Web Based Query Optimization Simulator

Edwin Richard Waite

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

Part of the Software Engineering Commons

Recommended Citation
https://scholarworks.lib.csusb.edu/etd-project/2519

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
WEB BASED QUERY OPTIMIZATION SIMULATOR

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Edwin Richard Waite

September 2004
WEB BASED QUERY OPTIMIZATION SIMULATOR

A Project,

Presented to the

Faculty of

California State University,

San Bernardino

by

Edwin Richard Waite

September 2004

Approved by:

Josephine Mendoza, Chair, Computer Science 

David Turner 

Ernesto Gomez
ABSTRACT

The Web Based Query Optimization Simulator (WBQOS) is a software tool designed to enhance understanding of query optimization within a Relational Database Management System (RDBMS). WBQOS allows the user to visualize and participate in query optimization, which enhances the learning process. While some portions of an RDBMS needed to be implemented in WBQOS, it is not an optimization module to be used within an RDBMS, rather it simulates the optimization process and visually represents it to the user through query trees.
ACKNOWLEDGMENTS

I express my thanks and appreciation to my graduate committee for their valuable assistance. Dr. Mendoza's advice during the design and implementation phases of the project provided much needed direction. Dr. Turner and Dr. Gomez offered valuable feedback and suggestions which helped immensely in creating the user interface for WBQOS.

The support of the National Science Foundation under award 9810708 is gratefully acknowledged.
DEDICATION

I dedicate this project to my beautiful wife Holly and three wonderful children Amanda, Jacob and Joshua. I am grateful for their love, support and sacrifices throughout my graduate studies.
# TABLE OF CONTENTS

**ABSTRACT** ............................................................... iii

**ACKNOWLEDGMENTS** .............................................. iv

**LIST OF FIGURES** ................................................... vii

**CHAPTER ONE: SOFTWARE REQUIREMENTS SPECIFICATION**

1.1 Introduction ....................................................... 1

1.2 Purpose of the Project .......................................... 1

1.3 Context of the Problem .......................................... 2

1.4 Significance of the Project ..................................... 2

1.5 Assumptions ....................................................... 2

1.6 Limitations ....................................................... 3

1.7 Definition of Terms ............................................. 3

1.8 Definition of Notations ......................................... 5

**CHAPTER TWO: COMPONENTS OF QUERY OPTIMIZATION**

2.1 Introduction ....................................................... 7

2.2 Relations and Relational Algebra ............................... 9

2.3 Query Trees ..................................................... 10

2.4 Heuristic Optimization Algorithm ............................ 12

**CHAPTER THREE: SOFTWARE DESIGN**

3.1 Introduction ....................................................... 18

3.2 Architecture ..................................................... 18

3.2.1 Client Tier ................................................... 19

3.2.2 Middle Tier .................................................. 19
3.2.3 Data Tier ................................. 19
3.3 Detailed Design ......................... 20
  3.3.1 Language Parser .................... 20
  3.3.2 Relational Algebra ................. 22
  3.3.3 Heuristic Optimizer ............... 24
  3.3.4 Query User Interface .............. 25
3.4 Use Case .................................. 27
  3.4.1 Use Case Query Optimization ..... 27

CHAPTER FOUR: MAINTENANCE
  4.1 Configuration ............................ 29
  4.2 Database Installation ................. 29
  4.3 Software Installation .................. 30

CHAPTER FIVE: USERS MANUAL
  5.1 Introduction ............................. 33
  5.2 Structured Query Language Interface .. 33
  5.3 Heuristic Optimization Interface ..... 36

CHAPTER SIX: CONCLUSIONS
  6.1 Future Direction ........................ 38
  6.2 Conclusion ................................ 39

APPENDIX A: PARSER GRAMMAR RULES .......... 40
APPENDIX B: INSTALLATION NOTES .......... 42
APPENDIX C: SOURCE CODE .................. 45
REFERENCES .................................. 172

vi
LIST OF FIGURES

Figure 1. Query Phases ........................................ 8
Figure 2. Canonical Tree ..................................... 10
Figure 3. Execution Order .................................... 11
Figure 4. Push Down Project ................................. 16
Figure 5. Architecture ........................................... 18
Figure 6. Communication Process ......................... 20
Figure 7. Parser Class Diagram ............................ 22
Figure 8. Relational Algebra Class Diagram ................ 23
Figure 9. Optimizer Class Diagram ......................... 24
Figure 10. Query Interface Class Diagram .................. 26
Figure 11. Query User Interface ............................. 32
Figure 12. Show Tables ....................................... 34
Figure 13. Show Attributes ................................... 35
Figure 14. Heuristic Optimization ........................... 37
CHAPTER ONE
SOFTWARE REQUIREMENTS SPECIFICATION

1.1 Introduction

In this project I present a software system called WBQOS (Web Based Query Optimization Simulator) as an educational tool for students and instructors involved in the field of query optimization.

