicate. Different predicates [Ore86, Bec92, Ind91, Gün93, BKS93, GR94, Bri94, LR94, AS94, PTSE95] indicate different spatial joins. We list some common ones:

   - **Intersection Query, Overlap Query** (intersection join) yields all objects having at least one point in common with a given object.
   - **Containment Query** (containment join) yields all objects that are enclosed by a given object.
   - **Enclosure Query** (enclosure join) yields all objects enclosing a given object.
   - **Adjacency Query** (adjacency join) yields all objects adjacent to a given object.

Usually, spatial joins are difficult to do rapidly since they combine two spatial relations. They are the most time-consuming operations in spatial databases. In

order to make spatial queries efficient, spatial join must be efficient as well [BKS93]. Efficient spatial join methods continue to be a major topic. The recent research can be found in [APRS00].

As mentioned above spatial join is the most expensive spatial operation. A closer look at these spatial predicates indicates that the *intersection join* plays an essential part in different spatial joins [GR94, GG98]. For example, in the case of Containment Query, Enclosure Query, or Adjacency Query, Intersection Query is an efficient filter that produces a set of candidate solutions typically much smaller than the Cartesian product of two spatial relations. The intersection query is the main implementation in this research. We use the following example to illustrate intersection query.

Example: Find all intersections of roads and rivers.

```
select all
  from land-use l, land-use k
where intersect(l.location, k.location)
  and l.usage = "road" and k.usage = "river"
```

The spatial predicate *intersect* qualifies all roads which intersect the rivers. The result of this operation is a join relation which contains all the attributes of the two participating relations. A tuple in the resulting join relation corresponds to two spatial objects that intersect each other. The intersections in this query are bridges.

2.1.3 Spatial Access Methods

The relevant literature and contributions in spatial databases are too large to attempt a complete survey in a limited section. Gaede and Günther have a recent comprehensive survey [GG98]. Most of these methods originally were designed to manage large sets of multidimensional points and to support exact match or partial match queries. Usually, the points in the database are organized in a number of buckets. Each bucket is a collection of records stored on a disk page and corresponds to some part of the entire data. The buckets are then built into a search tree or a hash table.

Early proposals such as the *k-d-tree* [Ben75, Ben79] and the *binary space partitioning* (*BSP*) tree [FKN80, FAG83] did not account for paged secondary memory and are therefore unsuited for very large spatial databases. These trees are binary search trees built by recursively partitioning the data space into subspaces. Point and range searches can be performed by means of simple tree traversals. The problem is they typically are not balanced and may have very deep subtrees causing a negative impact on the performance of tree operations.

Other access methods take secondary storage management into consideration. The grid file [NHS84] uses a directory and a grid-like partition of the data space to answer an exact match query with exactly two disk accesses. Though the grid file needs only two disk accesses, the grid file directory could expand exponentially. The problem is addressed in [CHLF89]. The directory expansion has only a quadratic or cubic rate in their method. Furthermore, there are multidimensional hashing schemes [Tam82, KS86, KS88], and hash trees where the directory is organized as a tree structure [Ouk85, Oto86, Fre87]. Tree-based access methods are also popular. They usually generalize the B-tree into higher dimensions, such as the k-d-B-tree [Rob81], the hB-tree [LS90], or the buddy tree [SK90].

The point access methods mentioned above cannot apply to databases containing extended spatial objects such as polygons. To manage such extended objects, point access methods have been modified using one of the following three techniques, as discussed in [SK88, Güt94, Sam95, GG98]: *transformation*, where extended spatial objects are mapped into higher-dimensional points; *overlapping regions*, where different buckets for each object may overlap; and *clipping*, where objects are decomposed into disjoint cells, which are mapped into buckets.

The transformation technique is based on parameterizing extended spatial objects, then representing them as points in a higher dimensional space [Hin85]. Search operations can be expressed as point and range queries in this higher dimensional space; they can be computed by means of a point index. The transformation approach is not always adequate for handling the wide variety of spatial queries. There

are four main drawbacks [Sam95, GG97]:

1. The transformation approach ignores the extent of data [Sam95]. Two nearby intervals in the original space may be far apart in point space [FSR87].

2. Formulation of range queries in point space may be much more complicated than it is in the original space.

3. Transformation works only for relatively simple objects such as rectangles.

4. Transformation of objects in very high dimensions is inefficient.

In the case of overlapping region schemes, objects are grouped into hierarchies, and then stored in another structure such as a B-tree. A well-known example is the R-tree [Gut84]. Each internal node in an R-tree corresponds to a bounding box which covers all bounding boxes containing the children nodes. Sibling nodes, which are nodes with an identical parent node, may correspond to overlapping intervals.

The problem of this approach is that an object is associated with only one bounding box and that box could overlap other boxes. The situation becomes even worse when there are objects of varying size in one bounding box, typical for geographic applications where one must represent objects of widely varying size such as buildings and states in the same spatial database. Moreover, each insertion of a new data object may increase the overlap. In the worst case, one must search the entire database to find a specific object. Many studies show it is very important to minimize the overlap [RL85]. This led the development of deferred-splitting techniques as in the R*-tree structure [BKSS90].

In the case of clipping schemes, during insertion, each new data object is decomposed into disjoint nodes in the index structure. Several leaf node entries may be created for the same object. The price paid for the disjointness is not only an increase in the average search time but also an increase in the frequency of node overflow, which makes updates much more complicated. Examples for clipping-based schemes where these problems can occur include the R^+-tree [SSH86, SRF87], an R-tree

variant where no overlaps are allowed except at the leaf level. The result is a very efficient point query but range query, insertion and deletion operations become rather complicated [GB91].

Gaede and Günther [GG98] attempted a summary for all experimental comparisons of different multidimensional access methods and made the following recommendation for the best performing ones: buddy (hash) tree [SK90], cell tree with oversize shelves [GG97], Hilbert R-tree [KF94], KD2B-tree [vO90], PMR-quadtree [NS87], R^+-tree [SRF87], and R^*-tree [BKSS90]. The multidimensional data models and access methods described above are not general enough and often allow only the representation and operations of specific spatial objects in fixed dimensions.

2.1.4 Geographical Information Systems (GIS)

GIS stands for Geographic Information Systems. A geographic information system is a computer system capable of assembling, storing, manipulating, and displaying geographically referenced information. Most existing GIS depend on a strict distinction between alphanumerical data and spatial data. For example, *Arc Spatial Data Engine* (ArcSDE) by the Environmental Systems Research Institute (ESRI) is integrated with and supports the major functions and capabilities of a commercial DBMS which has a separate module devoted to spatial data. Examples of such separate modules are *Oracle Spatial*, *IBM's DB2 Spatial Extender*, and *Informix's Spatial DataBlade*. The common shortcoming of these systems is the lack of uniformity for the representation of data. Many research models and prototypes also favor specific algebras and spatial extensions of conventional database models [RFS88, OM88, SV89, Güt89, GS93]. Furthermore, the operations based on simple data structures are inept for the costly operations involved in geospatial object management. Another weakness of traditional GIS is the lack of support for time domain. Therefore, the better solution is to introduce a flexible and extensible data model, a general query algebra, and efficient indexing methods.

2.2 Constraint Database

Constraint database systems, on the other hand, have the potential to overcome some problems mentioned above. Constraints were introduced into databases during the past few years. In general, constraints are incorporated into a database in three different ways [GG95]. First, integrity constraints on data define valid database states. For example, a person cannot have negative height. The database system must ensure the validity of these constraints during updates, insertions, and deletions. Second, queries can be interpreted as a kind of constraint: Each query predicate represents a constraint that the objects in the result must satisfy. Third, constraints can represent database objects or sets of objects such as all people of height 6 feet or less. This technique generalizes the traditional notion of a database tuple, which corresponds to a point constraint in this model. The database with constraints at a data level is then specified as a *constraint database.*

Although integrity constraints in databases have been studied for several years [KSS87, Wal91], they still have not integrated seamlessly into database systems other than constraint databases. The work proposed in [HHLvEB89] is an early study of integrating constraints into the database at the language level using queries. The potential of the constraint database was then presented by Kanellakis, Kuper and Revesz in their work [KKR90, KKR95] where they introduced the constraint as a basic data type and described the *Constraint Query Language* design principles. The constraint database model is at the intersection of databases, constraint programming, computational geometry and operations research [Bro96b, GW97]. Several issues about constraint databases are still under investigation [Bro96a, Bro96b, GW97]. For the recent survey of constraint databases, please refer to [KLP00].

2.2.1 Basic Definitions and Data Modeling

The basic idea in constraint databases is to generalize the relational model [Cod70] in order to have infinite databases that are finitely representable by constraint formulae

as a basic data type. A constraint (formula) identifies a formula of a decidable logical theory. Constraints have been used in areas such as programming languages, artificial intelligence, and operations research. Several classes of constraints with specificed domains have been proposed. The domain could be real numbers, rational numbers, integer numbers, or any other finite or infinite set. The universe of real numbers, rational numbers, and integer numbers can be denoted as $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{I}$ respectively. Some of constraint classes are presented here [KKR90, KKR95].

1. *Equality constraints* are all formulae of the form $x\theta y$ and $x\theta c$, where x, y are variables, c is a constant, and θ is $=$ or $\neq$. For example, $x = y, y \neq 4$.

2. *Dense-order constraints* are all formulae of the form $x\theta y$ and $x\theta c$, where x, y are variables, c is a constant, and $\theta \in \{=, \leq, <, \neq, >, \geq\}$. For example, $x > 3, y \leq 4, y > x$.

3. *Linear constraints* are all formulae of the form $a_1x_1 + \cdots + a_kx_k\theta a_0$, where $a_0, \cdots, a_k$ are constants, and $\theta \in \{=, \leq, <, \neq, >, \geq\}$. For example, $2x - 3y < 6, y + 2x > 4$.

4. *Polynomial constraints* all formulae of the form $p(x_1, \cdots, x_k)\theta 0$, where p is a polynomial with coefficients, variables $x_1, \cdots, x_k$, and $\theta \in \{=, \leq, <, \neq, >, \geq\}$. For example, $x^2 + y^2 = 9, y - 4x^2 > 16$.

Before we discuss the definitions of constraint databases, we need some terminology from mathematical logic. A *literal* is a propositional letter or its negation. For example, $P, Q, \neg P$, and $\neg Q$ are literals. A *fundamental conjunction* is a literal or the conjunction of two or more literals. For example, P and $\neg P \wedge Q$ are fundamental conjunctions. A *disjunctive normal form (DNF)* is either a fundamental conjunction or a disjunction of two or more fundamental conjunctions. A *fundamental disjunction* is a literal or the disjunction of two or more literals. For example, P and $\neg P \vee Q$ are fundamental disjunctions. A *conjunctive normal form (CNF)* is either a fundamental disjunction or a conjunction of two or more fundamental disjunctions.

The following definitions of constraint databases are from [KKR90, KG94b, GST94, KKR95, Kan95].

- A generalized (or *finitely representable* in [GST94]) k-tuple t_i over variables $x_1, \cdots, x_k$ is a finite conjunction $\varphi_{i,1} \wedge \cdots \wedge \varphi_{i,\ell_i}$, where $\varphi_{i,1}, \cdots, \varphi_{i,\ell_i}$ are constrants. This can be denoted as:

$$t_i = \bigwedge_{j=1}^{\ell_i} \varphi_{i,j}$$

- A generalized (finitely representable) relation R of arity k is a finite set of $\{t_1, \cdots, t_n\}$, where each $t_1, \cdots, t_n$ are over the same variables $x_1, \cdots, x_k$. The formula φ corresponding to the generalized relation R is a first-order formula in DNF of constraints, which uses at most k variables. This can be denoted as:

$$\varphi = \left\{ t_i | 1 \leq i \leq n, t_i = \bigwedge_{j=1}^{\ell_i} \varphi_{i,j} \right\}$$

or

$$\varphi = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{\ell_i} \varphi_{i,j}$$

- A generalized (finitely representable) database is a finite set of generalized relations.

As an example consider a tuple $t(2,3)$ of arity 2 in the relational database model. It can represent a single point in a two dimensional space. We can also denote it as $t(x,y)$ with $x = 2$ and $y = 3$, where x, y range over some finite set. $t(x,y)$ with $(1 \leq x \leq 3, 2 \leq y \leq 4)$ is a generalized tuple of arity 2, where x, y range over the real numbers. In this case it is a rectangle in two dimensional space, and it contains infinite number of points. This is an example that a generalized tuple of arity k could represent a infinite set of tuples (or points in k-dimensional space).

As mentioned in Section 2.1, multidimensional data appear in many applications. Currently multidimensional data must often be stored in a dedicated system because most database systems are not designed for multidimensional data. It is clear such a

separation makes applications harder to build and prevents high level query facility and various query optimizations. Generalized tuples have the potential to put all these various kinds of data within a single framework in a very natural way. Typically, alphanumeric data is captured as equality constraints, spatial and temporal data as linear constraints.

Example: The instance in Figure 2.1 is represented by the following three generalized relations:

$$R_1 = \{[5 \leq x \leq 8 \wedge -1 \leq y \leq 3] \vee [5 \leq x \leq 7 \wedge 3 \leq y \leq 4]\}$$

$$R_2 = \{[-1 \leq x \leq 3 \wedge 1 \leq y \leq 5] \vee [-2 \leq x \leq -1 \wedge 2 \leq y \wedge y - x \leq 6]\}$$

$$R_3 = \{[x - y = 0] \vee [x = 4] \wedge [-2 \leq y \leq 6]\}$$

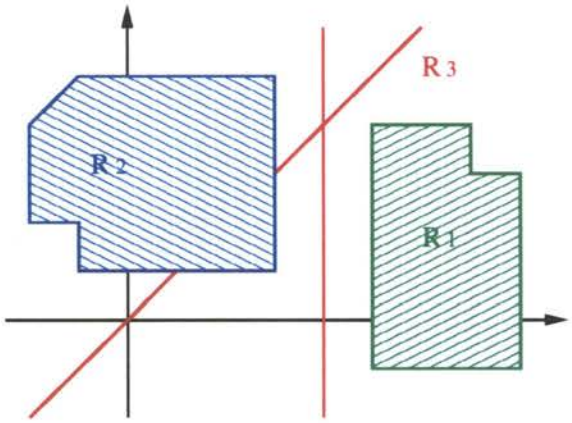


Figure 2.1 Example of Spatial Data

In this example, different spatial data (e.g. either a region or a line) are represented in a generalized tuples framework. When we consider the intersection query of either two regions or regions and lines, it becomes a very simple constraint intersection. If we use the multidimensional access methods in Section 2.1.3, the task seems less efficient: first search the objects based on MBB and then evaluate the query through the precise spatial information of the objects.

When we consider a general framework for spatial data, we could model spatial data as infinite sets in the relational space as shown in the above examples. A polygon in the plane is a 2-tuple which contains the infinite set of points in $\mathbb{Q}^2$ inside its frontier defined by the inequalities. A 3-dimensional pyramid is a 3-tuple which

contains the infinite set of points in $\mathbb{Q}^3$ inside its facets defined by the inequalities. An n-dimensional polyhedron is an n-tuple which contains the infinite set of points in $\mathbb{Q}^n$ defined by the inequalities of n variables.

2.2.2 The Algebra for Constraint Databases

It is important to define an algebra for constraint databases in order to make constraint databases as a practical technology. The algebraic approach represents the correct formal framework and a suitable basis for implementation. Since the introduction of constraint databases, a lot of work has been done to define an algebra for linear constraint databases. Linear constraint databases have been studied extensively because of their applicability and potential for efficient implementation. In [Bro95, HK95], linear constraints are incorporated into an object-oriented framework. In [PdBG94, VGG95, BBBC95, BBC97, AAK97, VGG98, GdR00], constraints are used in modeling spatial data and spatial queries. In [GST94, AAK95, AAK97, GVG00], the expressive power and complexity of linear constraint query languages are investigated.

Our main focus here is to find an algebra which can be used as the logical framework for the current implementation. *CALG* (a linear constraint algebra) proposed in [GST94, GL95] and *EGRA* (an extended algebra for constraint databases) introduced in [BBBC95, BBC97, BBC98] present the algebra for our main references.

The linear constraint algebra introduced in CALG is equivalent to the first-order logic over the class of linear constraint databases [GST94]. The basic concept is to apply the algebraic operators introduced by Codd for finite relations [Cod70] to either finite or infinite relations. A *symbolic algebra* is defined as an extension of the relation algebra, with operators applying to the finite representations of the possibly infinite relations. Similarly, EGRA is a direct extension of the relation algebra and is derived from algebra presented in the algebra of Kanellakis and Goldin [KG94a, PdBG94, Gol96].

The operations and mathematical symbols defined in these two algebras follow the

convention of relation algebra. They include *Cartesian product*, $\times$, *join*, $\bowtie$, *selection*, σ, *union*, $\cup$, *set difference*, $-$, *projection*, π, and *rename*, ρ. The algebraic operations are performed on sets of generalized tuples, i.e., on quantifier-free formulae in DNF.

From the definition of a generalized relation, consider an n-ary relation R represented by a quantifier-free formula, φ, of the form:

$$\varphi = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{\ell_i} \varphi_{i,j}$$

or

$$\varphi = \left\{ t_i | 1 \leq i \leq n, t_i = \bigwedge_{j=1}^{\ell_i} \varphi_{i,j} \right\}$$

where $t_i, \cdots, t_n$ are generalized tuples and the $\varphi_{i,j}$ are linear constraints of the form

$$\sum_{i=1}^{p} a_i x_i \theta a_0$$

where $\theta \in \{=, \leq, <, \neq, >, \geq\}$ and the x_i's denote variables and the a_i's are constants.

Here, we can see that the linear constraint model generalizes the relational data model in a very natural, simple and efficient way. It is not quite as simple to represent geographical data in the relational model because geographical data mixes alphanumeric data and spatial data.

Next, we consider the algebraic operations. In [BBC98], the operations are further categorized into tuple operators, which handle generalized relations as an infinite set of points, and set operators, which handle each generalized relation as a finite set of spatial objects. We follow the notions in [GST94, GL95]. Let R_1 and R_2 be two relations, and respectively e_1 and e_2 be sets of generalized tuples defining them.

1. $R_1 \bowtie R_2 = \{t_1 \wedge t_2 | t_1 \in e_1, t_2 \in e_2\}$.

2. $\sigma_P(R_1) = \{t_1 \wedge P | t_1 \in e_1, p \in P\}$.

3. $\pi_{\bar{x}} R_1 = \{\pi_{\bar{x}} t | t \in e_1\}$, where

$$\pi_{\bar{x}} t = \bigwedge_{1 \leq k \leq K, 1 \leq \ell \leq L} b^k \bar{x} - b_0^k \leq a_0^\ell - a_0 \bar{x} \wedge \bigwedge_{1 \leq i \leq I} c^i \bar{x} \leq c_0^i$$

defines a polyhedron $P(\bar{x}, y) \subseteq \mathbb{Q}^{n+1}$ described by the inequalities (once the coefficients of y have been normalized):

$$\begin{cases} a^{\ell}\bar{x} + y \leq a_0^{\ell} & for \ \ \ell = 1, \cdots, L \\ b^{k}\bar{x} - y \leq b_0^{k} & for \ \ k = 1, \cdots, K \\ c^{i}\bar{x} \leq c_0^{i} & for \ \ i = 1, \cdots, I \end{cases}$$

where $\bar{x}$ ranges over $\mathbb{Q}^n$, and y over $\mathbb{Q}$.

4. $R_1 \cup R_2 = \{t | t \in (e_1 \cup e_2)\}$.

5. $R_1 - R_2 = \{t_1 \wedge t_2 | t_1 \in e_1, t_2 \in \neg e_2\}$, where $\neg e$ is the set of tuples or disjuncts of a DNF formula corresponding to $\neg e$.

Similarly to the relational calculus, there is an equivalent constraint calculus for a constraint algebra [KKR90]. When we write a constraint-algebra expression, we provide a sequence of procedures that generates the answer to our query. By contrast, the constraint calculus is a nonprocedural query language. It describes the information without giving a specific procedure for obtaining that information. The recent discussion about the constraint calculus can be found in [GW97, Rev00a].

2.2.3 Constraint Solving

A class of constraints together with their semantics is often termed a *constraint domain*, for example, linear constraints over the rational numbers. A generalized tuple is a set of constraints over a given constraint domain. In order to ensure correctness and completeness for a constraint domain, an algorithm is required to decide whatever constraints over the given domain are satisfiable. Such an algorithm is termed a *constraint solver*. A generalized tuple is a set of constraints over a given constraint domain. In the worst case, a relational database should answer unquantified queries in polynomial time. For this result to apply to constraint databases, it is necessary that constraint solving itself has only polynomial complexity. Obviously each access to a generalized tuple needs the constraint solver, so very fast constraint solving is crucial.

Constraint solving was first investigated in *Constraint Logic Programming (CLP)* systems. Most CLP systems which manipulate constraints over $\mathbb{R}$ or over $\mathbb{R}^n$ for some n incorporate constraint solvers based on either the Simplex algorithm (linear constraints) [Dan63, Sch86, NT93] or on Buchberger's algorithm (non-linear constraints) [Buc85]. Constraint database systems need the the same kind of solver which evaluates all constraints at run-time.

Any particular CLP language typically will include several constraint domains. Then there will be several solvers for different constraint domains. Some of these solvers will be replaced by a set of *constraint checker* and *constraint agents.* A constraint checker is able to check the consistency of the constraints once all variables are instantiated. Constraint agents are those programs that ask the constraint solving from the constraint solvers. Constraint agents can handle constraints with uninstantiated variables, but do not guarantee detection of all inconsistencies.

Usually, the constraint solvers designed for CLP systems are more general and might not be efficient for the various applications on constraint database systems. In the actual implementation some specific modifications is necessary.

2.2.4 I/O Efficiency and Query Optimization

To access large sets of constraints efficiently, constraint database systems must be I/O efficient. I/O efficiency is accomplished by index structures such as B-trees and their variants (B^+ [BM72], B^* [Knu73]) in relational database systems. B-tree variants [Ram97, ZSI99, Ram00] and multidimensional access methods [KRVV93, Fre95, KRVV96] are proposed to index constraint databases. Multidimensional access methods that index a spatial object can be used to index sets of constraints which define or delimit this spatial object. Spatial indices are often only suited for indexing data in two or three dimension, but constraints often tend to be in four or more dimensions [GW97]. Moreover, indexing nested constraints in the presence of disjunctions of conjunctions should be scrutinized. These problems should be addressed in the future research.

Query optimization is another important issue in constraint database systems. Whether conventional optimization strategies such as cost-based plan selection, push-down selections, join ordering, algebraic transformation, and simplification can be used in constraint database systems requires further study. Traditional query processing strategies need to be investigated with respect to their applicability in constraint database systems. The recent research on optimization techniques can be found in [GLRS00].

2.2.5 Systems and Applications

The preceding discussion reflects the theoretical advance in constraint databases. Practical results are mainly in the constraint programming area [GW97]. There is a strong demand to implement constraint database systems. We can have a better understanding of constraint databases through implementation. Such systems are also a testbed for various promising application areas. The study on practical applications also introduce further research on theoretical issues.

We follow the work by Gaede and Wallace in [GW97] to introduce the implementation of constraint database systems. They illustrate three different approaches for building constraint database systems.

1. *Constraint programming centered view.* This approach extends current constraint programming systems with constructs to handle persistent data and other database features. The programming language enhanced by constructs to handle persistent data is a *persistent programming language.* This same approach has been employed to build object-oriented databases.

   The *ECL^{i}PS^e Constraint Logic Programming System* [WNS97, WS99] developed at the European Computer-Industry Research Centre (ECRC) is an example in this category. ECL^{i}PS^e provides various constraint solving libraries which can be used in different applications. ECL^{i}PS^e can store constraint variables and support multidimensional indexing, but it lacks an integrated constraint query language.

2. *Database centered view.* This approach extends the current database system (e.g., relational, object-oriented, deductive) by the notion of constraints. Such extensions attempt to preserve the current database system and provide an add-on constraint system [HCLM90, SK91]. It requires careful investigation of whether various database components could benefit from constraints, study on the impact of such an augmentation on other components, and introduction of new components capable of dealing with constraints efficiently. This approach finds its counterpart in object-relational database systems.

   The *CCUBE* system [BS97, BSCE97, BSCE99] is built on the object-oriented database system ObjectStore [LLOW91] and the CPLEX linear constraint solver [HJ88] to form a new system. It is designed to function as a generic platform for building systems such as Lyric [Bro95] or a constraint extension of an object-oriented query language. The CCUBE system relies on the Constraint Comprehensive Calculus instead of an algebra and an object-oriented database framework which has no uniform development standard [Edw98].

   The *DeCoR* system (*DEductive database system with COnstraints over Reals*) is a prototype of a constraint database system which is an extension of the deductive database system ProQuel [GM94]. It incorporates the constraint facilities on top of a commercial relational DBMS (Oracle). The DeCoR is aimed at solving diverse problems, particularly, temporal databases. The DeCoR system relies on a constraint calculus based on *Datalog* plus constraints. Datalog is a nonprocedural database query language derived from the logic-programming language Prolog. Though the DeCoR system, similar to the CCUBE system, uses a constraint calculus, it is a good reference for the system implementation.

   Dedale [GRSS97, GRS98, GRSS00] is a constraint database for spatial information. It is built upon an object-oriented database system, O_2. Data are stored in an object-oriented framework, with spatial data represented using linear constraints over $\mathbb{Q}^n$ (rational numbers) for some n. Dedale uses the

standard Object Query Language (OQL), with enhancement of special functions for constraint solving and geometric operations. Dedale is now used to run a geographical application from the French Institute for Geography. It depends on a linear constraint algebra. Dedale is another good reference for the system implementation though it builds on an object-oriented database system.

3. *Mixed view.* This approach builds a constraint database system from scratch. This approach avoids the risk of underestimating the complexity of the other corresponding part since each of the above two approaches depends on either database or constraint programming system. However, this approach certainly requires more effort. This approach also allows low-level query optimization techniques proposed in [BJM93].

   The *DISCO (Datalog with Integer and Set order COnstraints)* system [BR95, Rev97, Rev00b] is an example following this approach. DISCO provides a non-procedural, logic-based query language and allows uers to express the database inputs. DISCO also works by a translation to a procedural, algebraic language and a bottom-up evaluation that is guaranteed to terminate with some constraint database output. These features make DISCO applicable in various problems such as computer-aided design, scientific databases, and other areas where set-type data are used. However, DISCO relies on a constraint calculus and the type of constraints implemented in DISCO are integer gap-order and set-order constraints. These are not well-suited for the geographic application or other applications which rely on linear constraint algebra.

   MLPQ/GIS [RL97, KRW98, Pra98, RCKL00] is a constraint database system with a special emphases on spatio-temporal data. Features include data entry tools, icon-based queries, and Datalog queries. These query interfaces facilitate the database access; however, there is no direct integration with the conventional database system. It is a good reference for the implementation.

In addition, some connectivities which allow constraint programming platforms to access database systems have been built. For example, ECL^{i}PS^e has connectivity to Oracle and the Exodus storage manager [CDV88, Gra95]. The commercial CHIP (Constraint Handling in Prolog) system [DHSA88] can connect to Oracle. More constraint database systems and applications will be available.

2.3 Web Applications on CORBA/IIOP using Java

2.3.1 An Introduction to the Web

The Web is a distributed hypermedia information system proposed by Tim Berners-Lee at the European Laboratory for Particle Physics (CERN) near Geneva, Switzerland in 1989. The first Web server and browser then was created on a NeXT computer in 1990 [Gil97, NV96]. The main purpose of the Web is to provide an easy sharing of the information through the Internet or intranets where the Internet is the collection of global networks and an intranet is a network that is contained within an enterprise. The Web gained popularity since the appearance of the Web browser, Mosaic, which was created at the National Center for Supercomputer Applications (NCSA) in 1993. Since then various new Web applications are appearing. The Web is becoming a universal platform for running applications on the Internet and on intranets [Kha96].

HyperText Markup Language (HTML) is widely used as a document formatting language on the Web. HTML documents contain text, font specifications, and other formatting instructions. Links to other documents are specified by the *Universal Resource Locators (URLs)*. Most documents are accessed using *HyperText Transfer Protocol (HTTP)*, which is a protocol for transferring Web documents [Com95, Ber96, Dew97, Gil97, OHE96, OH98, SKS97]. HTML is a specific application of *Standard Generalized Markup Language (SGML)*, which is an international standard for the format of text and documents [Wig95, SKS97]. HTML can define only a single, inflexible document type. SGML is too complex for the Web. *Common Gateway Interface (CGI)* is designed to retrieve data on the Web server dynamically and

interactively [Ber96, NV96, Dew97]. It allows Web servers to execute a CGI program to respond the request from the client.

The Extensible Markup Language (XML) is designed to deliver SGML information over the Web while overcoming the limitations of HTML [Kha97, MFDG98, GQ99]. XML is a metalanguage to let Web users design their own markup language. XML is a simplified form of SGML which embraces the Web ethic [GP00]. XML has almost all of the capabilities of SGML but those that primarily affect document creation [Kha97]. The effort to develop effective techniques for database access through XML is presented in [ABS00].

Style sheet languages are used to describe how documents are presented on screens or in print. Style sheet languages separate structure and content from presentation [MFDG98, MFDG98]. *Cascading Style Sheets (CSS)* and *Extensible Stylesheet Language (XSL)* are two style sheet languages endorsed by *World Wide Web Consortium (W3C)* [Mey00]. While the combination of XML and style sheet languages increases flexibility, it also increases complexity and loses simplicity of original HTML proposals [Kor98].

JavaScript is a script language capable of defining functions in a Web document [Fla97]. JavaScript is standardized under the name *EcmaScript* [Dob98, Rul99]. *Document Object Model (DOM)* defines how HTML objects are exposed to the scripting language [Rul99]. *Dynamic HTML (DHTML)* is a term used to describe HTML pages with dynamic content [Dob97, MFDG98], which contains CSS, HTML, and a scripting language under the umbrella of DOM [Del99].

2.3.2 CORBA/IIOP

CORBA (Common Object Request Broker Architecture) is an industry-wide standard for creating distributed object systems [PWGB98, Bal00]. It is a middleware project undertaken by *Object Management Group (OMG)* which is a consortium of computer industry companies [Ber96, OHE97b, OH98]. CORBA is the heart of the OMG's architectural framework, the *Object Management Architecture (OMA)*.

CORBA is the OMG's answer to the need of interoperability between software written in different languages or on heterogeneous hardware platforms [PWGB98].

The central component of CORBA is the *Object Request Broker (ORB)*. It encompasses all of the communication infrastructure necessary to identify and locate objects, handle connection management, and deliver data. In general, the ORB is not required to be a single component; it is simply defined by its interfaces. The ORB Core is the most crucial part of the Object Request Broker; it is responsible for communication of requests. *Internet Inter-ORB Protocol (IIOP)* is to specify ORB interoperability [PWGB98, OHE97b, OH98, VD98]. IIOP is basically TCP/IP with some CORBA-defined message exchanges that serve as a common backbone protocol [Bal00]. ORB *Interface Definition Language (IDL)* is the language used to specify interfaces of objects independent of particular programming language representations. To assist object implementation *CORBAservices* for fundamental services and *CORBAfacilities* for application-level services are published.

Generally speaking, CORBA has the following benefits [Ber96, Bak97, PWGB98, Sie00]:

- Interoperability on heterogeneous computing environments.
- Scalability.
- Rapid application development.
- Code reuse.
- Separation of interface and implementation.

Java's inherent platform independence conveniently facilitates software reuse and portability in heterogeneous environments. CORBA's language-neutral approach to object interface specification allows objects to interoperate without respect to any specific implementation language. Java and CORBA, taken together, provide an architecture for developing highly resuable and portable distributed software systems [PWGB98].

2.3.3 The Object Web

The progression of Web technologies from simple document fetching to more complex, interactive applications followed several stages [OHE97b, OHE97a, VD98]:

- Fetching electronic documents published on the Web sites.
- Building interactive systems using CGI.
- Using Java applets to provide client-side functionality.
- Distributed computing through the association of Java and CORBA.

The HTTP/CGI paradigm has several drawbacks [OH98, OHE97b, OHE97a, VD98]. The various CGI extensions from different vendors simply lead to a totally incompatible server Web. The real solution is the association of distributed objects on the Web. One example is augmenting the Web infrastructure with CORBA/Java. This provides several benefits as described in [OH98, OHE97b, OHE97a, VD98, ZL00]:

- CORBA extends Java with a distributed object infrastructure.
- CORBA provides a scalable server-to-server infrastructure.
- CORBA avoids the CGI bottleneck and preserves state.

This trend of Internet innovation is called the **Object Web** [OH98, OHE97b, OHE97a] as shown in Figure 2.2.

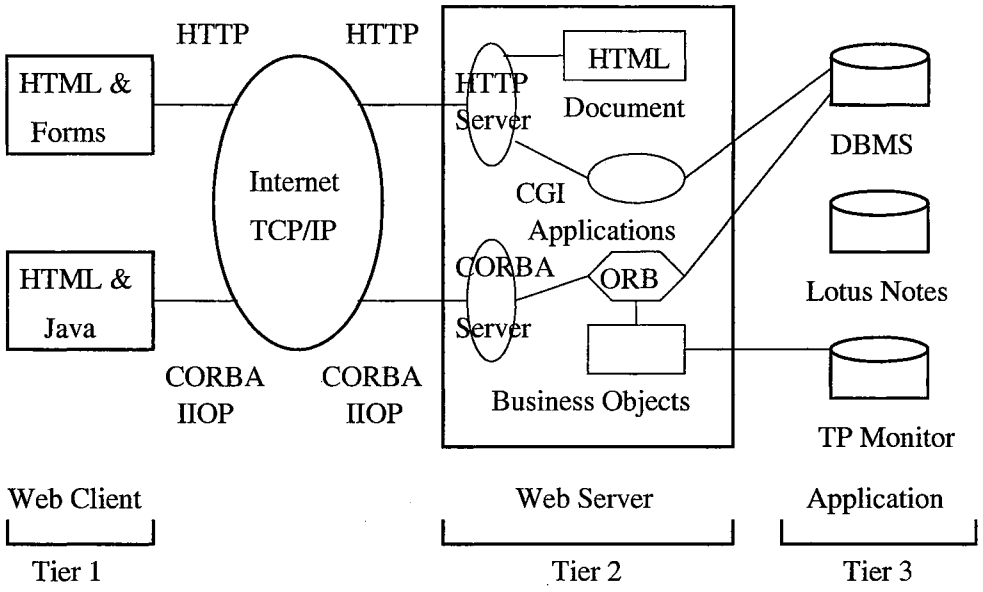


Figure 2.2 An Object Web Model Based on Java and CORBA ORBs [OHE97b].

The Web has been evolving in a fast pace. Emerging technologies are changing the face of the computing environment.

CHAPTER III

WEB-BASED APPLICATIONS

3.1 A Web-Based Tax Calculator

To motivate the implementation of a constraint database system in a distributed computing environment, we consider the design of an online income tax estimator. The purpose of this application is threefold. First, it shows how to build an application in a distributed computing environment. Second, it presents the ability of constraint databases to handle the data involving arithmetical computation. Finally, it reflects the necessity to integrate data represented by constraints with data represented using typical relations.

Generally speaking, the tax is a piecewise linear function of the income as shown in Figure 3.1.

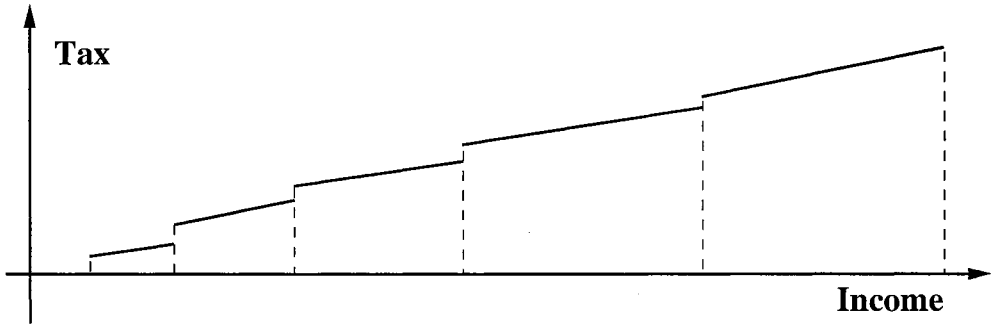


Figure 3.1 Dependency between Income and Tax

3.1.1 Creating the Constraint Database

The relation between the income and the corresponding tax can be modeled in different ways. To build up a constraint database we need to separate the data representable by the relational model from data represented by constraints. In this

example, we create a *Tax* database which contains a *taxpayer* table for holding the basic information of a taxpayer and a *deduction* table for the standard deduction and exemption. Table 3.1 and Table 3.2 are typical tables in a relational database:

Tax ID	First Name	Last Name	Income
123456789	Jane	Sebring	51720.51
102347852	Al	Goodwin	42081.23
151678001	Dan	Talley	31507.75
142693100	Mike	Johnson	33720.69

Table 3.1 Taxpayer Table

Type	Exemption	deduction
S	4400	2800
H	6450	2800
M	7350	2800
P	3675	2800

Table 3.2 Deduction Table

In Table 3.2 the *type* attribute indicates the filing status of the taxpayer where **S**, **H**, **M**, and **P** stand for single, head of household, married filing jointly, and married filing separately, respectively. The other table in this application is the tax rate schedule. Table 3.3 is the 2000 tax rate schedule for the single taxpayer from the Internal Revenue Service:

Over	But not Over		+%	on amount over
0	26,250		15	0
26,250	63,550	3,937.50	28	26,250
63,550	132,600	14,381.50	31	63,550
132,600	288,350	35,787.00	36	132,600
288,350		91,857.00	39.6	288,350

Table 3.3 Tax Rate Schedule

The tax rate schedule can be represented using the following linear constraints:

$$[r = 0.15 \wedge 0 \leq x < 26250] \vee$$
$$[r = 0.28 \wedge 26250 \leq x < 63550] \vee$$
$$[r = 0.31 \wedge 63550 \leq x < 132600] \vee$$

$$[r = 0.36 \wedge 132600 \leq x < 288350] \vee$$
$$[r = 0.396 \wedge 288350 \leq x]$$

where r is the tax rate and x is the income. We can simplify it and form the *tax table* as shown in Table 3.4.

x	θ	Income	Base Tax	Rate	Type
1	>	0.00	0.00	0.15	S
1	>	26250.00	3937.50	0.28	S
1	>	63550.00	14381.50	0.31	S
1	>	132600.00	35787.00	0.36	S
1	>	288350.00	91857.00	0.396	S

Table 3.4 Tax Table

x attribute contains the coefficient of the constraint. θ is used to store the operations in constraints. *Income* is the *right hand side (RHS)* of the equation. In this way we can store constraints in a relational database. In general, there are four comparison operations: $>$, $\geq$, $<$, and $\leq$. When we do the query in the database, θ can be used as an indication of the data range. The symbol $>$ indicates that the tax rate occurs when the income is more than the specified value. In this application storing constraints in a relational database is straightforward. We can do it without any mechanic operation. We can omit x attribute from our table in the actual implementation because the coefficients is same. One-dimensional linear constraints can store into a relational database in this straightforward way. For more complex applications manual transformation is not sufficient. Then we need a conversion program which will be illustrated in the later applications.

To calculate the tax, we first query the matching rate. This procedure is called *constraint satisfaction.* This function is used frequently for constraint solving. It simply scans the tuples of the relation and returns the *true* as soon as it finds a satisfiable tuple. Then we calculate the taxable income by subtracting the deduction and exemption from the gross income. Finally, we multiply the rate with the taxable income over the base earnings (RHS) plus the base tax to get the tax. Here is the equation to calculate the tax: $tax = base\ tax + rate \times (taxable\ income - RHS)$. We

use the sequential search to find the satisfiable tax rate. When the sequential search is not sufficiently fast in a large table, we need some other efficient search or index methods.

3.1.2 Developing the CORBA Application

Four components are required to build up the application:

- a Web server - *Apache HTTP server.*
- a Java programming environment - *SUN Java 2 SDK.*
- a CORBA compliant Object Request Broker - *Object Oriented Concept Orbacus* [Obj00].
- a relational database system - *Hughes Technologies mSQL.*

The choice of the software in our implementation is based on their usability and availability. It is only a reference. Any similar products should suffice the application. The Web browser is used to access the Java applets which are client parts of the application. CORBA classes are used by those Java applets to communicate across client and server. To access the application from the Web browser we need to put Java applets and CORBA classes in a place accessible by the Web server.

A CORBA object is an object with an interface defined in CORBA IDL. CORBA objects have different representations in clients and servers [Obj00].

- A *server* implements a CORBA object in a concrete programming language, for example in Java or C++. An implementation class based on the skeletons produced by IDL compiler should be written and then instantiated. This object implementation is called a *servant.*
- A *client* that utilizes a servant implemented by a server creates an object that invokes all operations to the servant via the ORB. Such an object is called a *proxy* or *stub.*

When a client invokes a method on the local proxy object, the ORB packs the input parameters and sends them to the server, which in turn unpacks these parameters and invokes the actual method on the servant. Any output parameter and return value follow the reverse path back to the client. From the client's perspective, access to the remote objects is like access to the local objects because all the communication details are hidden within the proxy object.

To allow the ORB to invoke a method on the servant when a request is received from a client, a servant must connect (bind) to the ORB. This connection is handled by the *Portable Object Adapter (POA)*, as shown in Figure 3.2 [Bal00, ZL00].

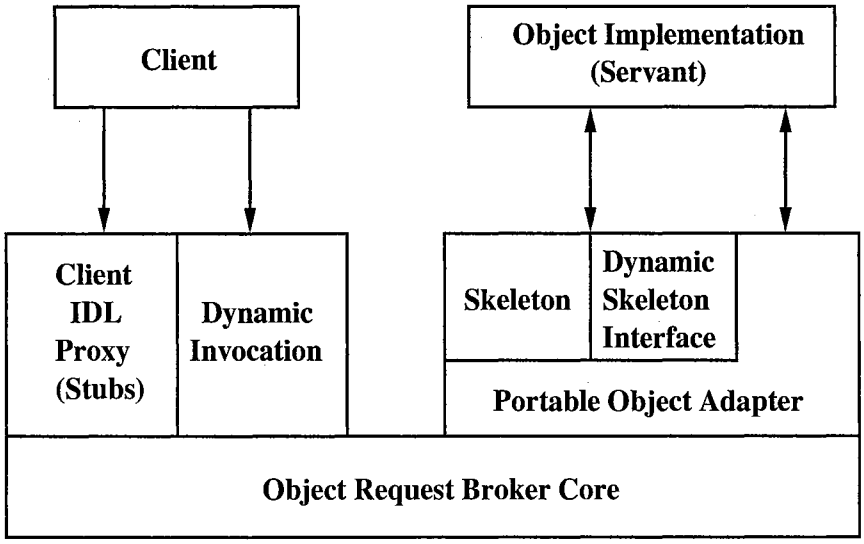


Figure 3.2 ORB Interface

The Portable Object Adapter was approved by the OMG to replace the deprecated *Basic Object Adapter (BOA)*. The POA is a far more powerful and flexible object adapter than the BOA. The POA was defined to support a portable implementation among different vendors. A server can use many POAs to support different policies. It includes a number of features that are essential for building scalable and high performance servers [Bal00, ZL00].

To develop a CORBA application we follow the steps as shown in Figure 3.3 [OH98, PWGB98, VD98, DCR99]:

1. Identify the objects used in an application: object analysis and design.

2. Specify the objects and their interfaces in an IDL file.

3. Compile the IDL file. The IDL compiler will generate client stub code and server skeleton code in the specified language. The IDL-to-Java compiler is used in our application.

4. Code the servers. Implement server objects whose interfaces are defined in the IDL file.

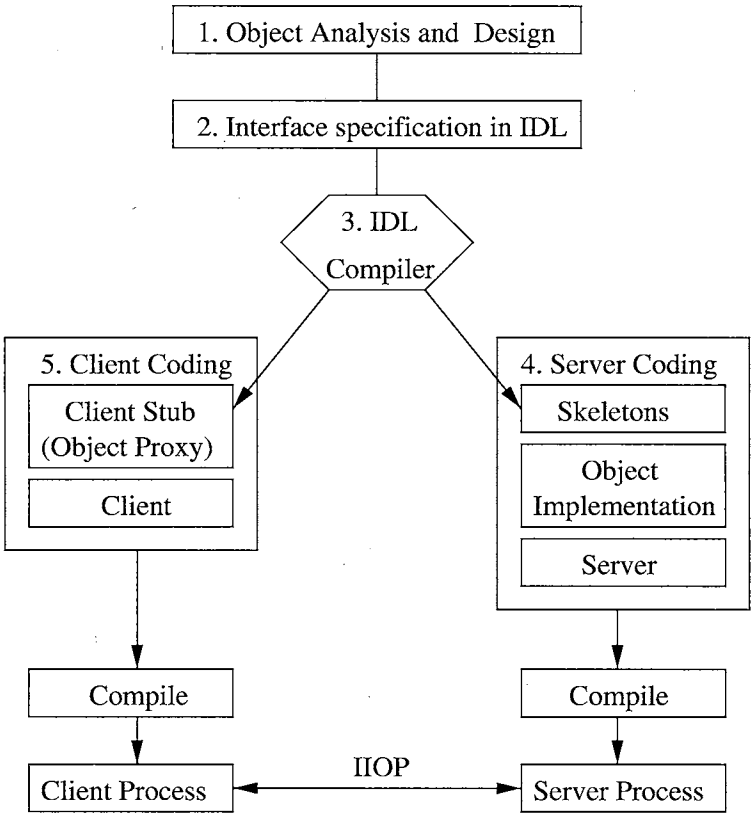


Figure 3.3 Create a CORBA application

5. Code the clients. Integrate stub codes to complete client design.

The object-oriented design is a basic concept in CORBA program development. An IDL file defines all required objects. A good design starts from a good object design in an IDL file. The IDL file of our tax application is shown as follows:

```
module Tax {
```

```
  // Define a taxpayer.
  struct Taxpayer {
    string tax_id;
    string first_name;
    string last_name;
    double income;
  };

  // Define the tax rate.
  struct Taxrate {
    char  theta;
    float income;
    float base_tax;
    float rate;
    char  type;
  };

  // Define the tax deduction.
  struct Taxdeduction {
    char type;
    float deduction;
    float exemption;
  };

  // Define the tax table.
  typedef sequence<Taxrate> TaxrateSeq;

  // Declare TaxManager interface.
  interface TaxManager {
    // Open/Get the information of the taxpayer.
    Taxpayer open(in Taxpayer tax_payer);
    // Update the income.
    Taxpayer updateincome(in Taxpayer tax_payer, in double amount);
    // Return the tax table.
    TaxrateSeq taxtable(in char type);
    // Return the tax deduction.
    Taxdeduction deduction(in char type);
  };
};
```

`Taxpayer` defines the data structure of a taxpayer which includes the tax ID, the first name, the last name, and the income. `Taxpayer` corresponds to the *taxpayer* table in our database. `Taxrate` defines an entry in our tax table. It contains `theta`

which is a mathematical operation, the income, the base tax, the tax rate, and the filing type. `TaxrateSeq` defines a sequence of `Taxrate`. This sequence corresponds to the *taxtable* table in the database. `Taxdeduction` is composed of the filing type, the deduction, and the exemption. `Taxdeduction` corresponds to the *taxdeduction* table in the database. `TaxManager` interface is defined in the IDL file. It contains four functions: open an account, update income, return the tax table, and show tax return. With these three data structures and one interface we can build up our application.

3.1.3 Running the Application

The application is built in three tiers: client, server, and middleware. In this example the client is a Java applet, the server is a database system, and the middleware is a HTTP/CORBA combination. The Java applet is an interface for users to initiate the requests to the CORBA objects. The CORBA objects then address the requests to the database system. The response from the database system goes back to the Java applet through requesting CORBA objects. The HTTP server is used to transfer Web contents.

Figure 3.4 shows the interaction between a Web-based client and its server on Object Web [OH98]:

1. Web browser loads HTML page.
2. Web browser retrieves Java applet from the HTTP server.
3. Web browser loads applet.
4. Applet invokes CORBA server objects.
5. Server objects respond to the requests or generate the next page.

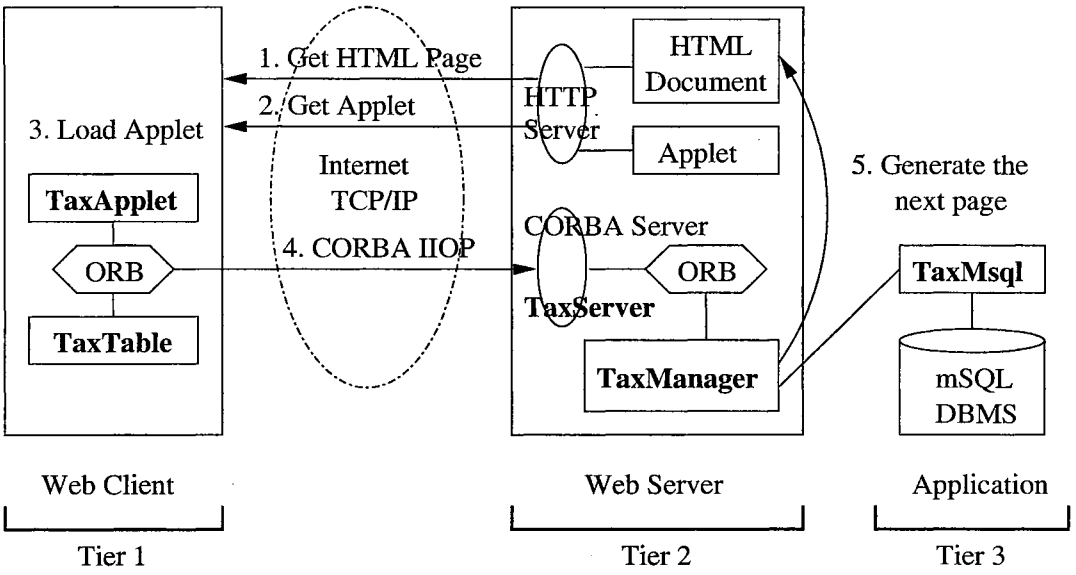


Figure 3.4 Running the Tax application

In the tax application, two Java applets are used to access the database system. **TaxApplet** is an interactive interface for users to access the account information. **TaxTable** retrieves the tax table from the database and displays it. **TaxServer** initiates a **TaxManager** CORBA object awaiting the incoming requests. **TaxManager** sends the requests to **TaxMsql** which finally sends the requests to the database. The reply from the database system goes through **TaxMsql** and **TaxManager** object and reaches the client applets. To separate **TaxMsql** and **TaxManager** has the following benefits:

- Each program is a functional module. **TaxManager** is a CORBA object. **TaxMsql** is dealing with the database system.
- The application code does not depend on the choice of a database system. We can connect the database through the APIs of the database system, JDBC, or ODBC without changing CORBA function calls.

Figure 3.5 shows the running example of the tax application.

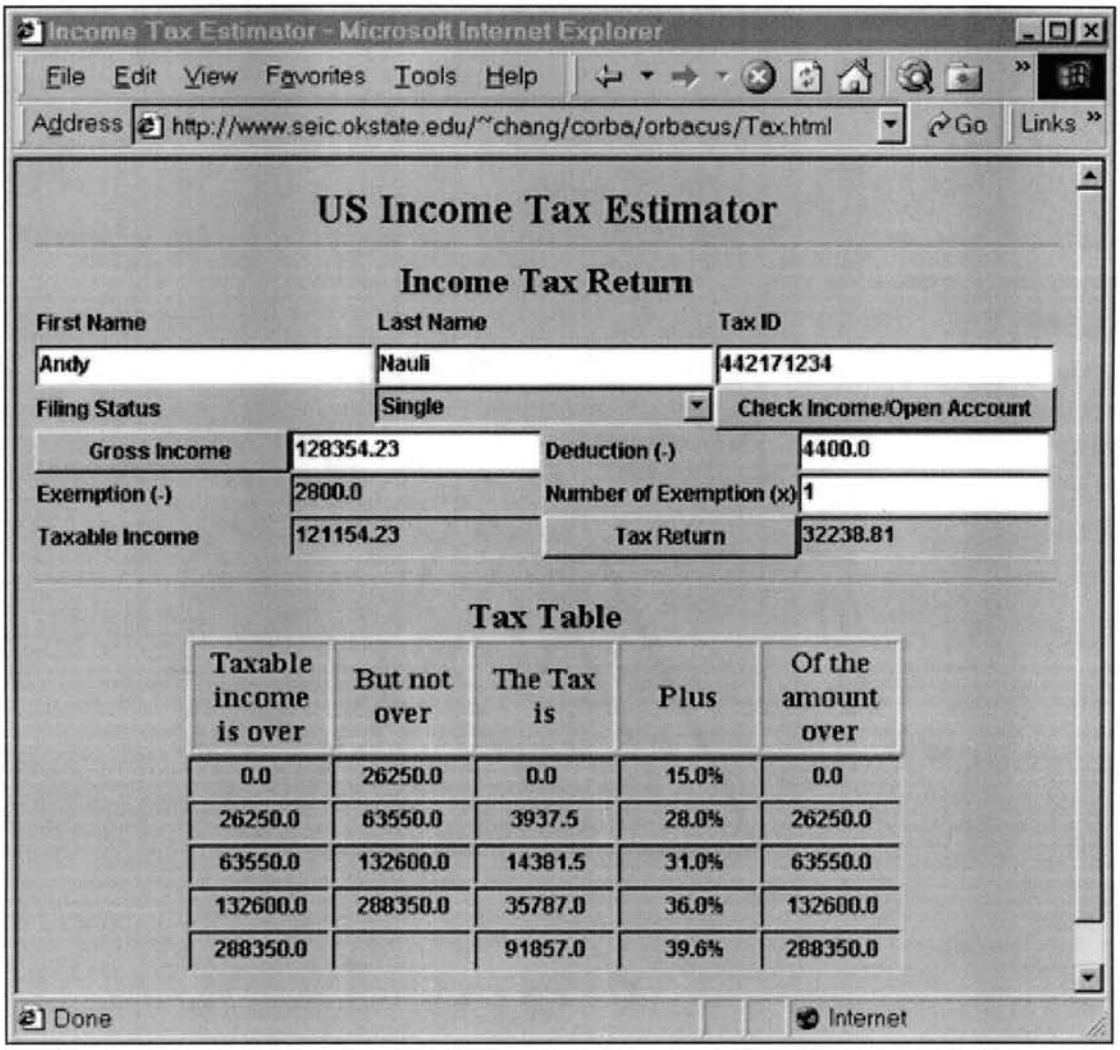


Figure 3.5 The Running Example of the Tax Calculation

3.2 An Interactive Stock Analysis Tool

With the proliferation of the electronic commerce various applications are targeted to facilitate the usage of the electronic business on the Internet. In this section, we device a stock analysis tool which is accessible through the Internet. We provide users with an interactive tool in CORBA distributed computing environment. It retrieves main financial data from COMPUSTAT stock and business database [Com97] and does useful analysis. In addition, we show how to store more complex constraint data into a relational database system in CORBA distributed computing environment.

Our data set is from COMPUSTAT Database. It is the most complete and

current database of U.S. and Canadian public companies and indexes available today [Com97]. In particular, we focus on S&P 500 companies. We build this application for two purposes. Firstly, it shows how a more complex application can be built in a distributed computing environment. Secondly, it presents possibility to store more complex constraint data into a relational database system.