1.2 Purpose of the Project

WBQOS was developed as a software tool to enhance instruction and learning in the field of query optimization within the context of relational database systems. It was designed to graphically and interactively represent the query optimization process, thus allowing students to visualize a complicated process and enhance the learning process. There are many algorithms used to optimize a query executed against a database, but not all of these processes could be represented in a single tool. WBQOS represents a heuristic algorithm to query optimization. It uses query trees to visually represent each step in the optimization process.
1.3 Context of the Problem

There are limited resources a student has available when studying query optimization. Typically the resources are comprised of an instructor, textbooks and journal papers. WBQOS is an additional resource, providing students with a software tool that they can interact with.

1.4 Significance of the Project

For those interested in working with database systems it is important to understand query optimization. Understanding how the optimization process works will enable a person to write more efficient queries. Another significant aspect of the project is the ability to use this tool inside the classroom, giving instructors, an additional teaching resource.

1.5 Assumptions

1. Users of the WBQOS will know Structured Query Language (SQL) and be engaged in learning about the query optimization process.
2. Users will have access to the Internet.
3. Users will access the application with a Java-enabled browser.
1.6 Limitations

During the development of the project, a number of limitations were noted. These limitations are presented here.

1. The SQL language recognized by the WBQOS SQL parser is a subset of ANSI SQL. In general the parser will accept a single SELECT-FROM-WHERE statement, with the SELECT being a comma-delimited list of attributes (See Appendix A for the rules of grammar used by the parser).

2. The SQL parser will identify invalid tables and attributes in a query, but it will not provide specifics on syntactical errors in the query. It will simply notify the user that the query is not syntactically correct.

1.7 Definition of Terms

The following terms are defined as they apply to the project.

A. WBQOS – Web Based Query Optimization Simulator.

B. Tuple – Represents a row in a relation.

C. Query Optimization – The process of determining the optimal execution plan for a given database query.
D. Optimal Query - The query that has the least amount of block transfers from secondary storage (i.e. hard disks) to main memory.

E. Query Tree - A tree data structure used to internally represent a query.

F. DBMS - Database Management System.

G. RDBMS - Relational Database Management System.

H. Optimizer - The query optimization module of an RDBMS.

I. SQL - Structured Query Language. A high-level query language used to access data from a database.

J. Heuristic Rule - A rule that works well for most situations, but is not guaranteed to work in every situation.

K. Applet - A Java application that runs in a browser.

L. JAR File - A Java archive file.

M. QUI - Query User Interface. The graphical interface of WBQOS where users can enter database queries.

N. SPJ - SELECT-PROJECT-JOIN query. A type of SQL query that is limited to a single SELECT-FROM-WHERE clause.
1.8 Definition of Notations

The following are notations that occur in the text and are defined here:

1. A relation can be defined with the following symbols:

\[ R(R_{A_1}, R_{A_2}, ..., R_{A_n}:\text{NUMERIC}) \]

The first \( n \) characters before the open parenthesis represents the name of the relation. The sequence of \( R_{A_n} \) symbols represents the attributes of the relation \( R \). There is an optional byte size that can be appended to the attribute in this manner \( R_{A_1}(30) \). The numeric after the colon represents the number of tuples (rows) in the relation.

2. An intermediate relation is a relation that results from performing a relational algebra operation and can be represented with the following symbols:

\[ IR_{R,T} (R_{A_1}, R_{A_2}, ..., R_{A_n}, T_{A_1}T_{A_2}, ..., T_{A_n}, ...:\text{NUMERIC}) \]

The subscripts after IR represents the relations involved. Each relation should have a unique symbol and there can be \( n \) number of relations. The sequence of \( R_{A_n}, T_{A_n}, ... \) are the attributes of the respective relations. As before the NUMERIC represents the number of tuples in the relation. There is an optional
superscript to denote a second, third, or n-ary instance of the intermediate relation, denoted by:

$$IR^{i,2,\ldots,n}$$

A typical example of an intermediate relation would be the resulting relation of joining two tables.
CHAPTER TWO
COMPONENTS OF QUERY OPTIMIZATION

2.1 Introduction

Before a query can be optimized it must pass through several stages. A query is first analyzed by a query parser, which checks the query for both syntactic and semantic correctness. The parser determines whether the query is syntactically correct by parsing each line of the query and verifying that it conforms to the rules of grammar defined for query language, in this case Structured Query Language (SQL). If the query is found to be syntactically incorrect an error is returned to the user, otherwise the parser then validates it semantically. This is accomplished by verifying that each attribute and relation, defined in the query, exists in the database where the query is executed.