The portion of the data set we are concerned with is the *year number*, the *earnings per share*, the *stock low*, the *stock high*, and the *stock value at year close* for the last ten years. We also include the basic information such as the *company name*, the *ticker*, the *address*, the *phone number*, the *industry classification*, and the *employee numbers*. The company basic information and the financial data set form two tables in the database we use.

To do a stock analysis we focus on two queries. One query retrieves the companies which have a price-earnings (P/E) ratio less or more than some constant. The P/E ratio is an important factor to consider a stock. A high P/E ratio indicates less profit or high investment [BM98]. On the contrary a low P/E ratio indicates high profit or less investment. A negative P/E ratio means the company is in debt. Investors choose stocks based on their preferences. Another measure is a *stock's volatility* which considers a stock whose value didn't vary by more than some percent of the price at year's end [BM98, GRSY96]. Volatility is a measure of riskiness about the future changes in the prices of assets and other economic variables. It is a fundamental parameter used to quantify risk in modern finance theory, and it is crucial input for virtually all decisions relating to risk management and strategic financial planning [BM98, CM88, Hot97, TB98, HW98].

3.2.1 Creating the Constraint Database

The database for the stock application is described in this section. The *company* and *finance* table are created in this database. The *company* table contains the basic information of a company.

An example of the *company* table is shown in Table 3.5.

Name	CISCO SYSTEMS INC
Ticker	CSCO
Address	170 W TASMAN DR
City	SAN JOSE
State	CA
Zip Code	95134-1706
Phone	408-526-4000
Industry	COMPUTER COMMUNICATION EQUIP
Exchange	NASDAQ
Employees	15000
Group	S&P 500, S&P Industrials, Fortune 500

Table 3.5 Company Table

The *company* table is a table which can be represented in a relational model. The other table in this application is the *finance* table which contains important financial parameters of a company. We define a constraint between the low price and the high price to indicate the share price is floating in that range. An example of the table is shown in Table 3.6:

Ticker	Year	EPS	Low Price	θ	High Price	Close Price	P/E Ratio
MSFT	98	0.835	31.094	$<$	72.000	69.343	59.222
MSFT	97	0.658	20.188	$<$	37.688	32.313	48.052
CSCO	98	0.420	17.167	$<$	48.875	46.406	72.539
CSCO	97	0.336	10.056	$<$	20.194	18.583	52.342

Table 3.6 Finance Table

The θ attribute is used to store the constraint operation. The symbol $<$ indicates that the share price is between the low price and the high price. In this application storing constraints in a relational database is straightforward as shown in the tax application. So no conversion program is needed. In this application θ is more symbolic than practical. This is a degenerated case of linear constraints. For more complex applications we need a conversion program.

For this data set, we explore four queries. The first query is: *Retrieve the company which has the specified ticker.* The second query is: *Find the ticker of a company.* The third one is: *Retrieve all tuples which have a price-earnings (P/E) ratio less or more than some constant.* The fourth query is a measure of a *stock's volatility*

which contains the third query and *Retrieve all tuples that correspond to stocks whose value didn't vary by more than some percentage of the price at year's end.* In terms of linear constraints it is $price_{high} - price_{low} - p \times price_{year-end} \leq 0$ where p is the percentage mentioned above. The sequential search used in the tax application applies to find the stock which meets this condition (constraint satisfying). When the table becomes large, the sequential search is not sufficiently fast, and efficient search or index methods is required.

3.2.2 Developing the CORBA Application

The system requirements and the procedure to develop this application are similar to the tax application. The Java applets access the remote database through the CORBA objects which reside on the Web. To develop a CORBA application we follow the steps as shown in Figure 3.2. The IDL file defines all required objects. A good IDL design is necessary for a good program design. The IDL file of our stock application is shown as follows:

```
module Invest {

  // Define a company.
  struct Company {
    string name;
    string ticker;
    string address;
    string city;
    string state;
    string zipcode;
    string phone;
    string industry;
    string exchange;
    unsigned long employees;
    string group;
  };

  // Define the finance information.
  struct Finance {
    string ticker;
    unsigned short year;
```

```
    float eps;
    float low_price;
    char  theta;
    float high_price;
    float close_price;
    float pe_ratio;
  };

  // Define the Company and Finance sequence.
  typedef sequence<Company> CompanySeq;
  typedef sequence<Finance> FinanceSeq;

  // Declare the Portfolio interface.
  interface Portfolio {
    // Get the company information.
    CompanySeq company(in string ticker);
    // Get the finance information.
    FinanceSeq finance(in string ticker, in unsigned short year);
    // Get the ticker by company name.
    string getTicker(in string company_name);
    // Get stocks by P/E.
    FinanceSeq pickStock(in unsigned short year, in char op,
                         in float pe_ratio);
  };
};
```

`Invest` is a module defining all objects and interfaces we use in this application. We create the *Invest* database for holding two data structures defined in the `Invest` module. `Company` is a data structure describing a company. It corresponds to the *company* table in the database. `Finance` defines main financial parameters of a company. It corresponds to the *finance* table in the database. `CompanySeq` and `FinanceSeq` define the sequence for `Company` and `Finance` respectively. `Portfolio` interface contains four functions. `company` returns the company information. `finance` shows the finance information of a company. `getTicker` searches the ticker of a company. `pickStock` picks the stocks which meet the specified criteria.

3.2.3 Running the Application

The application comprises three tiers: client, server, and middleware. Java applets, the database system, and the HTTP/CORBA combination play the same role as in the tax application.

Figure 3.6 shows the interaction between a Web-based client and its server on the Object Web [OH98].

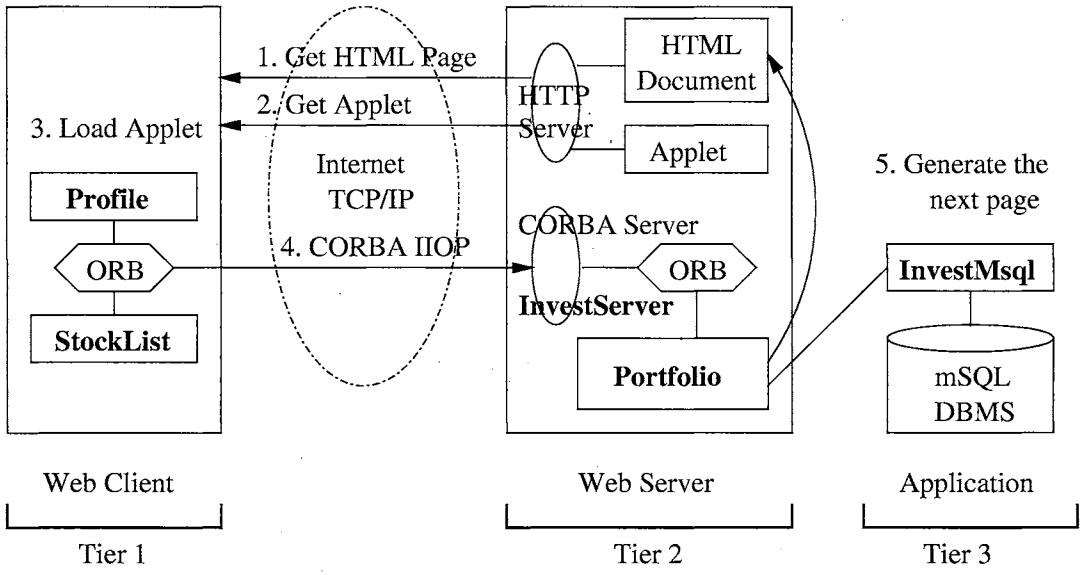


Figure 3.6 Running the Stock Application

In this application, **Invest** is the main page which includes **Quote**, **Profile**, and **PickStock**. **Quote** is the upper part of the main page. It is an interface to user's requests. According to different requests either **Profile** or **PickStock** page is loaded. Four actions are available on the **Quote** page: **Find Ticker**, **Company Profile**, **P/E Ratio**, and **Volatility**. For **Find Ticker** and **Company Profile** the **Profile** page is selected. For **P/E Ratio** and **Volatility PickStock** is chosen. We use functions in *JavaScript* to designate the requested page. **Profile** contains the **Profile** applet. **PickStock** includes the **Stocklist** applet. The **Profile** applet retrieves the company's profile from the database and displays it. **Stocklist** retrieves the finance information of companies which meet the criteria. **Portfolio** is a CORBA object awaiting the incoming requests. **Portfolio** sends the requests to **InvestMsql** which

finally sends the requests to the database. The reply from the database system goes through **InvestMsql** and **Portfolio** object and back to the client applets. **InvestMsql** and **Portfolio** are two separate modules which has the same benefits as in the tax application.

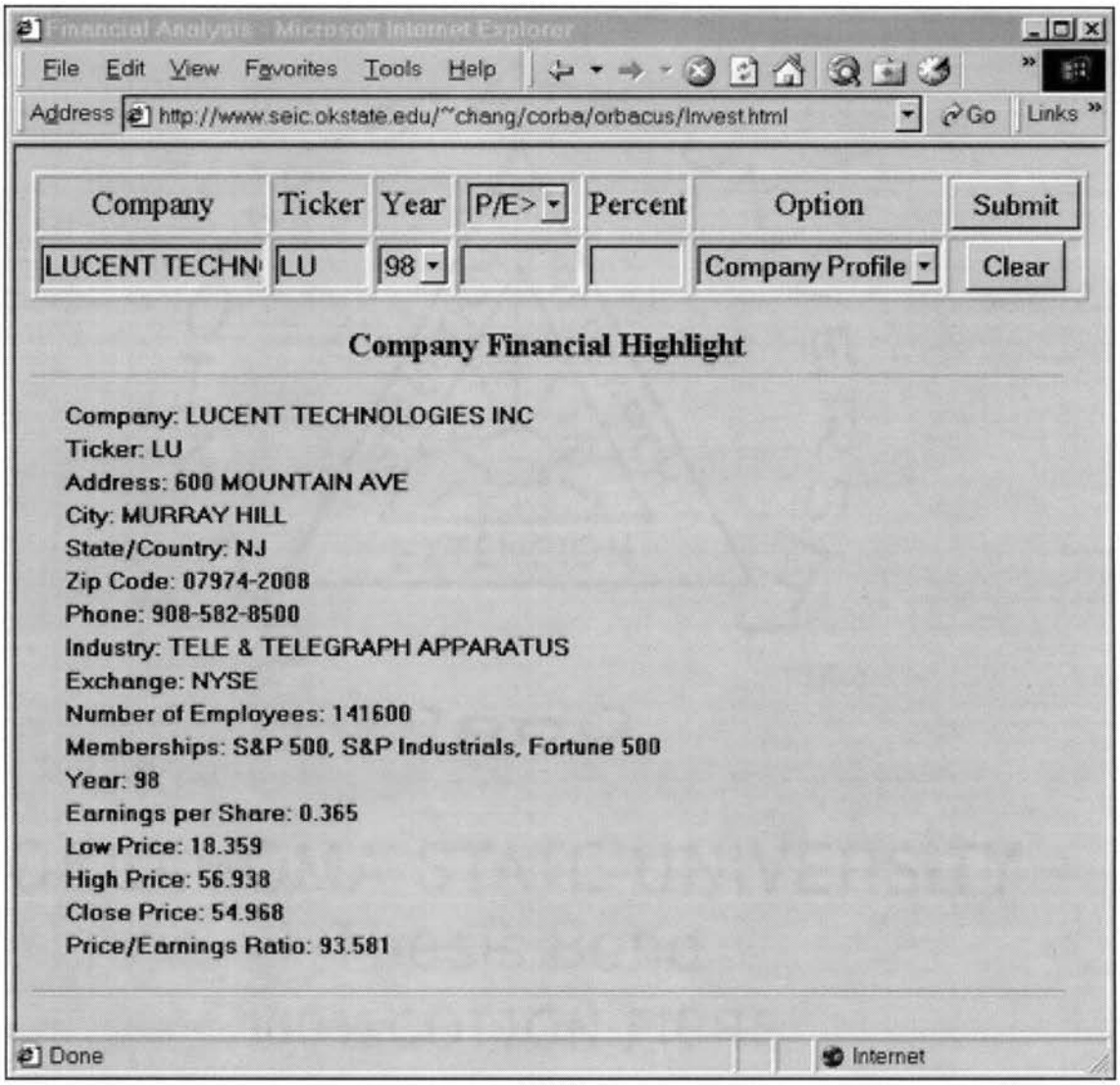


Figure 3.7 The Running Example of the Ticker Finder

Figure 3.7 shows the running example of the application in case of the company profile. Figure 3.8 shows the running example of the application in case of the volatility analysis.

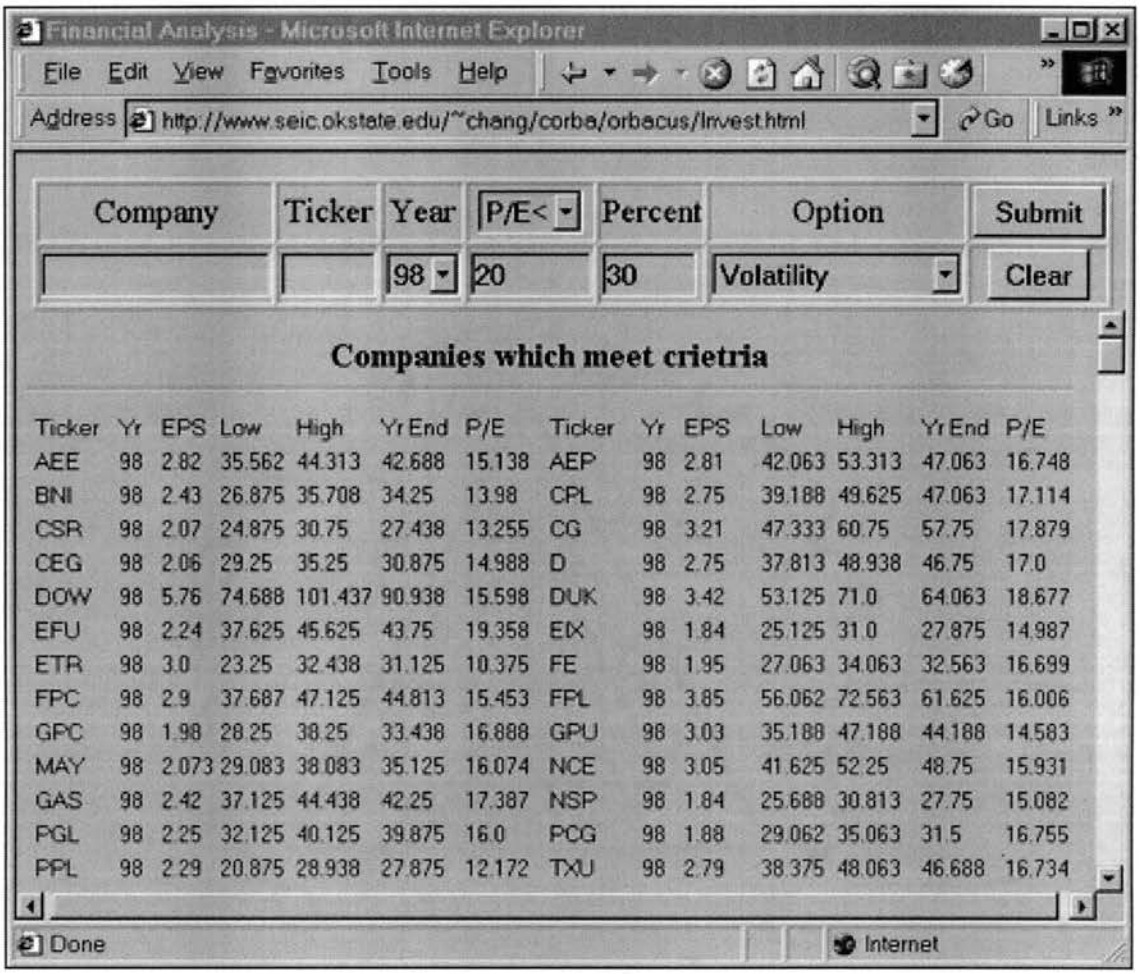


Figure 3.8 The Running Example of the Volatility Analysis

3.3 Summary

In this chapter, we present Web-based applications on CORBA and illustrate the method to create constraint databases for these applications. We simply store constraints into a relational database system and use the query commands available for the relational database system. This method fits well in one-dimensional constraints but for more complex data the conversion program is required. This will be presented in the next chapter.

CHAPTER IV

IMPLEMENTATION OF A QUERY INTERFACE

4.1 Constraints in a Relational Database

When we design a constraint database system, the index structure for the relational database is no longer sufficient. Some data structures have been developed for storing constraints [KRVV93, KRVV96, Fre95, Ram97, Ram00]. Most of them lack the integration with the index structure for the relational data model. In the previous chapter we propose a method to store constraints into a relational database system without extra index structure. When we extend this idea to two or more dimensions, we need certain conversion program which is the main focus of this chapter.

We use an example to illustrate how to build such a constraint database system. The application of this database is a geographic information system. The *database-table* table contains the feature information of a list of tuples. These tuples have the same geospatial features. The *database-field* table defines the data attributes for the tuple. The *database-data* table contains the alphanumeric data of the tuple. The *graph-specification* table specifies the required information for drawing a spatial object in the system. The *constraint* table comprises the constraints which define a spatial object. An example of the tables is shown from Table 4.1 to Table 4.5.

Table-ID	Description	Color	Tupl-Link
0	Roads	-8355712	1
1	Rivers	-16776961	2
2	Lakes	-16732162	3

Table 4.1 database-table

Field-ID	Name	Type
0	Spatial-ID	0
1	Owner	0

Table 4.2 database-field

Tuple-ID	Spatial-ID	Owner
0	Main Street	Stillwater
1	Duck Street	Stillwater
2	Stillwater Creek	Stillwater
3	Boomer Lake	Stillwater

Table 4.3 tuple-data

Graph-ID	Tuple-ID	Fill	Color
0	0	Y	-8355712
1	1	N	-16732162
2	2	N	-16776961
3	3	N	-16732162

Table 4.4 graph-specification

Graph-ID	x	y	θ	RHS
0	0	1	$=$	97
0	1	0	$\geq$	9
0	1	0	$\leq$	246

Table 4.5 constraint

The logical structure of the database among these tables is depicted in Figure 4.1. The Entity-Relationship diagram in Figure 4.1 consists of three entity sets, *database-table*, *tuple-data*, and *constraint*, related through two binary relationship sets *table-tuple* and *graph-specification* respectively. The attributes associated with *database-table* are *Table-ID*, *Description*, *Color*, and *Tuple-Link*. The attributes associated with *tuple-data* are *Tuple-ID*, *Spatial-ID*, and *Owner* which are specified in *database-field*. *database-field* has three attributes *Field-ID*, *Name*, and *Type*. This facilitates different database designs for different applications. *graph-specification* adds two attributes *Fill* and *Color* to relate *tuple-data* and *constraint*. The *constraint* table

has five attributes including *Graph-ID*, x, y, θ, and *RHS* which define a constraint. From the diagram we see that the relationship set *table-tuple* and *graph-specification* are one to many.

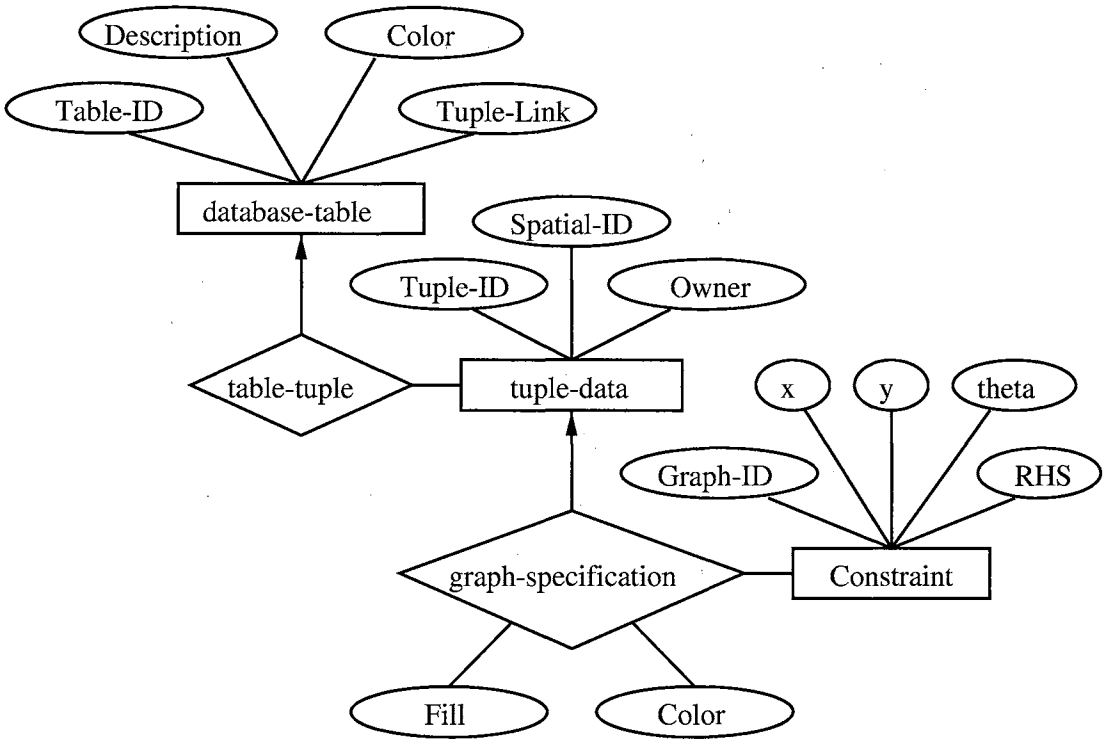


Figure 4.1 Entity-Relationship Diagram of the Database

Database Table	Data Structure Class	IDL Object
database-table	GeoTable	GeoTable
database-field	DBField	DBField
tuple-data	String[ ]	Tuple
graph-specification	GeoObject	GraphSpec
constraint	Constraint	Constraint

Table 4.6 Database Table to Data Structure

These tables have their corresponding data structures in the program as shown in Table 4.6. Basically, the data structure class contains the same data fields as its corresponding table in the database and its implementation method. For example, the *GeoTable* class is defined as follows:

```
public class GeoTable {
  String description;
```

```
  Color color;
  int tuplelink;

  public GeoTable(String description, Color color, int tuplelink) {
    this.description = description;
    this.color = color;
    this.tuplelink = tuplelink;
  }

  public void setDescription(String description) {
    this.description = description;
  }

  public void setColor (Color color) {
    this.color = color;
  }

  public void setLink(int tuplelink) {
    this.tuplelink = tuplelink;
  }
}
```

4.2 Algorithms

The use of constraints to model spatial data does not come for free. In fact, the vector format is often used to represent spatial data. It is identified by the vertices of the spatial object. For example, when we draw a polygon using the *Java* graphic library, the parameters we give are the vertices of the polygon. Same situation happens in ARC/INFO. The conversion from the vector format to linear constraints or from linear constraints to the vector format has time complexity of $O(n\ log\ n)$, where n is the number of points used to represent a given spatial object. This cost comes from decomposing the spatial object into convex hulls [PS85]. The main algorithms required in the system design are presented in this section.

4.2.1 Convert Points and Line Segments into Linear Constraints

The translation of points and line segments is straightforward.

Given a point (x_0, y_0), we can get the following linear constraints directly: $x = x_0, y = y_0$.

Given the two end points of a line segment: $(x_0, y_0), (x_1, y_1)$, the following algorithm can get the linear constraints representation of the line.

Input: Two end points of a line segment $(x_0, y_0), (x_1, y_1)$.

Output: The linear constraint representation of the line.

Algorithm 4.1 Line to Constraints

LinetoConstraint (L)
// L is the line with 2 different end points: $L_0 = (x_0, y_0), L_1 = (x_1, y_1)$. //
if $(x_0 \neq x_1)$

then return $\left\{ \begin{array}{l} (y_1 - y_0)x - (x_1 - x_0)y = (y_1 - y_0)x_0 - (x_1 - x_0)y_0, \\ x \geq min(x_0, x_1), \\ x \leq max(x_0, x_1). \end{array} \right\}$

else return $\left\{ \begin{array}{l} (y_1 - y_0)x - (x_1 - x_0)y = (y_1 - y_0)x_0 - (x_1 - x_0)y_0, \\ y \geq min(y_0, y_1), \\ y \leq max(y_0, y_1). \end{array} \right\}$

end

For example, we can use two end points (100, 100) and (200, 300) to represent a street. Using the above algorithm, we get the linear constraints representation of the street as follows:

$$\left\{ \begin{array}{l} 2x - y = 100, \\ x \geq 100, \\ x \leq 200. \end{array} \right\}$$

The above technique can expand to the spatio-temporal case shown in following:

Input: Two end points of a line segment $(x_0, y_0), (x_1, y_1)$. The departure time of the first point is t_0. The arriving time of the second point is t_1. The specified time is t where $t_0 \leq t \leq t_1$.

Output: The linear constraint representation of the location at time t.

Algorithm 4.2 Location of a Moving Object to Constraints

LocationtoConstraint (L, T)
// L is the line with 2 different end points: $L_0 = (x_0, y_0), L_1 = (x_1, y_1)$. //
// T contains the initial time t_0 and final time t_1. Specified time is t. //

$$\textbf{return} \left\{ \begin{array}{l} x = x_0 + \frac{(x_1 - x_0)}{(t_1 - t_0)} \times (t - t_0), \\ y = y_0 + \frac{(y_1 - y_0)}{(t_1 - t_0)} \times (t - t_0), \\ t \geq t_0, \\ t \leq t_1. \end{array} \right\}$$

For example, we drive a car from (100, 100) at 10 minutes after 5 and arrive (200, 300) at 20 minutes after 5. Using the above algorithm, we get the linear constraints representation of the location at t minutes after 5: $\left\{ \begin{array}{l} x = 10t, \\ y = 20t, \\ t \geq 10, \\ t \leq 20. \end{array} \right\}$

4.2.2 Triangulation Algorithm

Most spatial objects are depicted as polygons. Converting a polygon to its constraint format needs extra effort. Because each polygon may not be a convex hull polygon, it is not efficient to represent the polygon directly using linear constraints. But it is feasible to decompose each polygon into a set of triangles where each triangle is represented by its three corners. After triangulation, it is effective to represent each triangle using linear constraint model (See the Algorithm 4.4 in the next section). In this way each polygon can be represented by linear constraints.

Polygon triangulation is a fundamental algorithm in computational geometry. Triangulations form a huge subject in mathematics because of its wide applications. Classical applications of triangulation include computer graphics and finite element analysis. Methods of triangulation include greedy algorithms [O'R98, PS85], convex hull differences [TM84], and horizontal decompositions [Sei91].

The triangulation algorithm (Algorithm 4.3) used in the system implementation is adapted from the $O(n^2)$ algorithm by Joseph O'Rourke [O'R98]. It is sufficiently fast in our application. But to triangulate a polygon with a large number of vertices please

refer to Atul Narkhede's implementation [Nar94]. It is based on the Raimund Seidel's incremental randomized algorithm and extended to handle holes. The algorithm's expected complexity is $O(n\ log^*\ n)$. In practice, it is almost linear time for a simple polygon having n vertices. The triangulation introduces no additional vertex and divides the polygon into $n-2$ triangles.

Input: The corner points of a polygon.

Output: A set of triangles represented by their corner points.

Algorithm 4.3 Triangulation

```
Triangulate (P) // P contains vertices of a polygon. //
  n ← number of vertices of P
  T ← Triangle [n − 2]
  i ← 0
  while n > 3 do
    k ← −1
    for each V_j and k < 0 do // V is the set of vertices of P. //
      // Diagonal return true iff (V_i, V_j) is a proper internal diagonal of P //
      if Diagonal (V_{j−1}, V_{j+1})
      then
        T_i ← Triangle (V_{j−1}, V_j, V_{j+1})
        i ← i + 1; k ← j
      end
    end
    if k < 0 then k = 0
    remove V_k from P
    n ← n − 1
    if n = 3 then T_i ← Triangle (V_0, V_1, V_2)
  end
  if i = 0 then return null
  else return T
```

For example, given the P_1 polygon shown in Figure 4.2. The polygon has 5 corner points. The set of 5 corner points is the input into the triangulation algorithm. The output is a set of 3 triangles where each triangle is represented by its three corners in a single relational database table. This type of representation is called *2-spaghetti* data model. The 2-spaghetti data model is a pervasive model for representing spatial objects in GIS.

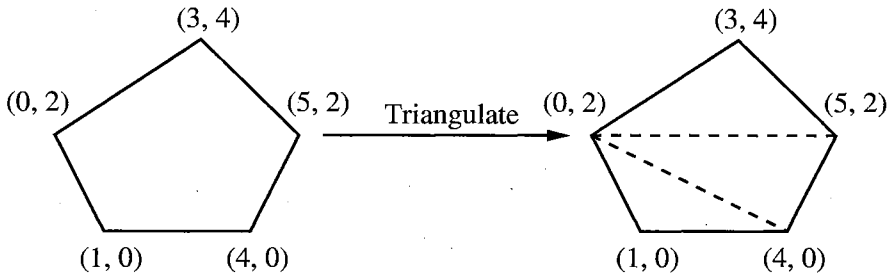


Figure 4.2 Polygon Triangulation

After triangulation, P_1 can be represented by the following table:

ID	x	y	x'	y'	x"	y"
P_1	0	2	3	4	5	2
P_1	0	2	5	2	4	0
P_1	0	2	4	0	1	0

Table 4.7 Triangular Representation

4.2.3 From 2-Spaghetti Data Model to Constraints

After triangulation, a polygon is decomposed into several triangles which can be represented by the 2-spaghetti data model. It is straightforward to translate a triangle from the 2-spaghetti data model into the linear constraint model. In this way the polygon can be represented by the combination of a set of triangles represented in linear constraint model.

The following is a simple algorithm that translates the 3-corner representation of a triangle into the linear constraint representation.

Algorithm: Get linear constraints from a triangle.

Input: Three corners of a triangle: $(x_0, y_0), (x_1, y_1), (x_2, y_2)$.

Output: The linear constraint representation of the triangle.

Algorithm 4.4 Translate a Triangle into Linear Constraints

TriangletoConstraint (T) // T contains 3 corners of the triangle where //
// $T_0 = (x_0, y_0), T_1 = (x_1, y_1), T_2 = (x_2, y_2)$. //
$L \leftarrow Constraint[3]$; $l \leftarrow 0$
for each pair of T_i, T_j of T **do** // T_k is the another corner point. //
// The line passing $(x_i, y_i), (x_j, y_j)$ is $(y - y_i)(x_j - x_i) = (y_j - y_i)(x - x_i)$. //
if $(y_j - y_i)x_k - (x_j - x_i)y_k \leq (y_j - y_i)x_i - (x_j - x_i)y_i$
then $L_l \leftarrow (y_j - y_i)x - (x_j - x_i)y \leq (y_j - y_i)x_i - (x_j - x_i)y_i$
else $L_l \leftarrow (y_j - y_i)x - (x_j - x_i)y \geq (y_j - y_i)x_i - (x_j - x_i)y_i$
$l \leftarrow l + 1$
end
return L

Using this algorithm we get 3 constraints that describe the same triangle.

For example, let us consider a triangle delimited by three corner points: (1,2), (3, 4), (4, 1). The algorithm can produce the constraint representation of the triangle:

$$\left\{ \begin{array}{lcl} x - y & \leq & -1, \\ 3x + y & \leq & 13, \\ x + 3y & \geq & 7. \end{array} \right\}$$

4.2.4 Compute Convex Hull from Constraints

To draw a spatial object from constraints requires two steps: first find the boundary points and then compute the convex hull from those points. Solving constraints gives us the boundary points. The convex hull algorithm suffices the second step [PS85]. The computation of the convex hull has been studied extensively and has various applications. The convex hull algorithm is described in detail in [O'R94, PS85]. The algorithm to compute the boundary points from constraints comprises the constraint solving and the convex hull algorithm. The following conversion algorithm is adapted from the algorithm presented in [Pra98, Wan99]. When we compute the boundary points from constraints we calculate the intersection points of lines which are formed by replacing the inequality operator with an equality sign of constraints. This technique is shown in the algorithm.

Input: Given a set of constraints.

Output: Ordered Boundary points of the convex hull.

Algorithm 4.5 Compute Convex Hull from Constraints

```
getConvexPoints (L) // L is a set of n constraints. //
  N ← {} // N is an empty node list. //
  P ← {} // P is an empty point list. //
  M ← Integer [||L||] // M is an integer array storing the line count in N. //
  for i ← 0 to ||L|| − 1 do // Find the intersection points. //
    for j ← i + 1 to ||L|| − 1 do
      if not Parallel (l_i, l_j) // l_0, l_1, ···, l_{n−1} are lines derived from L. //
      then
        p ← Intersection (l_i, l_j) // p is the intersection point of l_i and l_j. //
        if Satisfiable (p, L)
        then add (N, (p, l_i, l_j)); M_i ← M_i + 1; M_j ← M_j + 1
      end
  for i ← 0 to ||L|| − 1 do // Remove the internal node. //
    if M_i ≤ 1 and ||N|| > 1 and ||L|| > 3
    then
      for j ← 0 to ||N|| − 1 do
        v ← N_j
        if l_i = v_line1 or l_i = v_line2 then remove (N_j)
        // l_i is not a edge of the convex hull. //
      end
    end
  if ||N|| ≤ 2 // The convex hull is a point or a line segment. //
  then P ← N
  else // Sort the boundary points. //
    v ← N_0 // N_0 is the first point list. //
    e ← v_line1
    while v_p ∉ P do
      add (P, v_p); found ← false
      for each r ∈ N and not found do
        if (e = r_line1 or e = r_line2) and v_p ≠ r_p
        then
          v ← r; found ← true
          if e = v_line1 then e ← v_line2 else e ← v_line1
        end
    end
  end
  return P
```

For example, given the input constraints: $\left\{ \begin{array}{lcr} x & \geq & 1, \\ x-2y & \geq & -5, \\ 3x+y & \leq & 20, \\ x-y & \leq & 4, \\ x+3y & \geq & 4. \end{array} \right\}$
The output of this algorithm will be the point list: $(1,1),(1,3),(5,5),(6,2),(4,0)$.

4.2.5 Constraint Solving

Constraint solving has wide applications in mathematics, scientific, engineering, and industrial computations. In general constraint solving serves three purposes: 1) finding out constraints inconsistent with each other, 2) obtaining new constraints, 3) calculating numerical solutions. There are a vast number of different approaches to constraint solving, owing to the importance of the problem. The Simplex algorithm [Dan63, Sch86, NT93] is widely used to solve linear constraints. In our implementation we integrate a Java port of a Simplex solver called *lp_solve* by Michel Berkelaar [Ber97, Gro97]. This non-commercial linear programming code is able to solve problems as large as 30,000 variables and 50,000 constraints. *Lp_solve* can also handle smaller integer and mixed-integer problems.

4.2.6 Area

The computation of the area is essential in the spatial application. The algorithm of calculating the area can be found in any geometry book. To compute the area of a polygon we first compute the area of the triangles which constitute the polygon. Our algorithm is based on that in [O'R94]. The algorithm is listed below:

Input: A polygon.

Output: The area of the polygon.

Algorithm 4.6 Compute the Area of a Polygon

PolygonArea (P) // P is a polygon. //
  $A \leftarrow 0$ **for** $i \leftarrow 1$ **to** $\|P\| - 2$ **do** // $\|P\|$ is number of vertices of the polygon.
  // P_i is a vertex of P where $0 \leq i < \|P\|$. //
  $A \leftarrow A+$ **TriangleArea** (P_0, P_i, P_{i+1})
  return A

TriangleArea (a, b, c) // a, b, c are three corner points of the triangle. //
  return $|((c_x - b_x)(a_y - b_y) - (a_x - b_x)(c_y - b_y))/2|$

4.2.7 Intersection Algorithm

Intersection is a fundamental part of the more complex algebraic operations. We develop two types of intersections: graph intersection and relation intersection. Relation intersection is based on graph intersection. Graph intersection takes a set of graphs and finds the intersection. Relation Intersection takes two relations, r_1 and r_2, as input and performs a nested loop over both collections of tuples, taking the conjunction of each pair of tuples. The algorithm is shown as follows:

Input: Two relations r_1 and r_2.

Output: The intersection relation of r_1 and r_2.

Algorithm 4.7 Relation Intersection

TableIntersection (r_1, r_2) // r_1 and r_2 are two relations. //
  $r \leftarrow \{\ \}$
  for each $t_i \in r_1$ **do**
    for each $t_j \in r_2$ **do**
      $t \leftarrow t_i \cap t_j$
      if $t \neq \phi$ **then add** (r, t)
    end
  return r

4.3 Algebraic Operations

4.3.1 The Join Operation ⋈

The join $r = r_1 \bowtie r_2$ is obtained by computing the conjunction of each tuple of r_1 with each tuple of r_2. The conjunction of two tuples is the concatenation of their constraints. The join can be implemented by a direct call to the *TableIntersection* function. Consider two objects, Graph A and Graph B. Figure 4.3 shows the join operation.

Graph ID	x	y	θ	RHS
A	1	0	$\geq$	2
A	1	0	$\leq$	5

Table 4.8 Graph A

Graph ID	x	y	θ	RHS
B	0	1	$\geq$	1
B	0	1	$\leq$	3

Table 4.9 Graph B

Graph ID	x	y	θ	RHS
C	1	0	$\geq$	2
C	1	0	$\leq$	5
C	0	1	$\geq$	1
C	0	1	$\leq$	3

Table 4.10 Graph C = A ⋈ B

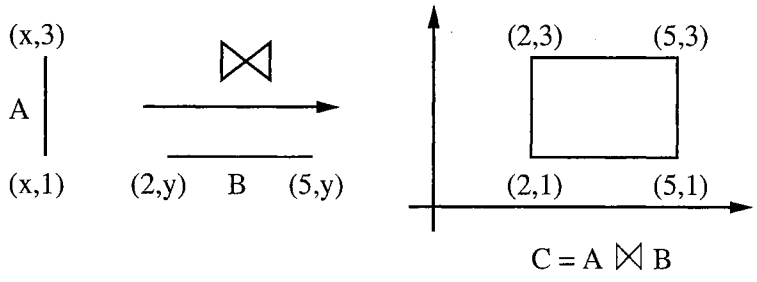


Figure 4.3 Join Operation

4.3.2 The Intersection Operation ∩

$r = r_1 \cap r_2$ is a special case of join. Here r_1 and r_2 are defined on the same attributes. This operator is implemented by a direct call to the *TableIntersection* function. Consider the following two objects, *Graph D* and *Graph E*. Figure 4.4 depicts the intersection and union operation.

Graph ID	x	y	θ	RHS
D	1	0	$\geq$	1
D	1	0	$\leq$	4
D	0	1	$\geq$	1
D	0	1	$\leq$	3

Table 4.11 Graph D

Graph ID	x	y	θ	RHS
E	4	-3	$\geq$	5
E	5	1	$\leq$	30
E	1	4	$\geq$	6

Table 4.12 Graph E

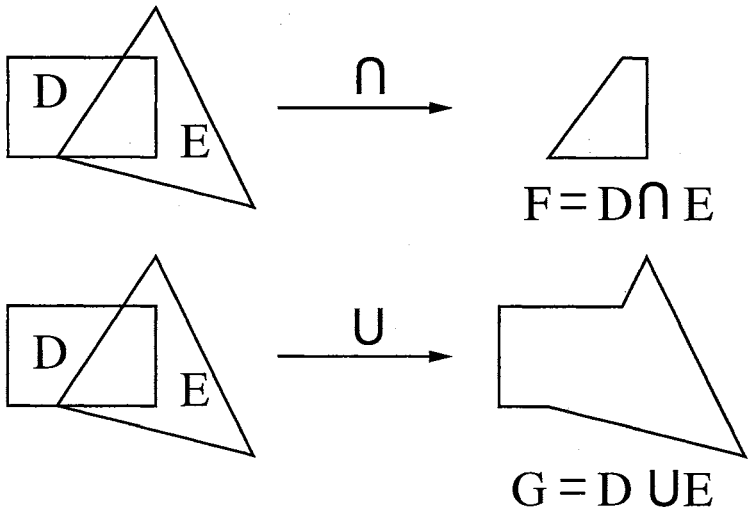


Figure 4.4 Intersection and Union Operation

The result of D ∩ E is F as shown in Figure 4.4.

Graph ID	x	y	θ	RHS
F	1	0	$\geq$	1
F	1	0	$\leq$	4
F	0	1	$\geq$	1
F	0	1	$\leq$	3
F	4	-3	$\geq$	5
F	5	1	$\leq$	30
F	1	4	$\geq$	6

Table 4.13 Graph F = D ∩ E

4.3.3 The Selection Operation σ

The implementation of $r = \sigma_p(r_1)$ is straightforward. p is a set of constraints has to be added to r_1. The operator is implemented by *TableIntersection*. The result is the intersection of r_1 and p.

4.3.4 The Union Operation ∪

$r = r_1 \cup r_2$ is defined as the set of tuples which belong either to r_1 and r_2. In this case we need to add two new tuples into the *graph-specification* table which point to the same tuple. The result of D ∪ E is G as shown in Figure 4.4.

Graph ID	Tuple ID	Fill	Color
D	0	Y	-8355712
E	0	Y	-16732162

Table 4.14 Graph G = D ∪ E

4.3.5 The Difference Operation −

$r = r_1 - r_2$ gives the set of spatial objects that are in r_1 but not in r_2. To do the difference D − E, first negate E and then intersect it with D. The result is shown in Figure 4.5.

Graph ID	x	y	θ	RHS
E	4	-3	$\leq$	5
E	5	1	$\geq$	30
E	1	4	$\leq$	6

Table 4.15 Graph −E

Graph ID	x	y	θ	RHS
H	1	0	$\geq$	1
H	1	0	$\leq$	4
H	0	1	$\geq$	1
H	0	1	$\leq$	3
H	4	-3	$\leq$	5
H	5	1	$\geq$	30
H	1	4	$\leq$	6

Table 4.16 Graph H = D – E

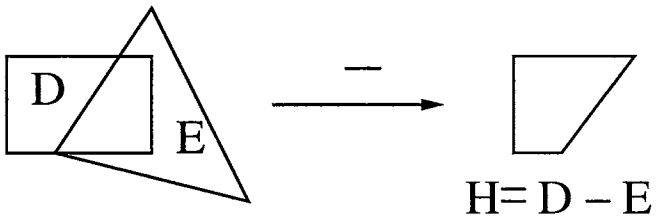


Figure 4.5 Difference Operation

4.3.6 The Projection Operation π

$r = \pi_{\bar{x}}(r_1)$ is defined as the relation whose tuples are the projection of each tuple of r_1 on the $\bar{x}$ axis. The Fourier elimination suffices in the implementation. Figure 4.6 shows the projection of *Graph I* on $\bar{x}$-axis and $\bar{y}$-axis.

Graph ID	x	y	θ	RHS
I	1	-2	$\geq$	-5
I	1	2	$\leq$	15
I	3	-2	$\geq$	13
I	1	0	$\geq$	0
I	1	1	$\geq$	4

Table 4.17 Graph I

Graph ID	x	y	θ	RHS
J	0	1	=	0
J	1	0	$\geq$	1
J	1	0	$\leq$	7

Table 4.18 J = $\pi_{\bar{x}}$(I)

Graph ID	x	y	θ	RHS
K	1	0	=	0
K	0	1	$\geq$	1
K	0	1	$\leq$	5

Table 4.19 K = $\pi_{\bar{y}}$(I)

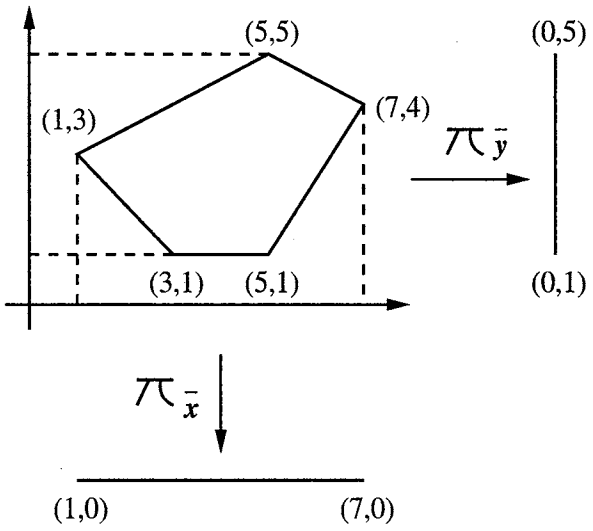


Figure 4.6 Projection Operation

4.4 Development Document

4.4.1 Environment and Software

The environment and software requirements are similar to the Web-based application as listed below:

- a Java programming environment - *SUN Java 2 SDK.*
- a CORBA compliant Object Request Broker - *Object Oriented Concept Orbacus.*
- a database system - *Hughes Technologies mSQL.*
- a linear programming solver - *Mixed Integer Linear Program Solver* by Michel Berkelaar [Ber97].

Java is used to build up the system. To access databases in a distributed computing environment CORBA classes are required. The mSQL database management system stores databases created in the system. One extra component is the linear programming solver which supports constraint queries.

4.4.2 Architecture

Figure 4.7 illustrates the architecture of the prototype. It consists of the following components:

1. A **Graphical User Interface (GUI)** which facilitates the easy access to the database.

2. A **query processor** to distinguish between standard SQL commands and constraint operations and to deliver the standard commands to the DBMS and constraint commands to the constraint function library and the constraint solver.

3. A **linear constraint solver** which resolves the linear constraints.

4. A **constraint algebraic function library** to respond constraint algebraic operations.

5. A **data portal** which allows the plain text input and output, and various file format conversion.

6. A **database portal** which contains the CORBA code and allows to access databases locally or remotely.

7. The **DBMS** which provides SQL processing and data storage management.

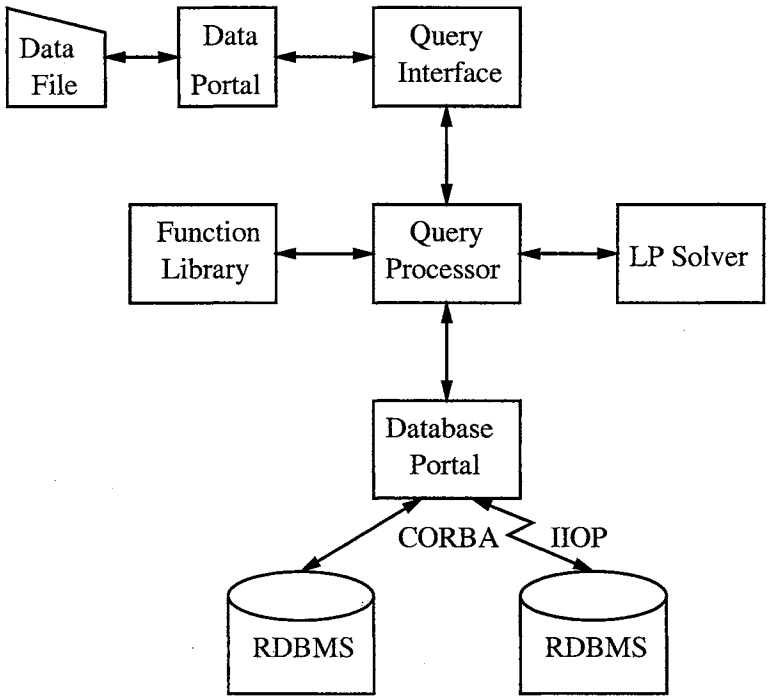


Figure 4.7 System Architecture

Our design of the system tends to flexible and extendible. Components can be replaced or added for later improvement.

4.4.3 Developing a CORBA Application

To develop a CORBA application we follow the steps as shown in Figure 3.2. The IDL file defines the objects of our constraint database system as shown below:

```
module QueryCDB {

  typedef sequence<string> StringSeq;

  // Define a geospatial entry.
  struct GeoTable {
    string description;
    long color;
    long tuplelink;
  };

  // Define a database field.
  struct DBField {
    string name;
```

```
  short type;
};

// Define a tuple.
struct Tuple {
  StringSeq value;
};

// Define a graph specification.
struct GraphSpec {
  long tupleID;
  boolean fill;
  long color;
};

// Define a constraint.
struct Constraint {
  long graphID;
  long x;
  long y;
  string theta;
  long rhs;
};

// Define the sequences.
typedef sequence<GeoTable> GeoTableSeq;
typedef sequence<DBField> DBFieldSeq;
typedef sequence<Tuple> TupleSeq;
typedef sequence<GraphSpec> GraphSpecSeq;
typedef sequence<Constraint> ConstraintSeq;

// Declare the database interface.
interface ExecDatabase {
  // Select database.
  void setDatabase(in string databasename);
  // Clear database.
  void initDatabase(in DBFieldSeq dbfields);
  // Get tables.
  GeoTableSeq getGeoTable();
  // Store tables.
  void storeGeoTable(in long tableID, in GeoTable spatialTable);
  // Get database fields.
  DBFieldSeq getDBField();
  // Store database fields.
```

```
    void storeDBField(in long fieldID, in DBField cdbField);
    // Get the tuple.
    TupleSeq getTuple(in string query);
    // store the tuple.
    void storeTuple(in long tupleID, in Tuple cdbtuple);
    // Get the graph specification.
    GraphSpecSeq getGraphSpec();
    // Store the graph specification.
    void storeGraphSpec(in long graphID, in GraphSpec graphs);
    // Get constraints.
    ConstraintSeq getConstraint(in long graphID);
    // Store constraints.
    void storeConstraint(in ConstraintSeq constraints);
  };
};
```

`QueryCDB` is a module defining objects and methods required for database access. Five data objects are specified in the module corresponding five data structure classes in the system as shown in Table 4.6. `ExecDatabase` interface contains 12 functions. `setDatabase` sets the database name. `initDatabase` initializes the database. `getGeoTable` and `storeGeoTable` get and store tables. `getDBField` and `storeDBField` get and store database fields. `getTuple` and `storeTuple` get and store data tuples. `getGraphSpec` and `storeGraphSpec` get and store graph specifications. `getConstraint` and `storeConstraint` get and store constraints.

4.4.4 Classes

- Main classes:
  - **QueryCDBApp** The main program to initiate the application.
  - **QueryFrame** The main window frame to hold the toolbox, the feature window, and the graph window.
  - **QueryServer** The server program to initiate the ORB objects which wait for the call from clients.
  - **ToolPanel** The toolbox listing the icons which indicate system functions.

  - **TableList** A graphical view for the relations (feature window).

- Drawing classes:

  - **ColorPad** An interface to adjust the color.
  - **GraphPad** A drawing pad for points, lines, rectangles, and polygons (graph window). It contains the control for mouse and keyboard inputs.
  - **Swatch** A color swatch showing the customized color.

- File classes:

  - **DataPortal** A class to read or save the database in the plain text format.
  - **FileTool** A dialog for the file access.

- Query classes:

  - **AreaDialog** A dialog for the area query.
  - **MinMaxDialog** A dialog for the minimum and maximum query which calls linear programming library classes through the LPSolver class.
  - **QueryDialog** A dialog which sends SQL commands to the database management system.
  - **RecordDialog** A dialog for the tuple update.
  - **TableDialog** A dialog for the relation definition.
  - **TableListDialog** A dialog for the relation update.

- Data structure classes:

  - **DBField** A class defining database attributes.
  - **DBList** A class defining a database which includes the database name, relations, attributes, and tuples.
  - **GeoObject** A base class for drawing objects.

  - **GeoTable** A base class for relations.
  - **Constraint** A class defining constraints.
  - **Triangle** A class defining a triangle by three points.

- Database classes:
  - **AccessDatabase** A class invoked by **TeleDatabase** to access the local and remote database.
  - **ExecDatabaseImpl** The implementation class for the CORBA POA class generated by IDL.
  - **TeleDatabase** A dialog for importing and exporting the database. It accesses the remote database by the ORB call.
  - **QueryMsql** A class accessing the mSQL database system.

- Computing geometry classes:
  - **CompConstraint** A class including subroutines for constraint computation.
  - **CompGeom** A class including subroutines for geometry computation.
  - **GeoConstraint** A class including subroutines for geometry operations in the vector format.

- Supporting classes:
  - **CloseWindow** A class which activates the event for closing of frames or dialogs.
  - **ImageButton** A basic structure defining the icon button with its event controls.
  - **LPSolver** An interface for **MinMaxDialog** to access linear programming library classes.
  - **MessageBox** A dialog box for some helpful messages.

CHAPTER V

QUERY APPLICATIONS

5.1 Graphical User Interface

A graphical user interface (GUI) facilitates friendly and efficient access to the system. It is the portal for data input/output, database query, and database import/export. A simple and graphic representation is essential for any geospatial application. The constraint representation is hard for the end user of the system to comprehend. The graphical interface provides a mechanism to convert the constraint representation into a graph type representation that is evident to the end user of the system. This conversion involves locating the boundary points of the geospatial objects that are represented by constraints. Once the boundary coordinates are found, the data can be visualized. In this section we describe the graphical user interface and also introduce the different features that we provide in the implementation of this general query system.

5.1.1 Data Input and Output

The data is stored in a constraint database format [KKR90, KKR95] for efficient data manipulation using linear constraint solving. But the constraint format is usually not so comprehensible as the vector format. So the user should have some simple method to input the data without worrying about the constraint data storage. Some of these options are suggested as follows [KRW98]:

Drawing : This is the primary way for the user to enter the data. The end user draws geospatial objects in the form of points, lines, rectangles, or polygons and stores them into the database. The user can use the icons in the drawing

toolbox to specify the filled type, the color, and the geometric shape of the objects.

Plain Text : A plain text format is designed for the constraint database (See Appendix D) which can be read by the system. This can be an electronic exchange format for the constraint database.

Constraint Database : The user is able to input the data into the database management system directly.

Translation : The translation allows the user to use the data in other data formats, such as a TIGER file, or an ARC/INFO GIS database.

Image Scanning : The user can convert the image directly into the constraint database.

Purchasing : The data can be acquired from data providers. For example, the data provider could provide data about land use in the required format which an be read by the system.

Drawing, plain text, and database input are implemented in this system. The rest of them is for the further development. In the previous chapter we show how to store the constraints into the relational database. Here, we give an example.

For example, Tweety Bird has land at 100 Perkins Road in Stillwater for sale. The price is $1000/Acre. The coordinates are (10, 10), (10, 20), (20, 20), and (30, 10). We can store the information in the relational database as shown in Table 5.1 and Table 5.2.

Owner	Address	City	Price
Tweety Bird	100 Perkins Road	Stillwater	$1000/Acre

Table 5.1 Alphanumeric Data

x	y	θ	RHS
1	0	$\geq$	10
0	1	$\geq$	10
0	1	$\leq$	20
1	1	$\leq$	40

Table 5.2 Spatial Data

5.1.2 Query Operations

Querying the database is made easy through various icons and dialog boxes in the GUI. Most queries provide a dialog box for data input or update. The GUI contains the following icons for querying purposes as shown in Figure 5.1:

Tables A dialog box for creating, deleting, and updating a relation. The **Design** button prompts a dialog box for defining the tuple attributes in the table.

Query A dialog box for issuing SQL-like commands to query the database. Both novel and experience users benefit from this interface. A novel user can use the button to select the condition for the query. An experienced can just issue the SQL command to query the database.

Import/Export A dialog box for accessing local or remote databases. The user can specify more than one database at a time. This allows the user to access and store data distributedly.

Area A dialog box for the area calculation.

Min/Max A dialog box for specifying the objective function to be maximized and minimized on those constraints selected by the user.

Intersection A dialog box for displaying the new tuple which is the result of the intersection for those tuples selected by the user.

Union A dialog box for displaying the new tuple which is the result of the union for those tuples selected by the user.

5.1.3 User Guide

1. To start the system, run **java QueryCDBApp**. You can see the GUI is composed of three parts as shown in Figure 5.1:

   (a) **Toolbox** All available functions are shown in this part. All functions can be accessed through pull-down menus. Important functions are also represented in icons.

   (b) **Feature window** This is the window which lists all relations (tables) created. It is called the feature window because each relation contains one spatial feature in most cases.

   (c) **Graph window** The constraints are drawn in this window.

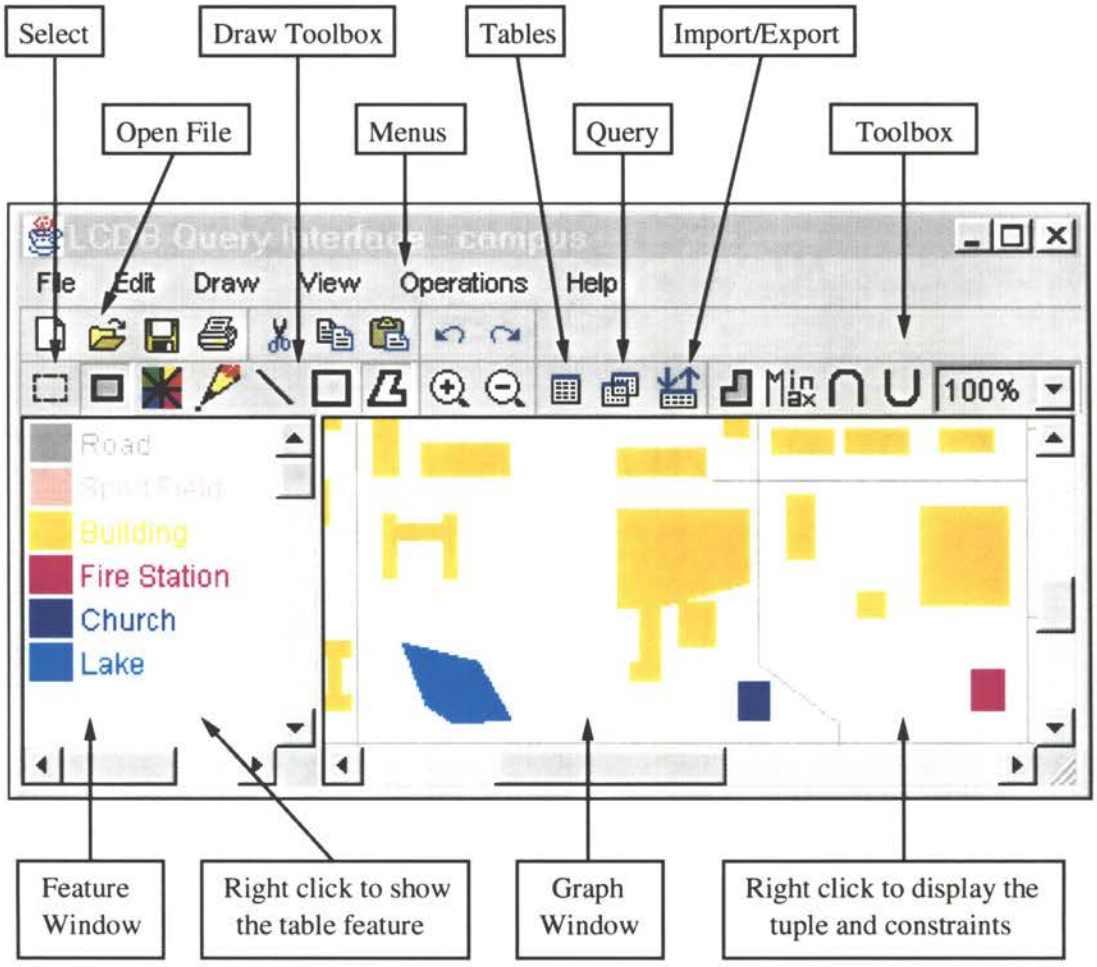


Figure 5.1 Graph Interface of the System

2. Now you can open a text file that contains the linear constraints description. Use *Open* on the *File* pull-down menu to choose a file. Or click the *Open File* icon to choose a file.

3. After a file has been read in, the relation names and their description will be displayed in the feature window. The graphs are drawn in the graph window.

4. Use the *zoom out* icon to make the graphs smaller. Use the *zoom in* to enlarge the graphs.

5. To input a polygon, first click the *polygon* icon to enable polygon drawing. To draw a polygon in the graph window, move the mouse to the start point. Left click the mouse to set the first point. Release the mouse, move to the next point, and left click the mouse to set this point. Follow this way to set the consecutive points. Left double click the mouse to set the final point of the polygon. Then the polygon is drawn. To input a point, a line, and a rectangle, follow the similar steps.

6. To update a relation (table) entry, click the *tables* icon or right click on the feature window. A dialog box will prompt the user to create, delete, and update a relation entry as shown in Figure 5.2.

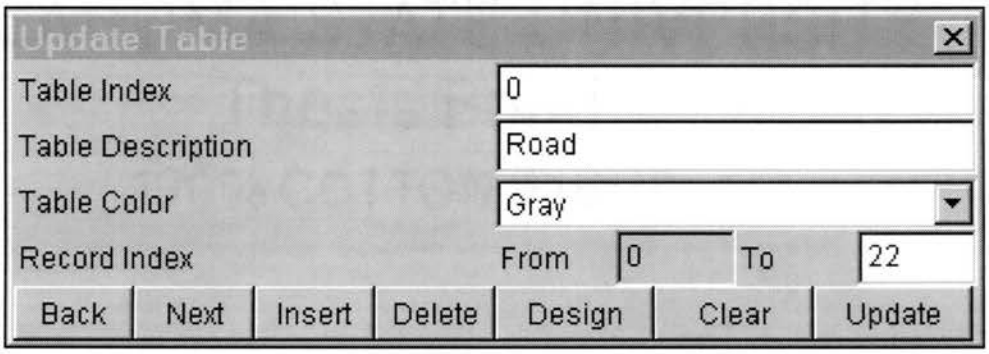


Figure 5.2 Update a Relation Entry

7. To define a relation (table), click the *design* button in the table dialog box. A dialog box will prompt the user to specify attributes of a tuple that are composed of the field name and data type as shown in Figure 5.3.

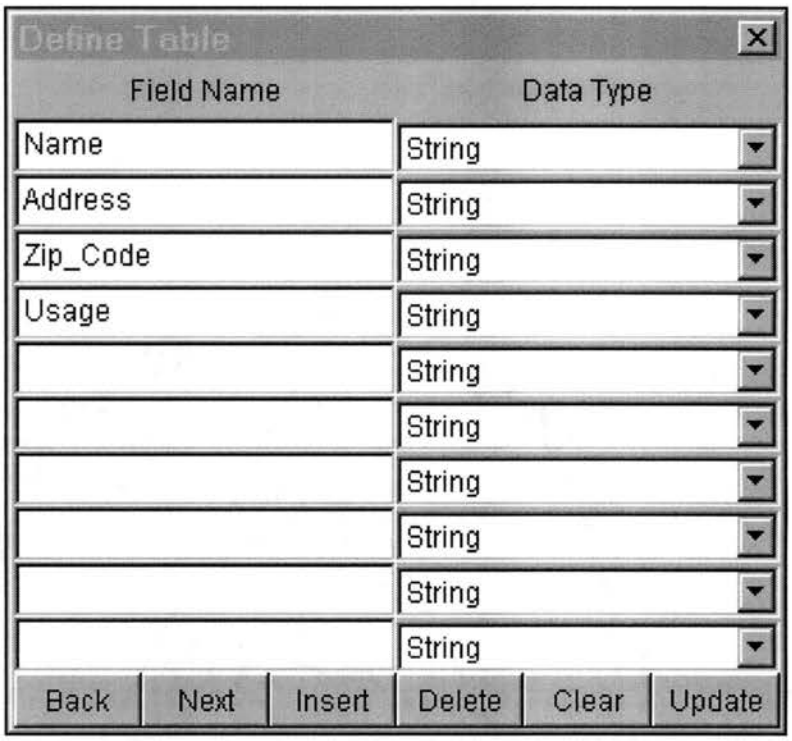


Figure 5.3 Define a Relation

8. To update a tuple (record), right click on the graph window. A dialog box will prompt the user to create, delete, and update the tuple as shown in Figure 5.4. This dialog box provides the user to access both the alphanumeric and constraint data in the tuple.

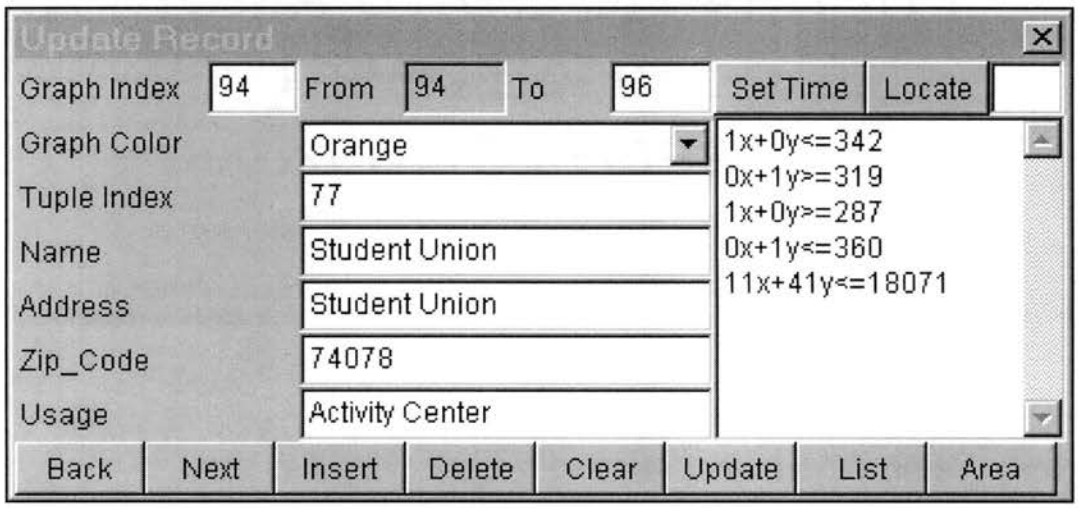


Figure 5.4 Update a Tuple

9. The graph index in the tuple dialog box contains three parts to indicate the graph currently pointed by the right mouse click, the first graph, and last graph in this tuple. The constraints of the graph are listed in the right side of the tuple dialog box. To view the constraints of another graph in this tuple, change the graph index value. To set the time stamp of the graph, assign the value in the time stamp field and click the *set time* button to mark it. The *locate* button allows the user to project the graph in a certain time.

10. To use the SQL-like command to query the database, click the *query* icon. A query dialog box as shown in Figure 5.5 appears for the user to input the query command. To do a window query, select a window and then query the database.

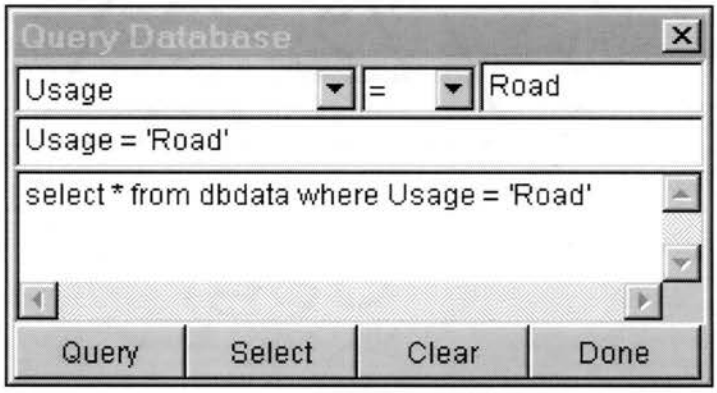


Figure 5.5 Query Database

11. To query the database by constraints, input constraints of concern in the command field of the query dialog box and click the *select* button as shown in Figure 5.6. The *select* button stands out for the constraint query purpose. It facilitates access to the constraint property of the database. The result is highlighted in the graph window and a result tuple will appear for the further specification. To query the location of a certain point, input the coordinate of the point in the command field and click the *select* button. The point will be drawn in the graph window.

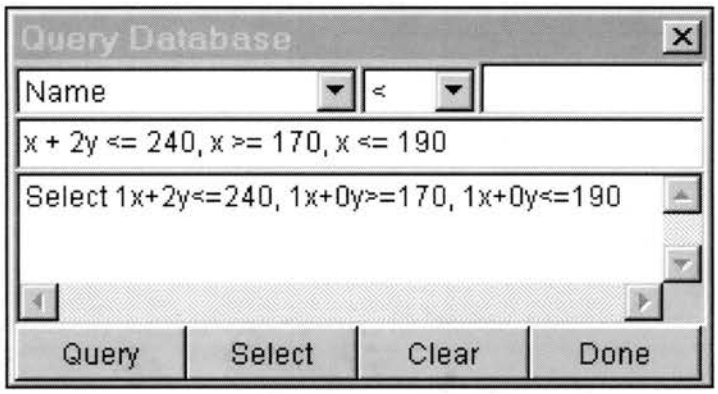


Figure 5.6 Query by Constraints

12. To import or export the database, click the *import/export* button in the database dialog box. Specify the host and the database name to import and export as shown in Figure 5.7. The user can import databases from more than one host. This facilitates access to distributed data.

Figure 5.7 Import/Export Database

13. To query the area of a table, left click the mouse to set the selected relation in the feature window and then click the *area* icon. A dialog box will prompt the user to input the value for addition, subtraction, multiplication, and division as shown in Figure 5.8. After input the value and select the operator, click the *calculate* button to compute the result. This function could be useful when the area needs to be calculated with other factor. To query the area of a tuple, click the *area* button on the tuple dialog box. To query the area of a graph, select the graph and click the *area* icon.

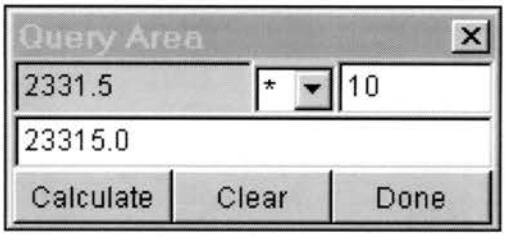


Figure 5.8 Query Area

14. To calculate the objective function, select the graph in the graph window and click the *min/max* icon. A dialog box will prompt the user to specify the objective function as shown in Figure 5.9. Click the *query* button to get the result.

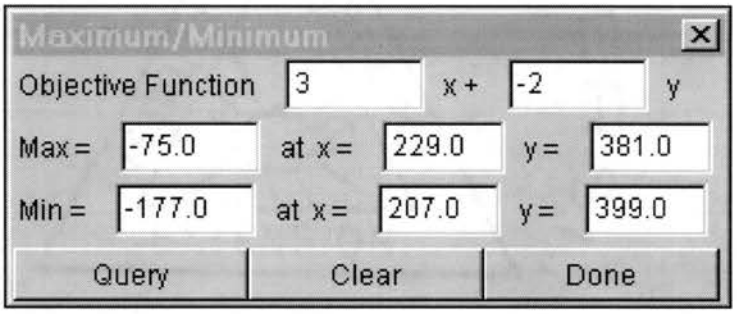


Figure 5.9 Minimum/Maximum Query

15. To take the intersection or union of the relations, select relations on the feature window and click the *intersection* or *union* icon. A tuple dialog box with the result will appear. To take the intersection or union of the graphs, select the graphs. Follow the same steps for the relations.

5.2 A Land-use Example

This example is adopted from the one in [AS91]. The authors extended a DBMS with spatial operations in that paper. Most spatial operations proposed in that paper and in Section 2.1.2 can be performed in our system in a simple way. For example, to use the *object_at* query, we need only to right click on the object in our system. Here, we create an OSU campus map in which each tuple consists of 4 attributes: *Name*, *Address*, *Zip Code*, and *Usage* as shown in Figure 5.10.

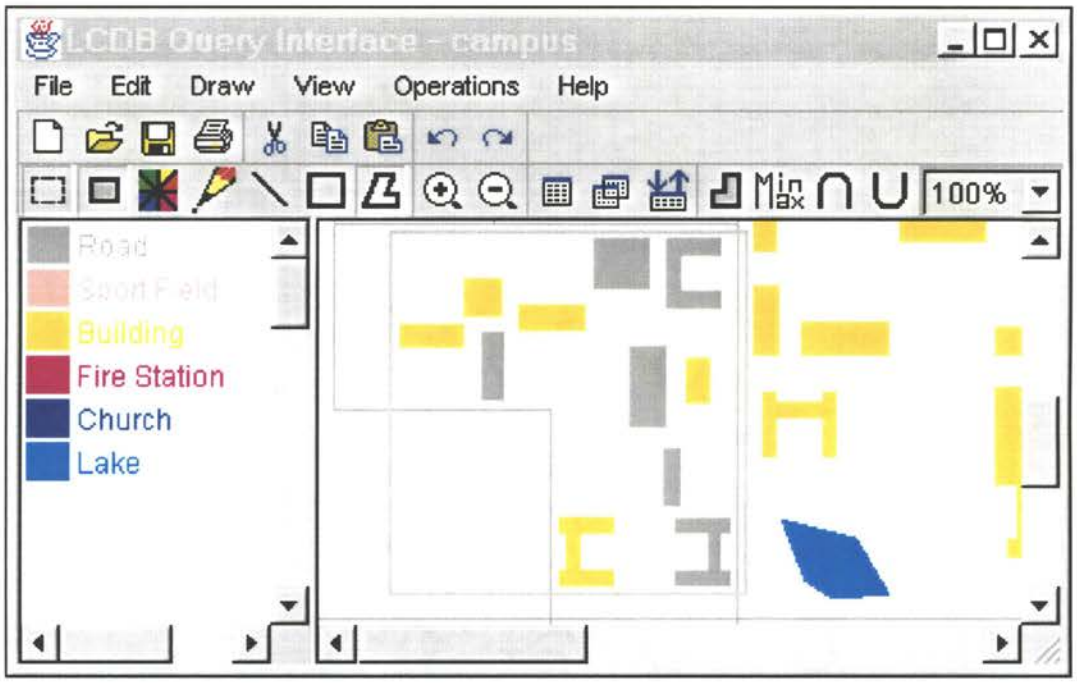


Figure 5.10 Land-use Database

As mentioned above we can know the information of each spatial object by right clicking on each spatial object. If we want to look for a specific target, just give the specification in the query dialog box. We can also query a region delimited by a select box. To do this window query, select the region first and then give the query command in the query dialog box as shown in Figure 5.11. For example, we want to know which building is used for the office and classroom. The result is highlighted as shown in Figure 5.10. The result table is shown in Figure 5.12.

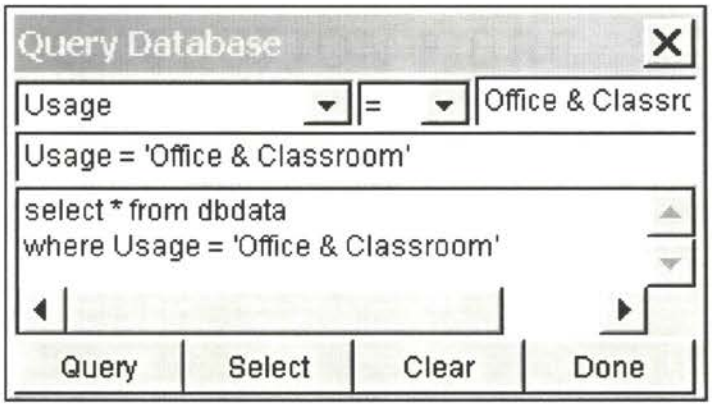


Figure 5.11 Land-use Query

Record List

Name	Address	Zip_Code	Usage
Scott Hall	Scott Hall	74077	Office & Classroom
Hes West	Hes West	74078	Office & Classroom
Human Env Sci	Human Env Sci	74078	Office & Classroom
Tele-Comm Center	Tele-Comm Center	74078	Office & Classroom
North Murray	North Murray	74078	Office & Classroom
South Murray	South Murray	74078	Office & Classroom

Figure 5.12 The Result of Land-use Query

5.3 Insecticide Example

A farmer has his field sprayed with three types of insecticides, namely Lorsban, Parathion, and Carbaryl. The region sprayed by each insecticide is represented by a relation in the database. The database is depicted in Figure 5.13. The cost schedule is shown in Table 5.3.

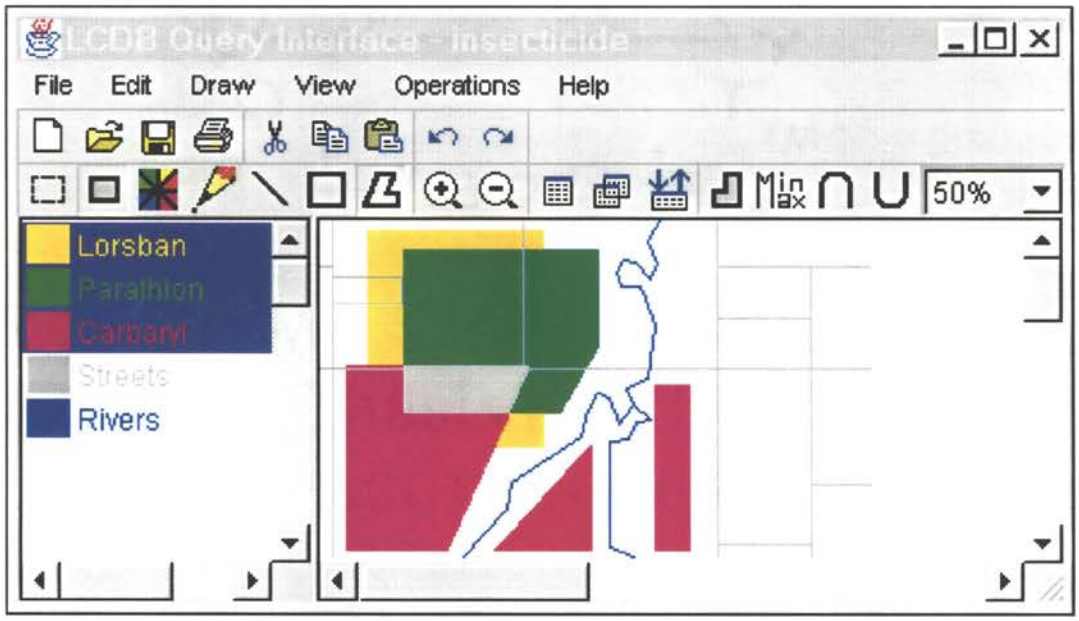


Figure 5.13 Insecticide Database

Insecticide	Cost/Acre	Application Cost/Acre
Lorsban	$2.50	$12.0
Parathion	$4.35	$12.0
Carbaryl	$3.25	$12.0

Table 5.3 The Cost Schedule of the Insecticides

The farmer can do several interesting queries:

1. The cost of each insecticide.

2. The region sprayed by any insecticide.

3. The region both two insecticides or all three were sprayed.

To answer the first query, just select each relation and click the *area* button. Then apply the cost schedule to get the cost. To answer the second query, we have to make a union of the three regions, i.e., highlight two or three relations on the feature window and click the *union* button. The union is computed and stored in a new tuple. To answer the intersection, click the *intersection* button. The intersection is drawn as a gray region in Figure 5.13 and the result tuple is shown in Figure 5.14.

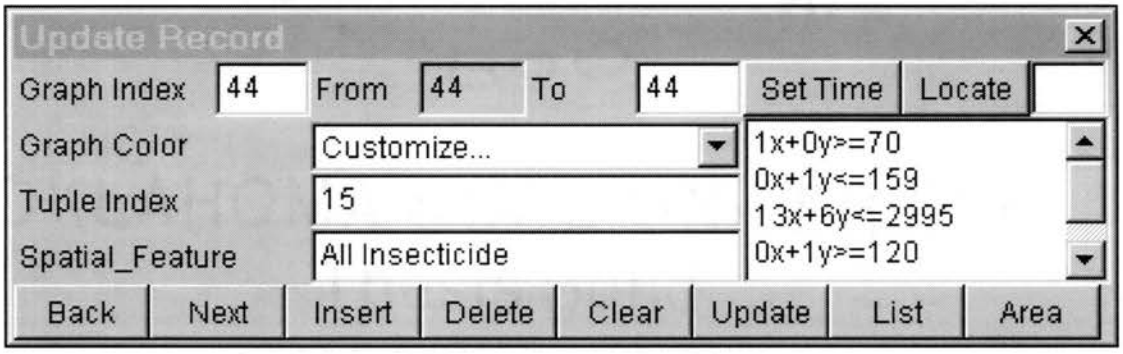


Figure 5.14 The Result of Intersection

5.4 Manufacture Example

This example is adopted from the one in [BJM93]. Consider a company manufactures two products from three resources. Two relations are defined in this database. The *order* relation contains the attributes: *Order No*, *Customer*, *Production ID*, and *Product Quantity*. An example is shown in Table 5.4.

Order No	Customer	Production ID	Product Quantity
1	Jane Sebring	1	100
2	Al Goodwin	2	120
3	Dan Talley	1	120
4	Mike Johnson	2	150

Table 5.4 Order Table

The *process* relation specifies a relationship between production and resources. P_1 and P_2 represent quantities of the first and second products respectively. R_1, R_2, and R_3 represent amounts of the first, second, and third resources available. The possible manufacturing process can be specified by a conjunction of the following constraints:

$$\begin{aligned} &2P_1 + 3P_2 \leq R_1, \\ &2P_1 + P_2 \leq R_2, \\ &P_1 + 4P_2 \leq R_3, \\ &P_1, P_2, R_1, R_2, R_3 \geq 0. \end{aligned}$$

This defines how the production of P_1 and P_2 depends on the amount of available resources of R_1, R_2, and R_3. The constraints for another manufacturing process are:

$$\begin{aligned} &P_1 + 5P_2 \leq R_1, \\ &-P_1 + 4P_2 \leq R_2, \\ &5P_1 \leq R_3, \\ &P_1, P_2, R_1, R_2, R_3 \geq 0. \end{aligned}$$

Suppose there are 500, 300, and 600 units of the first, second, and third resources. The profit for one unit of the first product and the second is \$20 and \$15 respectively. From the above information the *process* table can be created as shown in Table 5.5.

Process ID	P_1	P_2	θ	R
1	2	3	$\leq$	500
1	2	1	$\leq$	300
1	1	4	$\leq$	600
1	1	0	$\geq$	0
1	0	1	$\geq$	0

Table 5.5 Manufacturing Process Table

We are interested in the queries:

1. What is the maximum profit the company can make with each manufacturing pattern and how many units for each production?

2. Which manufacturing process can produce more first product regardless of the second product?

The database contains two feature tables: Process 1 and Process 2, as shown in Figure 5.15. To answer the first query, just select each relation and click the *min/max* icon. In the maximum/minimum dialog box, assign 20 to the x coefficient and 15 to the y coefficient respectively. Then click the *query* button to get the maximum/minimum value. We get the maximum profit for the first manufacturing process is $3500 at 100 units of the first product and 100 units of the second. The second manufacturing process produces the $3540 profit at 120 units of the first product and 76 units of the second as shown in Figure 5.16.

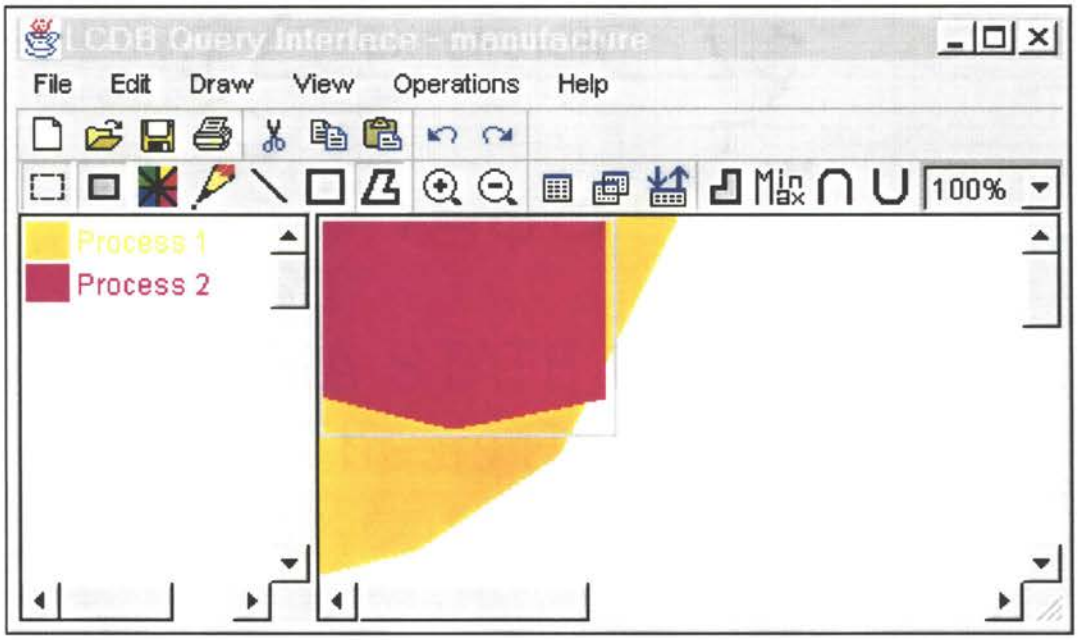


Figure 5.15 Manufacturing Pattern Database

For the second query, select each relation and in the maximum/minimum dialog box assign 20 to the x coefficient and 0 to the y coefficient respectively. There is $3000 profit at 150 units of the first product in the first manufacturing process and $2400

profit at 120 units of the first product in the second manufacturing process. So, the first manufacturing process produce more.

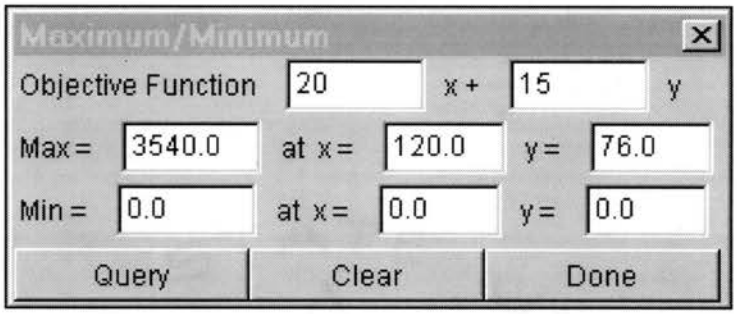


Figure 5.16 Query Maximum Production

5.5 Freightage Example

This problem is adopted from the postage example presented by Revesz and Li in [RL97]. Here we create a different scenario and use a freight company as an example. In our example, the freightage fee charged for packages is computed based on the weight of the package and the freightage rate. The freightage rate increases piecewise linearly with the weight of the package as shown in the Table 5.6. The packages that we want to dispatch are shown in the Table 5.7. The freightage database can be presented as shown in Figure 5.17. We want to query the freightage fee for each package.

Weight in pound (w)	Freightage in Dollars (f)
0-40	$f = 0.5w$
40-80	$f = 20 + 0.4(w - 40)$
80-120	$f = 36 + 0.3(w - 80)$
120-160	$f = 48 + 0.25(w - 120)$

Table 5.6 Freightage

Package ID	Weight in pound (w)
501	60
502	90
503	140

Table 5.7 Packages

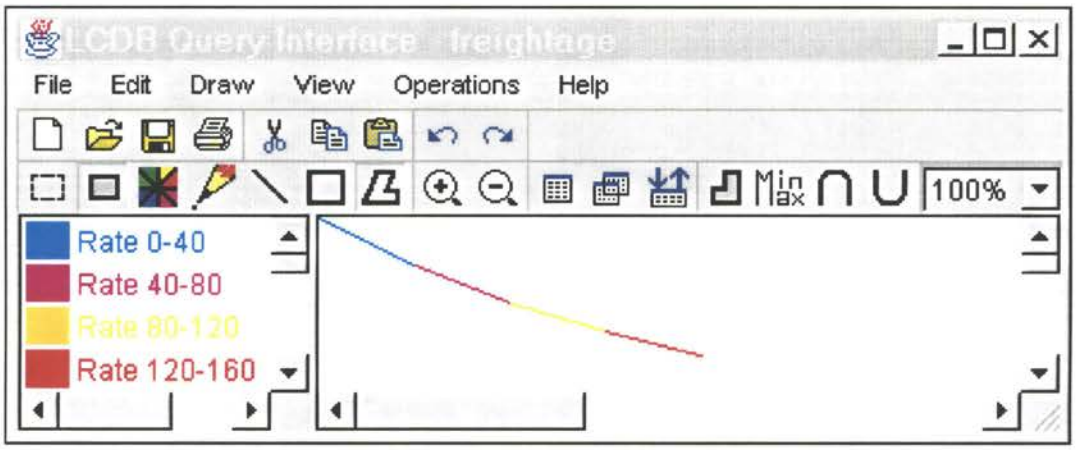


Figure 5.17 Freightage database

To query the freightage, input the weight in the command field in the query dialog box and click the *select* button as shown in Figure 5.18. A result tuple dialog box will be displayed as shown in Figure 5.18. In our case, we query the freightage fee of the 140 pound package and the result is $53.

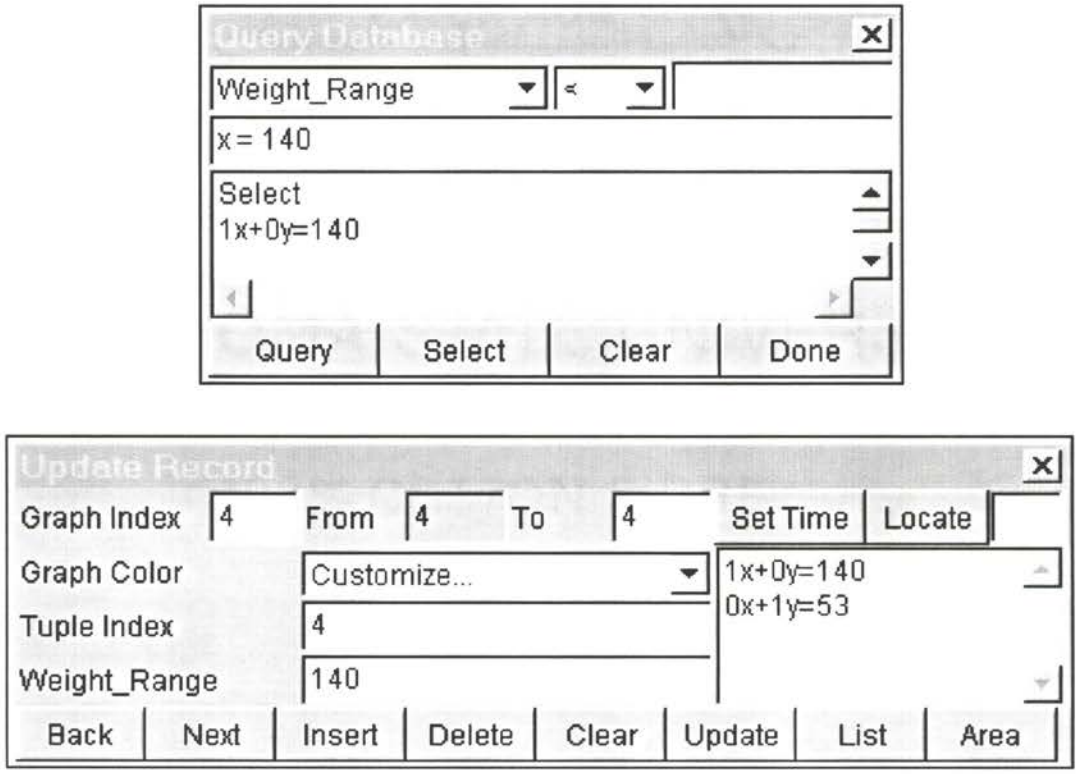


Figure 5.18 Query Freightage

5.6 Tracking Problem

Pete is doing a field trip. He gets his latitude and longitude from a Global Position System (GPS). He wants to know his current location and the target region delimited by $x + 2y \leq 240, x \geq 170$, and $x \leq 190$. A map database is drawn in Figure 5.19. To find his current position, he needs to input his coordinate in the command field of the query dialog box as shown in Figure 5.20. The result tuple is shown in Figure 5.21 and indicated by a black point in the gray trapezoid as shown in Figure 5.19.

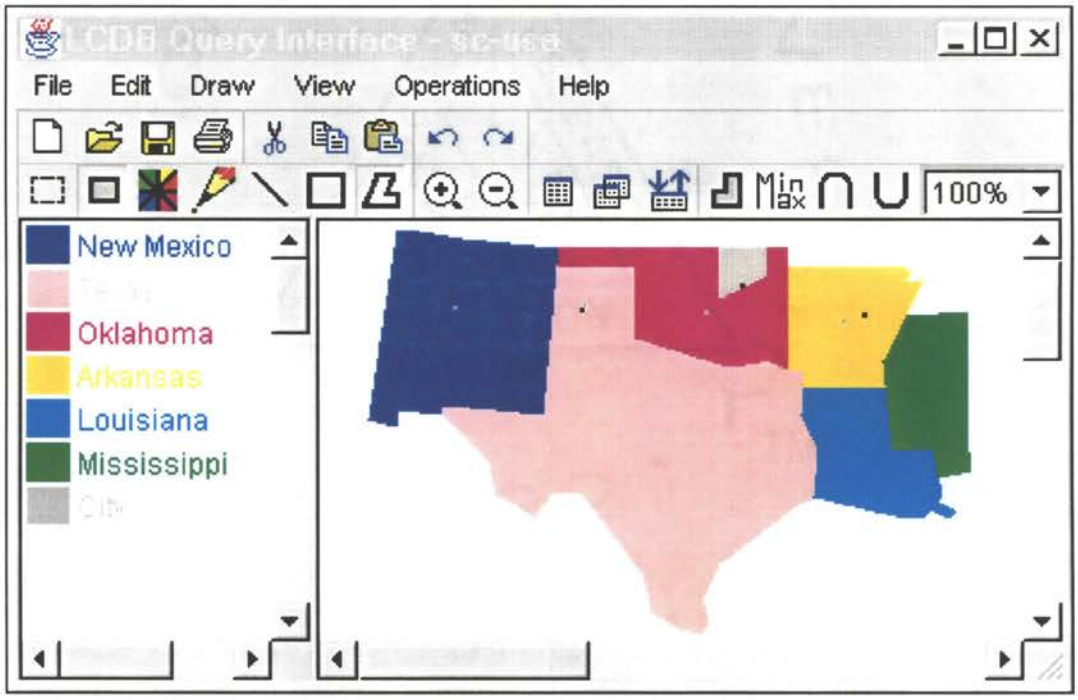


Figure 5.19 Map database

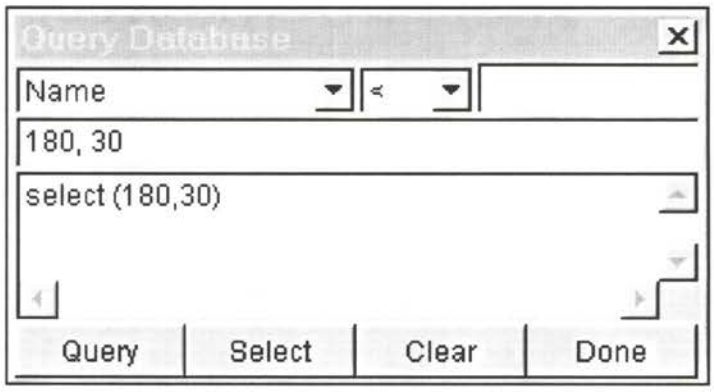


Figure 5.20 Locate Point

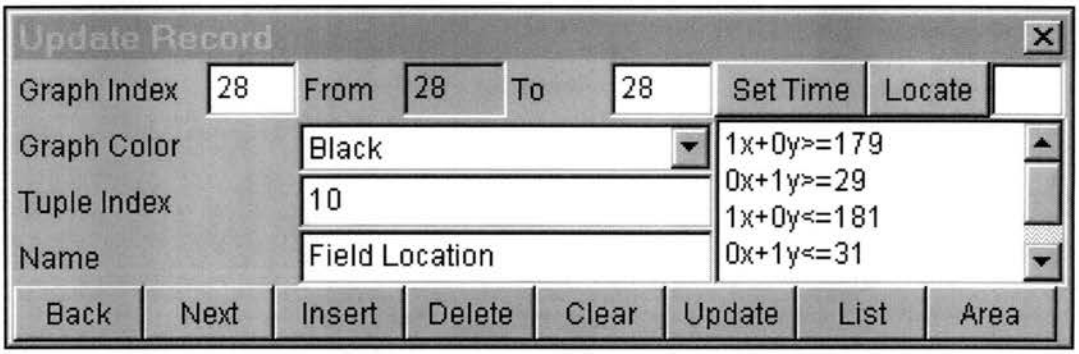


Figure 5.21 Field Location

To find the target region, he needs to input constraints into the command field of the query dialog box and click the *select* button as shown in Figure 5.6. The result tuple is shown in Figure 5.22 and drawn as a gray trapezoidal region in Oklahoma as shown in Figure 5.19.

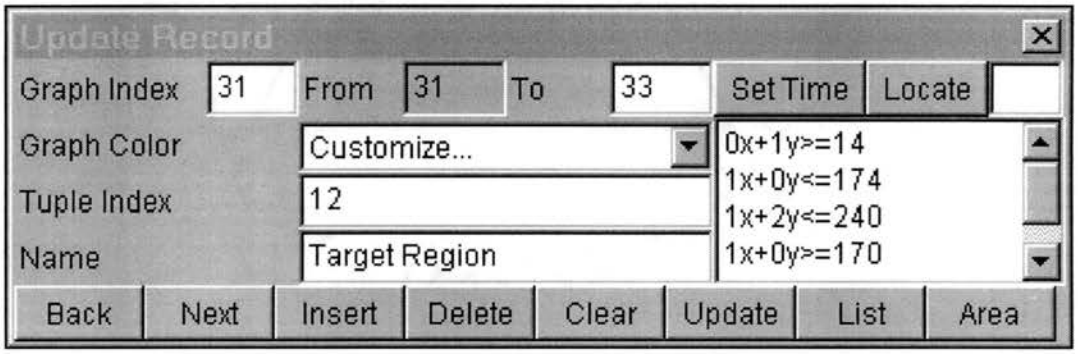


Figure 5.22 Target Region

5.7 Position Prediction

Jane plans to drive from Albuquerque, New Mexico to Oklahoma City, Oklahoma in 8 hours. She starts to drive at 8 AM. She would like to know:

1. Where should she be at 12 PM?

2. In this speed, can she arrive in Little Rock, Arkansas before 9 PM?

A south-central US map database is created as shown in Figure 5.19. Four cities including Albuquerque, Amarillo, Oklahoma City, and Little Rock are marked as gray points in the graph.

To answer these two queries, she selects the tuple of Albuquerque and Oklahoma City, inputs the time in the time field, and clicks the *set time* button to set the time stamp as shown in Figure 5.23. Then she gives the time of concern - 12 and click the *locate* button. The result tuple is depicted in Figure 5.24 and drawn as a black point nearby Amarillo in Figure 5.19. From the graph she knows she should be nearby Amarillo at 12 PM. To answer the second query, she sets different time - 21. The result is drawn as a black point in the east side of Little Rock as shown in Figure 5.19. This indicates that she can arrive in Little Rock by 9 PM in this speed.

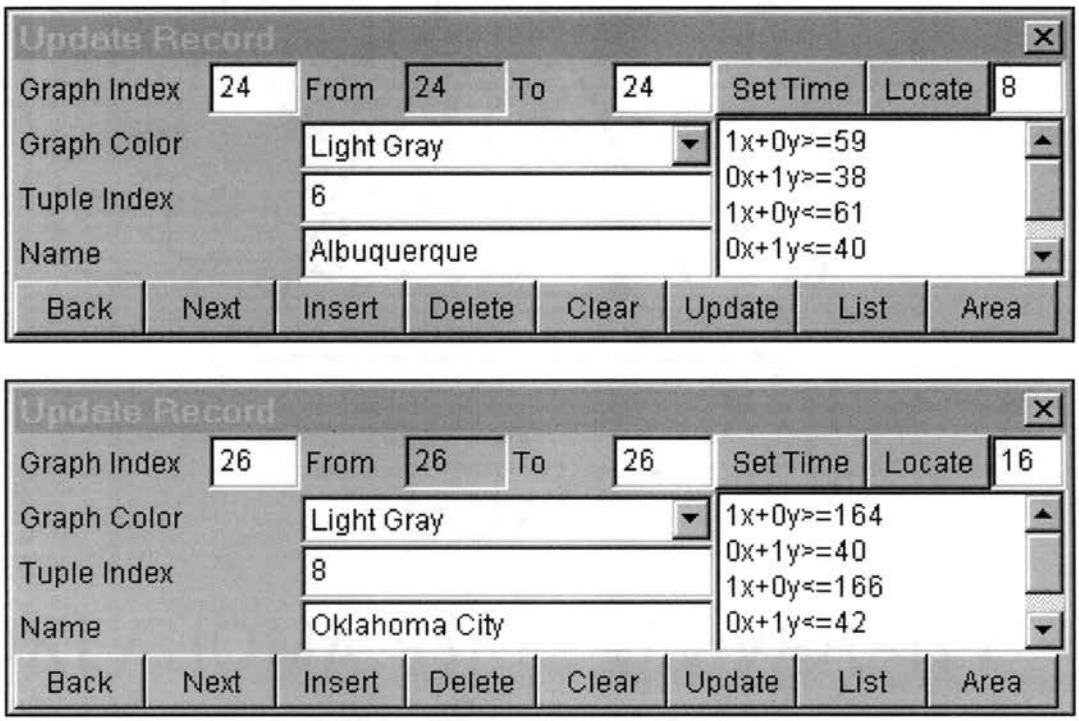


Figure 5.23 Set Time Stamp

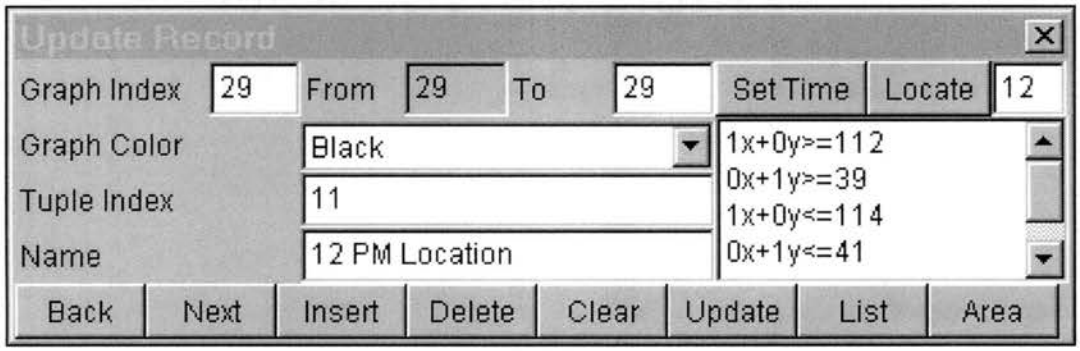


Figure 5.24 12 PM Location

5.8 Vegetation Example

Al is doing a vegetation research. He gets the vegetation map in 1980 and 1998 as shown in Figure 5.25 and Figure 5.26 respectively. He would like to know:

1. What is the possible vegetation in 1989?

2. Which area was once shortgrass prairie and later becomes tallgrass prairie?

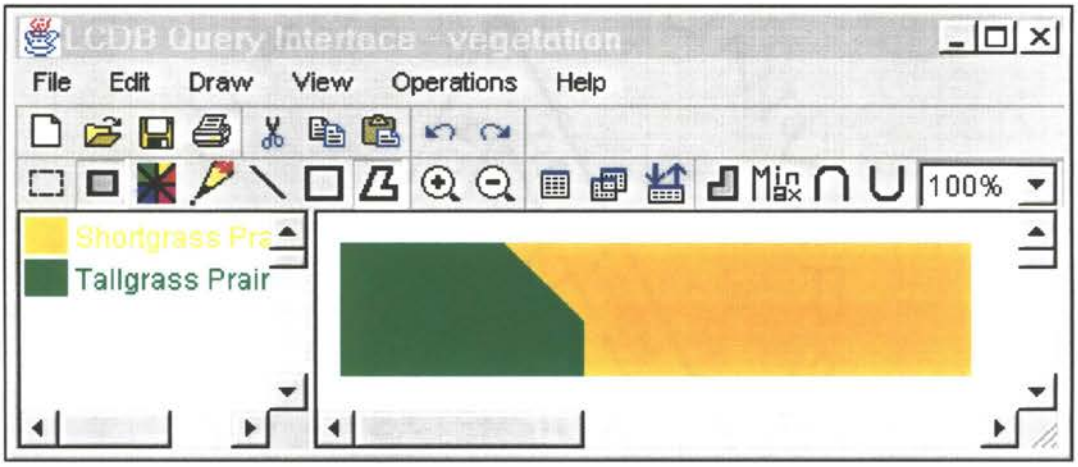


Figure 5.25 Vegetation Map of 1980

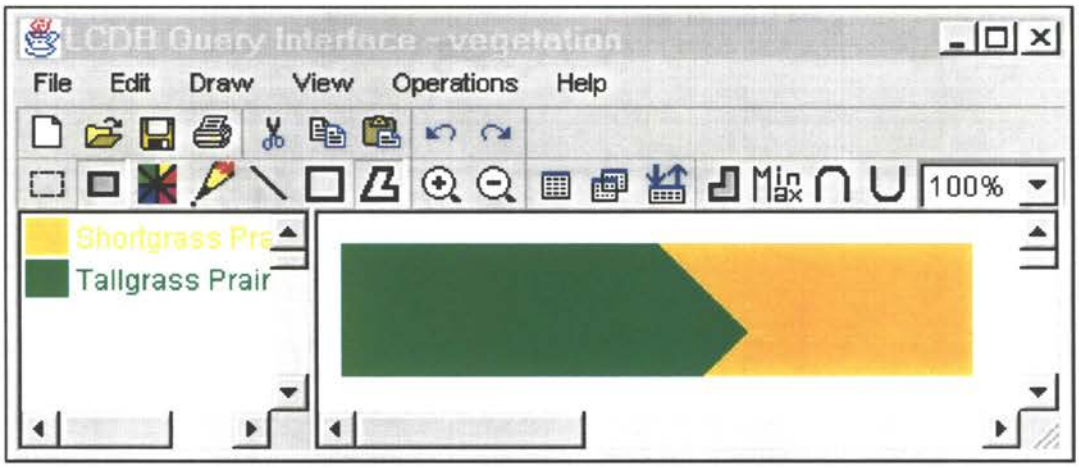


Figure 5.26 Vegetation Map of 1998

To answer the first query, he sets the time stamp of the tallgrass prairie in two different years. In the example, he sets 80 and 98. Then he sets 89 for the time stamp and clicks the *locate* button. The result tuple is identified as a gray region as shown in Figure 5.27.

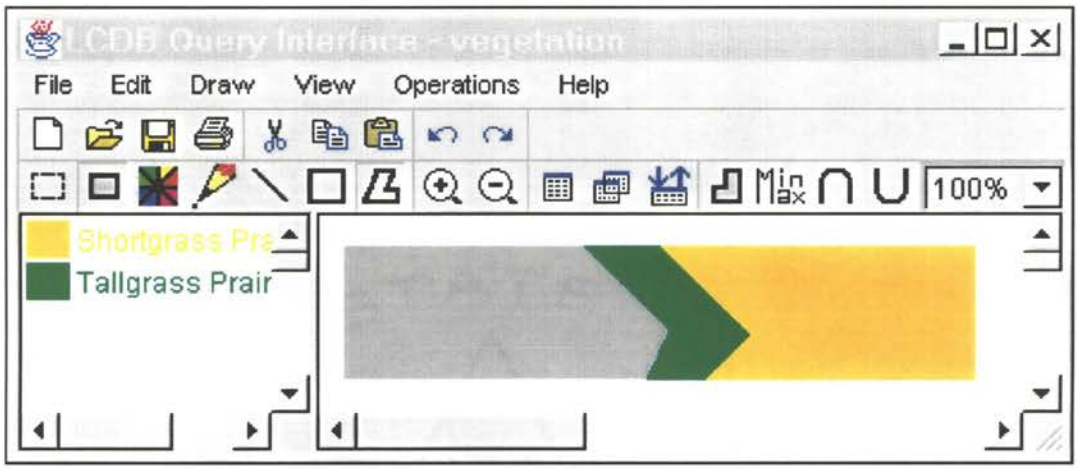


Figure 5.27 Vegetation Map of 1989

To answer the second query, he selects the shortgrass and tallgrass prairie in the feature window and then clicks the *intersection* icon. The result tuple is drawn as a gray region as shown in Figure 5.28.

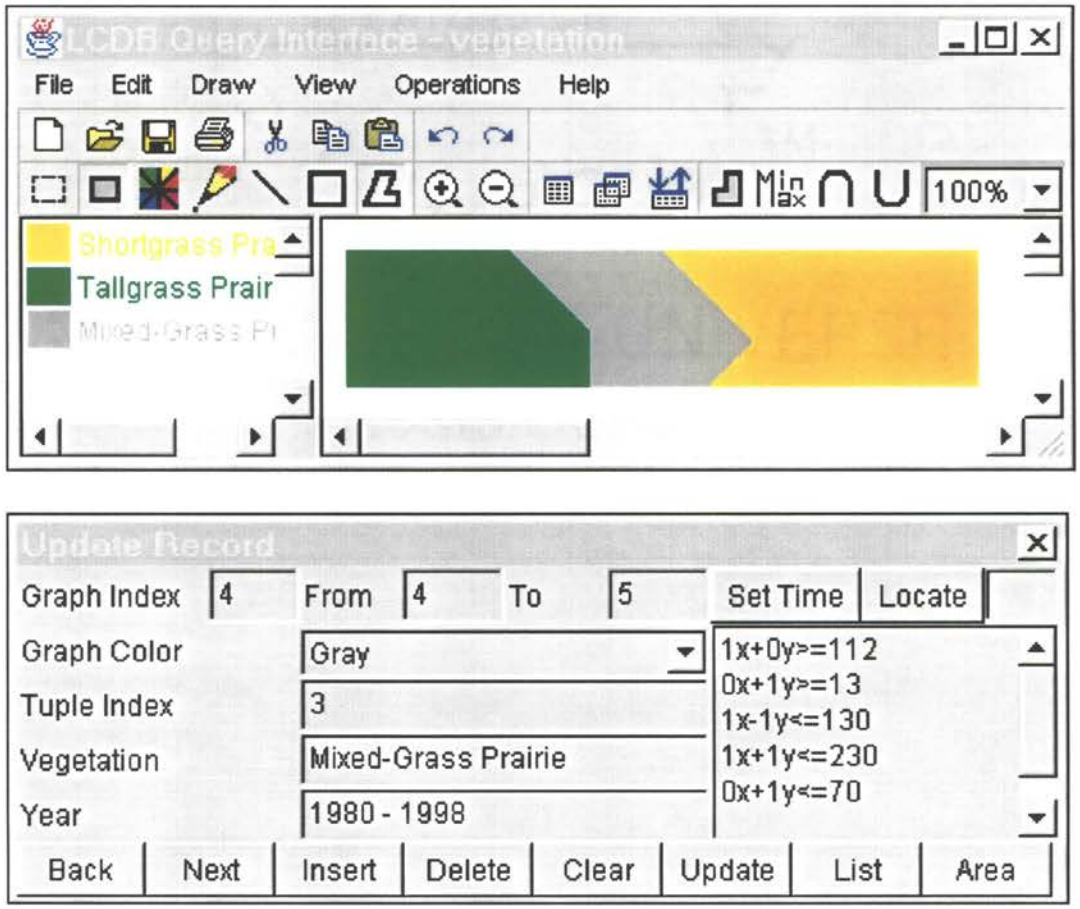


Figure 5.28 Vegetation Map of Mixed-Grass Prairie

CHAPTER VI

CONCLUSION

6.1 Contributions

This dissertation presents a Web-based tax calculator and a stock analysis tool which directly store constraint data into a conventional DBMS (e.g. RDBMS). We also develop the prototype of a constraint database system on CORBA using Java. Built upon top of a RDBMS, the system supports queries to process both constraint data and alphanumeric data. The applications and system have the following features:

- A graphic interface to build up a constraint database.
- A query interface to access the constraint database.
- Constraint data is stored in a RDBMS.
- Applications run on the system are inherently distributed.

Building such a system has the following benefits:

- Constraint data can be incorporated into the existing RDBMS.
- The SQL-like query interface can be used to query constraint data.
- A very simple interface facilitates access to constraint database systems.
- It can be a testbed for the constraint database methodology and further development.
- New constraint applications can be explored in this system.

- Applications can run through Web or CORBA on different platforms seamlessly.

- An implementation guide is introduced.

The graphical interface provides a layer of abstraction for end users. Most users need not to know the underlying linear constraint database used for storing the information and its implementation details. The linear constraint database can represent spatiotemporal data in a uniform framework. It enables the queries which were not possible to evaluate in a relational database. The system can extend the ability of traditional GIS systems such as ARC/INFO. The system can act as an intermediate tier which executes some queries which could not be handled by ARC/INFO. The combination of Java and CORBA facilitates distributed computing applications.

6.2 Open Questions and Future Work

The implementation of the systems is only a starting point for further development. Many issues remain open or under investigation. To allow the system to solve more complex and larger problems efficiently many improvements need to be made. There are numerous practical problems and possible enhancement:

- **Indexing techniques**: In current implementation there is no indexing structure for the constraint data. Though it is convenient to store the constraint data directly into a RDBMS, the indexing structure is necessary to improve performance of database operations in the case of large volume of data or high-dimensional data. Several index structures have been proposed for the constraint database [BCC99a, BCC99b, Fre95, KRVV93, KRVV96, Ram97, ZSI99]. Most of them are based on multidimensional access methods or B-Tree variants and still in theoretical study. Very few attempt has been made to integrate these indexing techniques into the database system because most of them are not general and simple enough to integrate both relational and constraint data.

- **Algebraic operations and normalization**: Not a few algebraic operations are proposed for the constraint data model [BBBC95, BBC98, Gol96, Gol00a, Gol00b, KKR90, KKR95, GRS98]. Many of them are theoretically possible but practically infeasible. We need to refine these algebraic operations to fit the implementation. In current implementation we follow the algebra in [GRS98]. Rectifying a feasible algebra for the further development is important. Constraints are said to be optimal (normal) if there is no redundant or duplicate information. In current implementation we first load constraints into vector format and then store them in the normalized form. This might work well in the small data set. For the large data set the normalized algorithm in [GRSS97] can be used.

- **Efficient constraint satisfying operation**: Constraint satisfaction is a basic operation in a constraint database. Almost every operations involve constraint satisfaction. Our implementation is based on Fourier elimination [Chv83]. To employ the other efficient constraint satisfying algorithms [Cha93, Imb94] is for further development.

- **Optimization of queries**: In current implementation the ability to query constraint data is limited. For complex queries over large numbers of stored constraints a query optimizer is necessary.

- **Comparative performance studies**: Currently, no study on performance comparison between constraint database paradigm and other methods is available. This information is valuable.

- **Temporal application**: Temporal and spatial information can be represented by constraints in a uniform framework. In the current implementation, no temporal animation is available. This ability is for the further enhancement.

- **Query enhancement for other GIS**: It is possible to use the linear constraint database to enchance the querying power of other GIS [Wan99]. The spatial

data in these systems can be coverted into the linear constraint data mode or constraint data can be translated into other GIS. In this way linear constraint database systems act as an intermediate layer which could implement some queries which is not possible in other GIS.

- **Multi-user and concurrency support**: Though our implementation allows multiple access at the same time, the current implementation has no concurrency control mechanism. The problems such as data sharing and transaction processing should be considered.

6.3 Concluding Comment

In this dissertation, we develop the systems and applications based on constraint databases and CORBA. Built upon top of a RDBMS, these systems support queries to process both constraint data and alphanumeric data which reside distributedly. The Web-based tax calculator and the stock analysis tool show the direct incorporation of one-dimensional constraint data into a RDBMS. They can be a basis for a more complete financial tool which is accessible through the combination of the Web, CORBA, and Java. A general query interface to constraint databases has been successfully implemented and tested with GIS applications, linear programming applications, and spatio-temporal applications. These problems are difficult to be solved in the traditional DBMS or GIS. Most spatial queries presented in Section 2.1.2 can be performed through this query interface in a simple way. This research shows that constraint database systems have the ability to handle several types of data uniformly and CORBA provides a platform for distributed computing.

It is still too early to discuss the performance because no query optimization and specific index structure have been built. Theoretically, high-dimensional data can be nicely integrated into the constraint framework, but in practice, it is a challenge to present them for efficient queries. These are the future research as indicated in the previous section.

BIBLIOGRAPHY

[AAK95] Foto N. Afrati, Theodoros Andronikos, and Theodoros G. Kavalieros. On the expressiveness of first-order constraint languages. In *Constraint Databases and Applications*, volume 1034 of *Lecture Notes in Computer Science*, pages 22–39, Friedrichshafen, Germany, September 1995. Springer.

[AAK97] Foto N. Afrati, Theodore Andronikos, and Theodore G. Kavalieros. On the expressiveness of query languages with linear constraints; capturing desirable spatial properties. In *Constraint Databases and Their Applications, Second International Workshop on Constraint Database Systems, CDB'97*, volume 1191 of *Lecture Notes in Computer Science*, pages 105–115. Springer-Verlag, 1997.

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web : From Relations to Semistructured Data and XML.* Morgan Kaufmann, San Francisco, CA, 2000.

[APRS00] Lars Arge, Octavian Procopiuc, Sridhar Ramaswamy, Torsten Suel, Jan Vahrenhold, and Jeffrey Scott Vitter. A unified approach for indexed and non-indexed spatial joins. In *Advances in Database Technology - EDBT 2000, 6th International Conference on Extending Database Technology*, volume 1777 of *Lecture Notes in Computer Science*, pages 413–429, Konstanz, Germany, March 2000. Springer.

[AS91] Walid G. Aref and Hanan Samet. Extending a DBMS with spatial operations. In *Advances in Spatial Databases, Second International Symposium, SSD'91*, pages 322–331, Zürich, Switzerland, August 1991.

[AS94] Walid G. Aref and Hanan Samet. The spatial filter revisited. In *6th International Symposium on Spatial Data Handling*, pages 190–208, Zürich, Switzerland, 1994.

[Bak97] Sean Baker. *CORBA distributed objects using Orbix.* Addison-Wesley, Reading, MA, 1997.

[Bal00] Henry Balen. *Distributed Object Architectures with CORBA.* SIGS Books/Cambridge, New York, NY, 2000.

[BBBC95] Alberto Belussi, Elisa Bertino, Michela Bertolotto, and Barbara Catania. Generalized relational algebra: Modeling spatial queries

in constraint databases. In *Constraint Databases and Applications*, volume 1034 of *Lecture Notes in Computer Science*, pages 40–67, Friedrichshafen, Germany, September 1995. Springer.

[BBC97] Alberto Belussi, Elisa Bertino, and Barbara Catania. Manipulating spatial data in constraint databases. In *Advances in Spatial Databases, 5th International Symposium, SSD'97*, volume 1262 of *Lecture Notes in Computer Science*, pages 115–141, Berlin, Germany, July 1997. Springer.

[BBC98] Alberto Belussi, Elisa Bertino, and Barbara Catania. An extended algebra for constraint databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 10(5):686–705, September 1998.

[BCC99a] Elisa Bertino, Barbara Catania, and Boris Chidlovskii. Approximation techniques for indexing two-dimensional constraint databases. In *Database Systems for Advanced Applications, Proceedings of the Sixth International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 213–220, Hsinchu, Taiwan, April 1999. IEEE Computer Society.

[BCC99b] Elisa Bertino, Barbara Catania, and Boris Chidlovskii. Indexing constraint databases by using a dual representation. In *Proceedings of the 15th International Conference on Data Engineering*, pages 618–627, Sydney, Austrialia, March 1999. IEEE Computer Society Press.

[Bec92] Ludger Becker. *A New Algorithm and a Cost Model for Join Processing with the Grid File.* PhD thesis, Universität-Gesamthochschule Siegen, Siegen, Germany, 1992.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM (CACM)*, 18(10), October 1975.

[Ben79] Jon Louis Bentley. Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering (TSE)*, 5(4), July 1979.

[Ber96] Alex Berson. *Client/Server Architecture.* McGraw-Hill Companies, Inc., New York, second edition, 1996.

[Ber97] Michel Berkelaar. The mixed integer linear program solver package - lp_solve. ftp://ftp.es.ele.tue.nl/pub/lp_solve/, 1997.

[BJM93] Alexander Brodsky, Joxan Jaffar, and Michael J. Maher. Toward practical constraint databases. In *19th International Conference on Very Large Data Base VLDB 1993*, pages 567–580, Dublin, Ireland, August 1993. Morgan Kaufmann.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using R-trees. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 237–246, Washington, D.C., May 1993.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, May 1990.

[BKSS94] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-step processing of spatial joins. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 197–208, Minneapolis, MN, May 1994.

[BM72] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1(3):173–189, 1972.

[BM98] Zvi Bodie and Robert C. Merton. *Finance.* Prentice Hall, Upper Saddle River, NJ, preliminary edition, 1998.

[BR95] Jo-Hag Byon and Peter Z. Revesz. DISCO: A constraint database system with sets. In *Proceedings of the 1st International Database Workshop on Constraint Database Systems (CDB'95)*, volume 1034 of *Lecture Notes in Computer Science*, pages 68–83, Friedrichshafen, Germany, 1995. Springer-Verlag.

[Bri94] Thomas Brinkhoff. *Der Spatial Join in Geo-Datenbanksystemen.* PhD thesis, Ludwig-Maximilians-Universität München, München, Germany, 1994.

[Bro95] Alexander Brodsky. The LyriC language: Querying constraint objects. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 35–46, San Jose, California, June 1995.

[Bro96a] Alexander Brodsky. Constraint database technology for electronic trade with complex objectives. *ACM Computing Surveys*, 28(4es):64–77, December 1996.

[Bro96b] Alexander Brodsky. Constraint databases: Promising technology or just intellectual exercise. In *ACM Workshop on Strategic Directions in Computing Research*, Cambridge, Massachusetts, June 1996. MIT Laboratory for Computer Science.

[BS97] Alexander Brodsky and Victor E. Segal. The C^3 constraint object-oriented database system: An overview. In *Constraint Databases*

and Their Applications, Second International Workshop on Constraint Database Systems, CDB '97, volume 1191 of *Lecture Notes in Computer Science*, pages 134–159, Delphi, Greece, 1997. Springer-Verlag.

[BSCE97] Alexander Brodsky, Victor E. Segal, Jia Chen, and Pavel A. Exarkhopoulo. The ccube constraint object-oriented database system. *Constaints - An International Journal*, 2(3/4), 1997.

[BSCE99] Alexander Brodsky, Victor E. Segal, Jia Chen, and Pavel A. Exarkhopoulo. The ccube constraint object-oriented database system. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 577–579, Philadephia, Pennsylvania, June 1999. ACM Press.

[Buc85] Bruno Buchberger. Gröbner bases: An algorithm method in polynomial ideal theory. In *Multidimensional Systems Theory*, pages 184–232, Dordrecht - Boston - Lancaster, 1985. D. Reidel Publishing Company.

[CDV88] Michael J. Carey, David J. DeWitt, and Scott L. Vandenberg. A data model and query language for exodus. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 413–423, Chicago, Illinois, September 1988.

[Cha93] Vijay Chandru. Variable elimination in linear constraints. *The Computer Journal*, 36(5):463–472, 1993.

[CHLF89] Sei H. Chun, G. E. Hedrick, Huizhu Lu, and D. D. Fisher. A partitioning method for grid file directories. In *Proceedings of the Thirteenth Annual International Computer Software and Applications Conference, COMPSAC 89*, pages 271–277, Orlando, Florida, September 1989. The IEEE Computer Society.

[Chv83] Vasek Chvátal. *Linear Programming.* W. H. Freeman, New York, NY, 1983.

[CM88] Robert W. Colby and Thomas A. Meyers. *The Encyclopedia of Technical Market Indicators.* Dow Jones-Irwin, Homewood, Ill., 1988.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM (CACM)*, 13(6):377–387, June 1970.

[Com95] Douglas Comer. *The Internet book : Everything you need to know about computer networking and how the Internet works.* Prentice Hall, Englewood Cliffs, N.J., 1995.

[Com97] Standard & Poor's Compustat. *PC Plus Basics.* Standard & Poor's Compustat, Englewood, CO, 1997.

[Dan63] George B. Dantzig. Linear programming and extensions. Technical report, Princeton University, Princeton, New Jersey, 1963.

[DCR99] Cynthia Della, Torre Cicalese, and Shmuel Rotenstreich. Behavioral specification of distributed software component interfaces. *IEEE Computer*, pages 46–53, July 1999.

[Del99] Tom Dell. *Dynamic HTML for Webmasters.* AP Professional, San Diego, CA, 1999.

[Dew97] D. Travis Dewire. *Second-Generation Client/Server Computing.* McGraw-Hill, New York, 1997.

[DHSA88] Mehmet Dincbas, Pascal Van Hentenryck, Helmut Simonis, Abderrahmane Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, pages 693–702, Tokyo, Japan, 1988.

[Dob97] Rick Dobson. Dynamic HTML explained, Part I. *Byte*, 22(11):53–54, Nov 1997.

[Dob98] Rick Dobson. Ecmscript : The holy standard. *Byte*, 23(7):47–48, July 1998.

[Eco95] The Economist. Will your next computer be a tin can and a wire? *The Economist*, pages 75 – 76, October 1995.

[Edw98] John Edwards. Changing database market hurts major vendors. *IEEE Computer*, pages 10–11, March 1998.

[Ege94] Max J. Egenhofer. Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, February 1994.

[FAG83] Henry Fuchs, G. D. Abram, and E. D. Grant. Near real-time shaded display of rigid objects. *Computer Graphics*, 17(3):65–72, 1983.

[FKN80] Henry Fuchs, Zvi Meir Kedem, and Bruce F. Naylor. On visible surface generation by a-priori tree structures. *Computer Graphics*, 14(3):124–133, 1980.

[Fla97] David Flanagan. *JavaScript: The Difinition Guide.* O'Reilly & Associates, Inc., Sebastopol, CA, second edition, 1997.

[Fre87] Michael Freeston. The BANG file: A new kind of grid file. In *Proceedings of the ACM SIGMOD 1987 Annual Conference*, pages 260–269, San Francisco, California, May 1987.

[Fre95] Michael Freeston. The application of multi-dimensional indexing methods to constraints. In *Proceedings of the 1st International Database Workshop on Constraint Database Systems (CDB'95)*, volume 1034 of *Lecture Notes in Computer Science*, pages 102–119, Friedrichshafen, Germany, 1995.

[FSR87] Christos Faloutsos, Timos K. Sellis, and Nick Roussopoulos. Analysis of object oriented spatial access methods. In *Proceedings of the ACM SIGMOD 1987 Annual Conference*, pages 426–439, San Francisco, California, May 1987.

[GB91] Oliver Günther and Jeff Bilmes. Tree-based access methods for spatial databases: Implementation and performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 3(3):342–356, Sep 1991.

[GdR00] David Gross and Michel de Rougemont. Uniform generation in spatial constraint databases and applications. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 254–259, Dallas, Texas, May 2000. ACM.

[GG95] Volker Gaede and Oliver Günther. Constraint-based query optimization and processing. In *Proceedings of the 1st International Database Workshop on Constraint Database Systems (CDB'95)*, volume 1034 of *Lecture Notes in Computer Science*, pages 84–101, Friedrichshafen, Germany, 1995.

[GG97] Oliver Günther and Volker Gaede. Oversize shelves: A storage management technique for large spatial data objects. *International Journal of Geographic Information Systems*, 11(1):5–32, Jan 1997.

[GG98] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–321, June 1998.

[Gil97] Paul Gilster. *The Web Navigator*. Wiley Computer Publishing, New York, 1997.

[GL95] Stéphane Grumbach and Zoé Lacroix. Computing queries on linear constraint databases. In *Proceedings of the Fifth International Workshop on Database Programming Languages, Database Programming Languages (DBPL-5)*, Gubbio, Umbria, Italy, September 1995. Springer.

[GLRS00] Stéphane Grumbach, Zoé Lacroix, Philippe Rigaux, and Luc Segoufin. Optimization techniques. In *Constraint Databases*, pages 319–334. Springer Verlag, Berlin/Heidelberg/New York, 2000.

[GM94] Roman Gross and Robert Marti. DeCoR a deductive constraint database system. In *Proceedings of the Tenth Logic Programming Workshop, WLP 94*, pages 65–68, Zürich, Switzerland, October 1994. Institut für Informatik der Universität Zürich.

[Gol96] Dina Q. Goldin. Constraint query algebra. *Constaints - An International Journal*, 1(1), 1996.

[Gol00a] Dina Goldin. Constraint algebras. In *Constraint Databases*, pages 335–342. Springer Verlag, Berlin/Heidelberg/New York, 2000.

[Gol00b] Dina Goldin. Constraint algebras. In *Constraint Databases*, pages 335–342. Springer Verlag, Berlin/Heidelberg/New York, 2000.

[GP00] Charles F. Goldfarb and Paul Prescod. *The XML handbook.* Prentice Hall PRT, Upper Saddle River, NJ, second edition, 2000.

[GQ99] Ian S. Graham and Liam Quin. *XML Specification Guide.* Wiley, New York, 1999.

[GR94] Volker Gaede and Wolf-Fritz Riekert. Spatial access methods and query processing in the object-oriented GIS GODOT. In *AGDM'94 Workshop*, pages 40–52, Delft, The Netherlands, 1994.

[Gra95] B. Granveaud. Study and implementation of spatial indexing methods. Technical report, ECRC, Arabellastr. 17, München, Germany, 1995.

[Gro97] Network Design Group. Java port of lp_solve 2.0. http://www.cs.wustl.edu/~javagrp/help/LinearProgramming.html, 1997.

[GRS98] Stéphane Grumbach, Philippe Rigaux, and Luc Segoufin. The DEDALE system for complex spatial queries. In *Proceedings ACM SIGMOD International Conference on Management of Data SIGMOD 1998*, pages 213–224, Seattle, Washington, June 1998. ACM Press.

[GRSS97] Stéphane Grumbach, Philippe Rigaux, Michel Scholl, and Luc Segoufin. DEDALE, a spatial constraint database. In *International Workshop on Database Programming Languages DBPL-6*, Estes Park, Colorado, USA, 1997.

[GRSS00] Stéphane Grumbach, Philippe Rigaux, Michel Scholl, and Luc Segoufin. The dedale prototype. In *Constraint Databases*, pages 365–382. Springer Verlag, Berlin/Heidelberg/New York, 2000.

[GRSY96] Jonathan Goldstein, Raghu Ramakrishnan, Uri Shaft, and Jie-Bing Yu. Using constraints to query R*-tree. Technical report, Computer Sciences Department, University of Wisconsin - Madison, Madison, Wisconsin, February 1996.

[GS93] Ralf Hartmut Güting and Markus Schneide. Realms: A foundation for spatial data types in database systems. In *Advances in Spatial Databases, Third International Symposium, SSD'93*, Lecture Notes in Computer Science, pages 14–35, Singapore, 1993. Springer-Verlag.

[GST94] Stéphane Grumbach, Jianwen Su, and Christophe Tollu. Linear constraint query languages expressive power and complexity. In *Workshop on Logic and Computational Complexity*, Indianapolis, IN, USA, October 1994. Springer.

[Gün93] Oliver Günther. Efficient computation of spatial joins. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 50–59, Vienna, Austria, April 1993.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD Conference on Management of Data*, pages 47–57, Boston, Massachusetts, June 1984.

[Güt89] Ralf Hartmut Güting. Gral: An extensible relational database system for geometric applications. In *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, pages 33–44, Amsterdam, The Netherlands, 1989.

[Güt94] Ralf Hartmut Güting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399, Oct 1994.

[GVG00] Marc Gyssens, Luc Vandeurzen, and Dirk Van Gucht. Linear-constraint databases. In *Constraint Databases*, pages 199–229. Springer Verlag, Berlin/Heidelberg/New York, 2000.

[GW97] Volker Gaede and Mark Wallace. An informal introduction to constraint database systems. In *Constraint Databases and Applications*, volume 1191 of *Lectures Notes in Computer Science*, pages 7–52, Delphi, Greece, 1997. Springer.

[HCLM90] Laura M. Haas, Walter Chang, Guy M. Lohman, John McPherson, Paul F. Wilms, George Lapis, Bruce G. Lindsay, Hamid Pirahesh, Michael J. Carey, and Eugene J. Shekita. Starburst mid-flight: As the dust clears. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):143–160, March 1990.

[HHLvEB89] Michael R. Hansen, Bo S. Hansen, Peter Lucas, and Peter van Emde Boas. Integrating relational databases and constraint languages. *Computer Languages*, 14(2), 1989.

[Hin85] K. H. Hinrichs. *The Grid File System: Implementation and Case Studies of Applications.* PhD thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, 1985.

[HJ88] Michun Hsu and Thomas E. Cheatham Jr. Rule execution in CPLEX: A persistent objectbase. In *Advances in Object-Oriented Database Systems, 2nd International Workshop on Object-Oriented Database Systems OODBS 1988*, volume 334 of *Lecture Notes in Computer Science*, pages 150–155, Bad Münster am Stein-Ebernburg, Germany, September 1988. Springer.

[HK95] Luis Hermosilla and Gabriel M. Kuper. Towards the definition of a spatial object-oriented data model with constraints. In *Proceedings of the 1st International Database Workshop on Constraint Database Systems (CDB'95)*, volume 1034 of *Lecture Notes in Computer Science*, pages 120–131, Friedrichshafen, Germany, 1995.

[Hot97] Steve Hotopp. Practical issues concerning volatility and its measurement, past and predicted. In *Volatility in the Capital Markets : State-of-the-Art Techniques for Modeling, Managing, and Trading Volatility*, pages 1–33. Glenlake Publishing and Fitzroy Dearborn, Chicago and London, 1997.

[HSH92] Zhexue Huang, Per Svensson, and H. Hauska. Solving spatial analysis problems with GeoSAL, a spatial query language. In *Proceedings of the 6th International Working Conference on Scientific and Statistical Database Management (SSDBM 1992)*, pages 1–17, Monte Verita, Switzerland, 1992.

[HW98] James C. Van Horne and John M. Wachowicz. *Fundamentals of Financial Management.* Prentice Hall, Upper Saddle River, N.J., 1998.

[Imb94] Jean-Louis Imbert. Linear constraint solving in clp-languages. In *Constraint Programming: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science*, pages 108–127, Châtillon-sur-Seine, France, May 1994. Springer Verlag.

[Ind91] Spatial Join Indices. Spatial join indices. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 500–509, Kobe, Japan, April 1991.

[Int99] International Organization for Standardization. *ISO/IEC 13249-3:1999 Information technology - Database languages SQL Multimedia and Application Packages - Part 3: Spatial.* International Organization for Standardization, Geneva, Switzerland, 1999.

[Kan95] Paris. C. Kanellakis. Constraint programming and database languages: A tutorial. In *14th ACM Symposium on Principles of Database Systems (PODS)*, pages 46–53, San Jose, California, 1995.

[KF94] Ibrahim Kamel and Christos Faloutsos. Hilbert R-tree: An improved R-tree using fractals. In *Proceedings of the 20th International Conference on Very Large Data Bases, (VLDB'94)*, pages 500–509, Santiago de Chile, Chile, September 1994. Morgan Kaufmann.

[KF95] Ed Krol and Paula Ferguson. *The Whole Internet for Windows 95 : User's Guide & Catalog.* O'Reilly & Associates, Sebastopol, CA, 1995.

[KG94a] Paris. C. Kanellakis and D. Q. Goldin. Constraint programming and database query languages. Technical Report CS-94-31, Department of Computer Science, Brown University, Providence, Rhode Island, 1994.

[KG94b] Paris. C. Kanellakis and Dina. Q. Goldin. Constraint programming and database query languages. In *Symposium on Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 96–120, Sendai, Japan, April 1994.

[Kha96] Rohit Khare. The next big thing on the Web. *Byte*, page 42, July 1996.

[Kha97] Rohit Khare. XML: A door to automated Web applications. *IEEE Internet Computing*, pages 78–87, Jul/Aug 1997.

[KKR90] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint query languages. In *9th ACM Symposium on Principles of Database Systems (PODS 1990)*, pages 299–313, Nashville, Tennessee, April 1990. ACM Press.

[KKR95] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.

[KLP00] Gabriel M. Kuper, Leonid Libkin, and Jan Paredaens. *Constraint Databases.* Springer Verlag, Berlin/Heidelberg/New York, 2000.

[Knu73] Donald E. Knuth. *The Art of Computer Programming. Searching and Sorting*, volume 3. Addison-Wesley, Reading, MA, 1973.

[Kor98] Jukka Korpela. Lurching Toward Babel: HTML, CSS, and XML. *Computer*, 31(7), July 1998.

[Kro94] Ed Krol. *The Whole Internet User's Guide & Catalog.* O'Reilly & Associates, Sebastopol, CA, second edition, 1994.

[KRVV93] Paris C. Kanellakis, Sridhar Ramaswamy, Darren Erik Vengroff, and Jeffrey Scott Vitter. Indexing for data models with constraints and classes. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1993*, pages 223–243, Washington DC, May 1993. ACM Press.

[KRVV96] Paris C. Kanellakis, Sridhar Ramaswamy, Darren Erik Vengroff, and Jeffrey Scott Vitter. Indexing for data models with constraints and classes. *Journal of Computer and System Sciences*, 52(3):589–612, 1996.

[KRW98] Pradip Kanjamala, Peter Z. Revesz, and Yonghui Wang. Mlpq/gis: A gis using linear constraint databases. In *9th International Conference On Management Of Data (COMAD' 98)*, Hyderabad, India, December 1998. Tata-McGraw-Hill.

[KS86] Hans-Peter Kriegel and Bernhard Seeger. Multidimensional order preserving linear hashing with partial expansions. In *International Conference on Database Theory ICDT'86*, volume 243 of *Lecture Notes in Computer Science*, pages 203–220, Rome, Italy, September 1986. Springer-Verlag.

[KS88] Hans-Peter Kriegel and Bernhard Seeger. Plop-hasing: A grid file without directory. In *IEEE 4th International Conference on Data Engineering*, 1988.

[KSS87] Robert A. Kowalski, Fariba Sadri, and Paul Soper. Integrity checking in deductive databases. In *13th International Conference on Very Large Data Bases*, pages 61–69, Brighton, England, September 1987. Morgan Kaufmann.

[LLOW91] Charles Lamb, Gordon Landis, Jack A. Orenstein, and Danel Weinreb. The objectstore system. *Communications of the ACM (CACM)*, 34(10):50–63, October 1991.

[LR94] Ming-Ling Lo and Chinya V. Ravishankar. Spatial joins using seeded trees. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 209–220, Minneapolis, Minnesota, May 1994.

[LS89] David B. Lomet and Betty Salzberg. hB-tree : A robust multi-attribute search structure. In *Proceedings of the Fifth International Conference on Data Engineering*, pages 296–304, Los Angeles, California, February 1989.

[LS90] David B. Lomet and Betty Salzberg. The hB-tree: A multiattribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems (TODS)*, 15(1):625–658, March 1990.

[Mey00] Eric A. Meyer. *Cascading Style Sheets : The Definitive Guide.* O'Reilly, Sebastopol, CA, May 2000.

[MFDG98] Scott Mace, Udo Flohr, Rick Dobson, and Tony Graham. Weaving a better Web. *Byte*, 23(3):58–68, March 1998.

[Nar94] Atul Narkhede. Fast polygon triangulation algorithm based on seidel's algorithm. Implementation Report, Department of Computer Science, The University of North Carolina at Chapel Hill, Chapel Hill, NC, 1994.

[NHS84] Jürg Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. The Grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems (TODS)*, 9(1):38–71, March 1984.

[NS87] Randal C. Nelson and Hanan Samet. A population analysis for hierarchical data structures. In *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference*, pages 270–277, San Francisco, California, May 1987.

[NT93] Evar D. Nering and Albert W. Tucker. *Linear Programs and Related Problems.* Academic Press, Inc., 1993.

[NV96] Christian Neuss and Johan Vromans. *The Webmaster's Handbook : Perl Power for your Web Server.* International Thomson Computer Press, London/Boston, 1996.

[Obj00] Object-Oriented Concepts Inc. *ORBacus for C++ and Java.* Object-Oriented Concepts, Inc., 44 Manning Road, Billerica, MA 01821, September 2000.

[OH98] Robert Orfali and Dan Harkey. *Client/Server Programming with Java and CORBA.* Wiley Computer Publishing, New York, NY, second edition, 1998.

[OHE96] Robert Orfali, Dan Harkey, and Jeri Edwards. *The Essential Client/Server Survival Guide.* Wiley Computer Publishing, New York, NY, second edition, 1996.

[OHE97a] Robert Orfali, Dan Harkey, and Jeri Edwards. CORBA, Java, and object Web. *Byte*, 22(10):95–100, Oct 1997.

[OHE97b] Robert Orfali, Dan Harkey, and Jeri Edwards. *Instant CORBA.* Wiley Computer Publishing, New York, NY, 1997.

[OM88] Jack A. Orenstein and Frank Manola. Probe: Spatial data modeling and query processing in an image database application. *IEEE Transactions on Software Engineering*, 14(5):611–628, May 1988.

[Ope99] Open GIS Consortium Inc. *OpenGIS Simple Features Sepcification for SQL Revision 1.1.* Open GIS Consortium, Inc., May 1999.

[O'R94] Joseph O'Rourke. *Computational geometry in C.* Cambridge University Press, Cambridge/New York/Melbourne, 1994.

[O'R98] Joseph O'Rourke. *Computational Geometry in C.* Cambridge University Press, Cambridge/New York/Melbourne, 2nd edition, September 1998.

[Ore86] Jack A. Orenstein. Spatial query processing in an object-oriented database system. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, pages 326–333, Washington, D.C., May 1986.

[Oto86] Ekow J. Otoo. Balanced multidimensional extendible hash tree. In *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 100–113, Cambridge, Massachusetts, March 1986. ACM Press.

[Ouk85] M. Aris Ouksel. The interpolation-based grid file. In *Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 20–27, Portland, Oregon, March 1985. ACM Press.

[Par95] Jan Paredaens. Spatial databases, the final frontier. In *Database theory - ICDT'95 : 5th International Conference*, Lecture Notes in Computer Science, pages 14–32. Springer-Verlag, Berlin/Heidelberg/New York, 1995.

[PdBG94] Jan Paredaens, Jan Van den Bussche, and Dirk Van Gucht. Towards a theory of spatial database queries. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1994)*, pages 279–288, Minneapolis, Minnesota, May 1994. ACM Press.

[Pra98] Kanjamala P. Pradip. Implementation a geographic information system using linear constraint databases. Master's thesis, University of Nebraska, Lincoln, Nebraska, July 1998.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, Berlin/Heidelberg/New York, 1985.

[PTSE95] Dimitris Papadias, Yannis Theodoridis, Timos K. Sellis, and Max J. Egenhofer. Topological relations in the world of minimum bounding rectangles: A study with R-trees. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 92–103, San Jose, California, May 1995.

[PWGB98] Doug Pedrick, Jonathan Weedon, Jon Goldberg, and Erik Bleifield. *Programming with VisiBroker : A Developer's Guide to VisiBroker for Java.* Wiley Computer Publishing, New York, NY, 1998.

[Ram97] Sridhar Ramaswamy. Efficient indexing for constraints and temporal databases. In *Database Theory - ICDT '97, 6th International Conference*, volume 1186 of *Lecture Notes in Computer Science*, pages 419–431, Delphi, Greece, January 1997. Srpinger.

[Ram00] Sridhar Ramaswamy. I/o-efficient algorithms for cdbs. In *Constraint Databases*, pages 343–360. Springer Verlag, Berlin/Heidelberg/New York, 2000.

[RCKL00] Peter Z. Revesz, Rui Chen, Pradip Kanjamala, Yiming Li, Yuguo Liu, and Yonghui Wang. The mlpq/gis constraint database system. In *ACM SIGMOD Conference 2000*, page 601, Dallas, Texas, May 2000.

[Rev97] Peter Z. Revesz. Problem solving in the DISCO constraint database system. In *Constraint Databases and Their Applications, Second International Workshop on Constraint Database Systems, CDB '97*, volume 1191 of *Lecture Notes in Computer Science*, pages 302–315, Delphi, Greece, 1997. Springer-Verlag.

[Rev00a] Peter Z. Revesz. Datalog and constraints. In *Constraint Databases*, pages 155–170. Springer Verlag, Berlin/Heidelberg/New York, 2000.

[Rev00b] Peter Z. Revesz. The disco system. In *Constraint Databases*, pages 383–389. Springer Verlag, Berlin/Heidelberg/New York, 2000.

[RFS88] Nick Roussopoulos, Christos Faloutsos, and Timos Sellis. An efficient pictorial database system for psql. *IEEE Transactions on Software Engineering*, 14(5):639–650, May 1988.

[RL84] Nick Roussopoulos and Daniel Leifker. An introduction to PSQL: A pictoral structured query language. In *IEEE Workshop on Visual Language*, 1984.

[RL85] Nick Roussopoulos and Daniel Leifker. Direct spatial search on pictorial databases using packed R-trees. In *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, pages 17–31, Austin, Texas, May 1985.

[RL97] Peter Z. Revesz and Yiming Li. Mlpq: A linear constraint database system with aggregate operators. In *Proceedings of the International Database Engineering and Applications Symposium, IDEAS 1997*, pages 132–137, Montreal, Candada, August 1997. Concordia University.

[Rob81] John T. Robinson. A search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18, Ann Arbor, Michigan, 1981. ACM Press.

[Rul99] Jeff Rule. *Dynamic HTML : The HTML Developer's Guide.* Addison-Wesley, Reading, MA, 1999.

[Sam95] Hanan Samet. Spatial data structures. In *Modern Database Systems: The Object Model, Interoperability, and Beyond*, pages 338–360. Addison-Wesley/ACM Press, New York, N.Y., 1995.

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Son, Inc., 1986.

[Sei91] Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1(1):51–64, 1991.

[Sie00] Jon Siegel. *Corba 3 Fundamentals and Programming.* John Wiley & Sons, New York, NY, 2000.

[SK88] Bernhard Seeger and Hans-Peter Kriegel. Techniques for design and implementation of efficient spatial access methods. In *Fourteenth International Conference on Very Large Data Bases*, pages 360–371, Los Angeles, California, 1988. Morgan Kaufmann.

[SK90] Bernhard Seeger and Hans-Peter Kriegel. The buddy-tree: An efficient and robust access method for spatial data base systems. In *16th International Conference on Very Large Data Bases*, pages 590–601, Brisbane, Queensland, Australia, August 1990. Morgan Kaufmann.

[SK91] Michael Stonebraker and Greg Kemnitz. The Postgres next generation database management system. *Communications of the ACM (CACM)*, 34(10):78–92, October 1991.

[SKS97] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts.* The McGraw-Hill Companies, Inc., New York, 1997.

[SRF87] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The R^+-tree: A dynamic index for multi-dimensional objects. In *12th International Conf. on VLDB*, pages 507–518, Brighton, England, Sep 1987.

[SSH86] Michael Stonebraker, Timos K. Sellis, and Eric N. Hanson. An analysis of rule indexing implementations in data base systems. In *Expert*

Database Systems, Proceedings From the First International Conference, pages 465–476, Charleston, South Carolina, April 1986. Benjamin Cummings.

[SV89] Michel Scholl and Agnès Voisard. Thematic map modeling. In *Design and Implementation of Large Spatial Databases, First Symposium SSD'89*, volume 409 of *Lecture Notes in Computer Science*, pages 167–192, Santa Barbara, California, July 1989. Sringer-Verlag.

[Tam82] Markku Tamminen. The extendible cell method for closest point problems. *BIT*, 22(1):27–41, 1982.

[TB98] Richard Jack Teweles and Edward S. Bradley. *The Stock Market.* John Wiley & Sons, Inc., New York, NY, 7th edition, 1998.

[TM84] S. B. Tor and Alan E. Middleditch. Convex decomposition of simple polygons. *ACM Transactions on Graphics (TOG)*, 3(4):244–265, October 1984.

[VD98] Andreas Vogel and Keith Duddy. *Java Programming with CORBA.* Wiley Computer Publishing, New York, NY, second edition, 1998.

[VGG95] Luc Vandeurzen, Marc Gyssens, and Dirk Van Gucht. On the desirability and limitations of linear spatial database model. In *Advances in Spatial Databases, 4th International Symposium, SSD'95*, volume 951 of *Lecture Notes in Computer Science*, pages 14–28, Portland, Maine, USA, August 1995. Springer.

[VGG98] Luc Vandeurzen, Marc Gyssens, and Dirk Van Gucht. An expressive language for linear spatial database queries. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System PODS 1998*, pages 109–118, Seattle, Washington, June 1998. ACM Press.

[vO90] Peter van Oosterom. *Reactive Data Structures for Geographic Information Systems.* PhD thesis, University of Leiden, Leiden, The Netherlands, 1990.

[Wal91] Mark Wallace. Compiling integrity checking into update procedures. In *IJCAI'91*, Sydney, Australia, 1991.

[Wan99] Yonghui Wang. A query enhancement method for the ARC/INFO GIS system. Master's thesis, University of Nebraska, Lincoln, Nebraska, January 1999.

[Wig95] Richard W. Wiggins. *The Internet for Everyone : A Guide for Users and Providers.* McGraw-Hill, New York, 1995.

[WNS97] Mark Wallace, Stefano Novello, and Joachim Schimpf. ECL^{i}PS^e: A platform for constraint logic programming. Technical report, William Penney Laboratory, IC-Parc, Imperial College, London, August 1997.

[WS99] Mark Wallace and Joachim Schimpf. Eclipse: Declarative specification and scaleable implementation. In *Practical Aspects of Declarative Languages, First International Workshop, PADL '99*, volume 1551 of *Lecture Notes in Computer Science*, pages 365–366, San Antonio, Texas, January 1999. Springer.

[ZL00] Ron Zahavi and David S. Linthicum. *Enterprise Application Integration with CORBA Component and Web-Based Solutions.* Wiley, New York, NY, 2000.

[ZSI99] Hongjun Zhu, Jianwen Su, and Oscar H. Ibarra. An index structure for spatial joins in linear constraint databases. In *Proceedings of the 15th International Conference on Data Engineering*, pages 636–643, Sydney, Austrialia, March 1999. IEEE Computer Society Press.

APPENDIX A

GLOSSARY

Cascading Style Sheets (CSS) A style sheet mechanism that is developed to meet the needs of Web designers and users.

Common Gateway Interface (CGI) An interface for running external programs under a Web server.

Common Object Request Broker Architecture (CORBA) An architecture developed by Object Management Group (OMG) to provide portability and interoperability of distributed computing objects over heterogeneous networks.

Conjunctive Normal Form (CNF) A formula that is either a fundamental disjunction or a conjunction of two or more fundamental disjunctions.

Constraint Logic Programming (CLP) Programming that augments Prolog by adding constraints to the clauses.

Database Management System (DBMS) A software system that consists of a collection of interrelated data and a set of programs to control and manage those data efficiently. The collection of data, usually referred to as the database, contains information about one particular enterprise.

Disjunctive Normal Form (DNF) A formula that is either a fundamental conjunction or a disjunction of two or more fundamental conjunctions.

Document Object Model (DOM) A specification defines how HTML objects are exposed to the scripting languages.

Dynamic HyperText Markup Language (DHTML) Web pages with dynamic content that contains Cascading Style Sheets (CSS), HyperText Markup Language (HTML), and JavaScript.

Extensible Markup Language (XML) A language designed to enable the use of

SGML on the World Wide Web.

Fundamental Conjunction A literal or the conjunction of two or more literals.

Fundamental Disjunction A literal or the disjunction of two or more literals.

Generalized Database A finite set of generalized relations.

Generalized Relation A finite set of generalized tuples.

Generalized Tuple A tuple that contains a conjunction of constraints on a predefined number of variables.

Geographic Information System (GIS) A computer-based technology that captures, stores, analyzes, and displays geospatial information.

Hypermedia A multimedia system that is enriched with links to other documents.

Hypertext A system for storing pages of textual information that contain links to other documents.

HyperText Markup Language (HTML) A document formatting language used to build Web pages.

HyperText Transfer Protocol (HTTP) A protocol for transferring Web documents.

Interface Definition Language (IDL) A language used to specify interfaces of objects independent of particular programming language representations.

International Standard Organization (ISO) An international organization that defines international standards.

The Internet The collection of global networks.

Internet Inter-ORB Protocol (IIOP) A TCP/IP protocol with some CORBA-defined message exchanges.

Intranet A network that is contained within an enterprise.

Multidimensional Access Method The large class of access methods that support searches in spatial databases.

Object Management Architecture (OMA) The architectural framework of Object Management Group.

Object Management Group (OMG) A consortium of computer industry com-

panies that establishes the middleware standard for distributed computing and promotes its usage.

Object Query Language (OQL) A query language used in object-oriented databases.

Object Request Broker (ORB) A key component of Common Object Request Broker Architecture (CORBA) which encompasses all of the communication infrastructure necessary to identify and locate objects, handle connection management and deliver data.

Object Web Augmenting the Web infrastructure with CORBA/Java.

Point Access Methods (PAM) Methods that primarily focus on spatial searches on point databases.

Portable Object Adapter (POA) An object adapter defined by OMG to support a portable connection between the servant and the ORB among different vendors.

Relational Database Management System (RDBMS) A database management system that accesses data defined in the relation data model.

Spatial Access Method (SAM) Methods used to index or access extended objects, such as lines, polygons, or even higher-dimensional polyhedra.

Spatial Join A spatial operation that computes all pairs of objects that satisfy the specified spatial predicate.

Standard Generalized Markup Language (SGML) An international standard for the format of text and documents.

Structured Query Language (SQL) A standard for the data access and definition language in relational databases.

SQL3 SQL standards that are currently in draft and expected for release in 1998.

SQL3/MM SQL standard with Multimedia extension.

Topologically Integrated Geographic Encoding and Referencing (TIGER) A geospatial file format used in U.S. Bureau of the Census.

Transmission Control Protocol/Internet Protocol (TCP/IP) A set of communication protocols developed to allow cooperating computers to share resources across networks and the Internet.

Universal Resource Locator (URL) An indicator used to specify the location of the information.

Web Infrastructure A service or facility that is fundamental to the Web.

Web Object The integration of distributed objects and the Web.

World Wide Web A distributed hypermedia information delivery system, also known as WWW, Web or W3.

APPENDIX B

LIST OF ACRONYMS

BSP	Binary Space Partitioning
CAD	Computer-Aided Design
CERN	European Laboratory for Particle Physics
CGI	Common Gateway Interface
CHIP	Constraint Handling in Prolog
CLP	Constraint Logic Programming
CNF	Conjunctive Normal Form
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheet
DBMS	Database Management System
DHTML	Dynamic HyperText Markup Language
DNF	Disjunctive Normal Form
GIS	Geographic Information System
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
ISO	International Standard Organization
MBB	Minimum Bounding Box
NCSA	National Center for Supercomputer Applications
OLAP	On-Line Analytic Processing
OMA	Object Management Architecture

OMG	Object Management Group
OQL	Object Query Language
ORB	Object Request Broker
PAM	Point Access Methods
POA	Portable Object Adapter
RDBMS	Relational Database Management System
SAM	Spatial Access Method
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TIGER	Topologically Integrated Geographic Encoding and Referencing
URL	Universal Resource Locator
VLSI	Very Large Scale Integration
WWW	World Wide Web
W3	World Wide Web
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

APPENDIX C

SETUP AND RUNNING OF THE APPLICATION

A stock application as an example shows how to set up and run the applications in this dissertation.

C.1 System Requirements

To run the program we require:

- a Web server - *Apache HTTP server*
- a Java programming environment - *SUN Java 2 SDK*
- an object request broker - *Object Oriented Concept Orbacus*
- a database system - *Hughes Technologies mSQL*

Java API for mSQL is not available in Hughes Technologies mSQL. We need to write Java programs to access mSQL or use available Java API for mSQL. We use **mSQL.class** in our implementation. mSQL.class is a Java class library that allows applications or applets to access and manipulate mSQL databases. It implements most of the mSQL C API. To demonstrate the portability of Java/CORBA implementation we run our applications on UNIX and Microsoft Windows platforms.

C.2 Setting Environment Variables

We set up the **CLASS_PATH** to the directory where we extract Orbacus Java class library, `OB.jar` and mSQL Java class library, `msql.jar` so that the Web clients can access those classes through HTTP. Here is an example for the C Shell on UNIX:

```
setenv CORBA_PATH ~/orbacus
setenv CLASSPATH .:$CORBA_PATH/classes
```

The case for the Korn Shell and the Bourne Shell on UNIX can be done as follows:

```
CORBA_PATH=$HOME/orbacus; export CORBA_PATH
CLASSPATH=.:$CORBA_PATH/classes; export CLASSPATH
```

Our sample for Windows 95/98/Me/NT/2000 is:

```
set CORBA_PATH=d:\htdocs\corba
set CLASSPATH=.;%CORBA_PATH%\classes
```

We can set the specified paths permanently by adding them into `autoexec.bat` on Windows 95/98/Me or specifying them in User Variables in the system environment of Windows NT/2000.

C.3 Creating the Database

To create the database using in our example we follow the following steps:

1. Start the mSQL server which is `msql2d`.

2. Create the **Invest** database. The command is `msqladmin create Invest`.

3. Create the database tables. This is our creating session:

```
> msql Invest
mSQL > create table company (
-> name char(30) not null,
-> ticker char(5) not null,
-> address char(40) not null,
-> city char(20) not null,
-> state char(11) not null,
-> zipcode char(10) not null,
-> phone char(14) not null,
-> industry char(30) not null,
-> exchange char(6) not null,
-> employees int not null,
-> group char(40) not null) \g
```

```
Query OK.  1 row(s) modified or retrieved.

mSQL > create table finance (
->  ticker char(5) not null,
->  year int not null,
->  eps real not null,
->  low_price real not null,
->  high_price real not null,
->  close_price real not null,
->  pe_ratio real not null) \g

Query OK.  1 row(s) modified or retrieved.
```

4. Input data. **getCoFinance** and **getCoInfo** are programs to read data into the database.

C.4 Compiling the Programs

We use **make** utility to simplify the compilation. Here is the sample of our `Makefile` for UNIX:

```
all:
        jidl --no-comments Invest.idl
        javac -d ../../classes *.java

clean:
        rm -rf Invest
        rm -rf ../../classes/Invest
```

The `make.bat` batch file is used on Windows 95/98/Me/NT/2000:

```
@echo off
rem Makefile

if "%1"=="" goto all
if "%1"=="clean" goto clean
goto usage

:all
```

```
echo Building the Invest example ...
call jidl --no-comments Invest.idl
javac -d ../../classes Invest\*.java *.java
goto end

:clean
del Invest\*.*
rmdir Invest
goto end

:usage
echo Usage: make [clean]

:end
```

To compile the program type the `make` command.

C.5 Creating a Web Page

Three HTML Web pages which embed the necessary applets and parameters for our application are created. **Invest.html** is the main page which contains **Quote.html** and **Profile.html** or **PickStock.html**.

```
<html>
<head>
<title>Financial Analysis</title>
</head>
<head>
<script language="JavaScript">
function pickPage() {
  var facto = frames['up'].document.criteria;

  for (var i = 0; i < facto.enquiry.length; i++)
    if (facto.enquiry[i].selected) act = facto.enquiry[i].value;
  if (act == "tickerfinder") {
    msg = "Ticker Finder";
    frames['low'].window.location = "Profile.html";
  }
  if (act == "profile") {
    msg = "Company Profile";
    frames['low'].window.location = "Profile.html";
  }
```

```
  if (act == "price-earnings") {
    msg = "P/E (Price Earnings) Ratio Analysis";
    frames['low'].window.location = "PickStock.html";
  }
  if (act == "volatility") {
    msg = "Volatility Analysis";
    frames['low'].window.location = "PickStock.html";
  }
}
</script>
<frameset rows="93,*" framespacing=0>
  <frame name="up"  src="Quote.html" frameborder=0 scrolling=no>
  <frame name="low" src="Profile.html" frameborder=0>
</frameset>
</body>
</html>
```

Quote.html use fuctions written in JavaScript to decide to load either **Profile.html** or **PickStock.html**.

```
<html>
<head>
<title>Quote Financial Database</title>
</head>
<body>
<script language="JavaScript">
function clearup() {
  document.criteria.company.value="";
  document.criteria.ticker.value="";
  document.criteria.pe_ratio.value="";
  document.criteria.percent.value="";
}
</script>
<form name="criteria">
  <table border=1>
    <tr>
    <td align=center>Company
    <td align=center>Ticker
    <td align=center>Year
    <td align=center>
      <select name="pe_op">
        <option value="gt-pe"><b>P/E&gt;</b>
        <option value="lt-pe"><b>P/E&lt;</b>
```

```
      </select>
    <td align=center>Percent
    <td align=center>Option
    <td align=center>
      <input type=button value="Submit"
          OnClick="parent.pickPage()">
    </tr>
    <tr>
    <td><input type=text name="company" size=15></td>
    <td><input type=text name="ticker" size=5></td>
    <td>
      <select name="year">
        <option value=98>98  <option value=97>97
        <option value=96>96  <option value=95>95
        <option value=94>94  <option value=93>93
        <option value=92>92  <option value=91>91
        <option value=90>90  <option value=89>89
      </select>
    </td>
    <td><input type=text name="pe_ratio" size=7></td>
    <td><input type=text name="percent" size=5></td>
    <td>
      <select name="enquiry" OnChange="parent.pickPage()">
        <option value="tickerfinder">Find Ticker
        <option value="profile">Company Profile
        <option value="price-earnings">P/E Ratio
        <option value="volatility">Volatility
      </select>
    </td>
    <td align=center>
      <input type=button value="Clear" name="clearbutton"
          OnClick="clearup()">
    </tr>
  </table>
</form>
</body>
</html>
```

Profile.html embeds **Profile** applet.

```
<html>
<head>
<title>Company Profile</title>
</head>
```

```
<script language="JavaScript">
function getProfile() {
  ticker = parent.frames['up'].document.criteria.ticker.value;
  if (ticker != null)
    parent.frames['up'].document.criteria.company.value =
                        document.Stock.getProfile(ticker, 98);
}
</script>
<body onLoad="getProfile()">
<center>
<b>Company Financial Highlight</b>
<hr>
<applet name="Stock" mayscript code=StockApplet.class
       width=630 height=350 codebase=classes>
  <param name=org.omg.CORBA.ORBClass value=com.ooc.CORBA.ORB>
  <param name=org.omg.CORBA.ORBSingletonClass
        value=com.ooc.CORBA.ORBSingleton>
</applet>
<hr>
</center>
</body>
</html>
```

PickStock.html contains **PickStock** applet.

```
<html>
<head>
<title>Company Profile</title>
</head>
<script language="JavaScript">
function getList() {
  var facto = parent.frames['up'].document.criteria;
  for (i = 0; i < facto.year.length; i++)
    if (facto.year[i].selected) year = facto.year[i].value - 0;
  pe_ratio = parent.frames['up'].document.criteria.pe_ratio.value;
  if (pe_ratio == null || pe_ratio == "") pe_ratio = 0.0;
  else pe_ratio = pe_ratio - 0.0;
  percent = parent.frames['up'].document.criteria.percent.value;
  if (percent == null || percent == "") percent = 0.0;
  else percent = percent * 0.01;
  if (facto.pe_op[0].selected)
    pe_op = facto.pe_op[0].value;
  else pe_op = facto.pe_op[1].value;
  if (pe_op == "gt-pe") op = ">";
```

```
  else op = "<";
  if (facto.enquiry[3].selected)
    document.Stock.volatility(year, op, pe_ratio, percent);
  else
    document.Stock.getList(year, op, pe_ratio);
}
</script>
<body onLoad="getList()">
<center>
<b>Companies which meet criteria</b>
<hr>
<applet name="Stock" code=StockList.class width=630 height=4870
    codebase=classes>
  <param name=org.omg.CORBA.ORBClass value=com.ooc.CORBA.ORB>
  <param name=org.omg.CORBA.ORBSingletonClass
        value=com.ooc.CORBA.ORBSingleton>
</applet>
</center>
</body>
</html>
```

In order to use *Orbacus* ORB instead of ORB built in the Web browser we must have the following settings:

```
  <param name=org.omg.CORBA.ORBClass
        value=com.ooc.CORBA.ORB>
  <param name=org.omg.CORBA.ORBSingletonClass
        value=com.ooc.CORBA.ORBSingleton>
```

C.6 Running the Programs

Here is the steps to run the programs:

1. Run **java InvestServer** to start the server.

2. Access the `Invest.html` through **Appletviewer** or the Web browser such as **Internet Explorer** or **Netscape Navigator** to start the client.

APPENDIX D

INPUT FILE FORMAT FOR THE QUERY INTERFACE

The plain text format contains 5 parts: database name, relations, attributes, tuple data, and graph.

```
Campus
Description Color
Building,-14336

Name,Address,Zip_Code,Usage
0 0 0 0
Math Science,Math Science,74078,Office & Classroom

Tuple-ID Color
0 -14336
X   Y   Th   RHS
1   0   >=   186
0   1   >=   277
1   0   <=   197
0   1   <=   306
```

VITA

Chin-Chih Chang

Candidate for the Degree of

Doctor of Philosophy

Thesis: IMPLEMENTATION AND APPLICATIONS OF QUERY INTERFACES TO CONSTRAINT DATABASES IN A DISTRIBUTED COMPUTING ENVIRONMENT

Major Field: Computer Science

Biographical:

Personal Data: Born in Hsinchu City, Taiwan, Republic of China, December 5, 1963, the son of Hai-Ti Chang and Show Chen.

Education: Graduated from Hsinchu High School, Hsinchu City, Taiwan in June 1982; received Bachelor of Engineering degree in Mechanical Engineering from Tamkang University, Tamsui, Taipei County, Taiwan in June 1986; received Master of Science degree in Engineering Science from National Cheng Kung University, Tainan City, Taiwan in June 1990. Completed the requirements for the Doctor of Philosophy with a major in Computer Science at Oklahoma State University in December, 2000.

Professional Experience: Research Assistant, Spatial and Environmental Information Clearing House, Oklahoma State University, from June 1994 to December 2000. Teaching Assistant, Department of Computer Science, Oklahoma State University, from January 1994 to May 1994.