Next the query is translated into relational algebra operations and stored in some internal data structure. Often the internal structure is some tree type structure. If this is the case each internal node in the tree represents an operation and each leaf node is a relation. Once the query is represented internally in relational
algebra operations, optimization algorithms can be applied to the query to determine an optimal ordering of the operations and the most efficient data access algorithms to use (See Figure 1).

Figure 1. Query Phases

There is an important distinction about the type of queries in WBQOS project. During the design phase of this project I decided to narrow the type of queries accepted by WBQOS to single block Select-Project-Join queries, better known as SPJ's. The term single block is taken to mean a query with a single SELECT-FROM-WHERE statement. Typically a query optimizer takes multi-block queries (nested queries) and separates them into single block queries, then applies the optimization process to each single block query and
finally synthesizes the individual results into a single execution plan. Consequently, multi-block query optimization could also be considered as many instances of single block query optimization. Therefore the added complexity of nested queries did not seem necessary for this project.

2.2 Relations and Relational Algebra

In the now famous paper "A Relational Model of Data for Large Shared Data Banks" E.F. Codd introduced the relational database model[1]. "The model uses the concept of a mathematical relation—which looks somewhat like a table of values—as its basic building block. . ."[2]. To manipulate the data within relation(s), a set of relational algebra operations were developed. The following is a list of the relational algebra operations applicable to this project:

PROJECT - denoted by the symbol Π<attribute list>
SELECT - denoted by the symbol σ<select condition>
CARTESIAN PRODUCT - denoted by the symbol X
JOIN - denoted by the symbol |X|<join condition>
2.3 Query Trees

WBQOS uses a binary tree (or query tree) as the internal data structure for the SQL query. Each internal node of the tree is a relational algebra operation and each leaf node is a relation. The query is first represented as an unoptimized or "canonical" tree as stated in [2] (See Figure 2).

\[
\Pi<\text{fname, lname}>
\sigma<\text{ssn} = \text{essn} \text{ and pno} = \text{pnumber} \text{ and pname} = \text{"ProductX"}>
\]

\[
\begin{array}{c}
X \\
\text{Project} \\
\text{Employee} \quad \text{Works_On}
\end{array}
\]

Figure 2. Canonical Tree

Each operation in the tree also represents an intermediate relation resulting from the operation. Each
intermediate relation becomes the input for the next operation. Operation execution order for a query tree is from left to right and bottom to top (See Figure 3).

(6th) \[ \Pi\text{<fname, lname>} \]

(5th) \[ \sigma\text{<pno = pnumber>} \]

(4th) \[ X \]

(3rd) \[ \sigma\text{<pname = 'ProjectX'>} \]

(2nd) \[ \sigma\text{<ssn = essn>} \]

(1st) \[ X \]

---

Figure 3. Execution Order
2.4 Heuristic Optimization Algorithm

The concept behind heuristic optimization is to replace resource intensive operations with more efficient ones, to commute the order of operations where advantageous and to add operations that will reduce intermediate results.

The heuristic optimizer in WBQOS is patterned after the algorithm found in [2] and [3]. There are four components to the optimizer, they are: Push Down SELECT (PDS), Commute Relations (CR), Create JOINS (CJ), and Push Down PROJECT (PDP). The optimizer uses the transformation rules for relational algebra defined in [2], to ensure the resulting query trees are equivalent to the original.

The goal of the PDS component is to take a conjunctive SELECT operation and decompose it into individual SELECT operations, which are then pushed down the query tree as far as the transformation rules allow. This reduces the number of tuples in each relation a SELECT operation is executed against. The result is illustrated in the following example:

CARTESAIN PRODUCT operation without a SELECT operation.

Relation T is defined as: \( T(T_{A1},T_{A2},T_{A3},T_{A4},T_{A5},T_{A6}:5000) \)

Relation S is defined as: \( S(S_{A1},S_{A2},S_{A3}:2200) \)
\[ T \times S = IR_{TS} \text{ defined as:} \]

\[ IR_{TS}(T_{A1}, T_{A2}, T_{A3}, T_{A4}, T_{A5}, S_{A1}, S_{A2}, S_{A3}:11000000) \]

Apply SELECT operations before CARTESIAN PRODUCT.

\[ \sigma<T_{A1} = c_1>(T) = IR_T(T_{A1}, T_{A2}, T_{A3}, T_{A4}, T_{A5}, T_{A6}:100) \]

When the selectivity of condition \( T_{A1} = \) some constant \( c_1 \) is 100.

\[ \sigma<S_{A3} = c_2>(S) = IR_S(S_{A1}, S_{A2}, S_{A3}:500) \]

When the selectivity of condition \( S_{A3} = \) some constant \( c_2 \) is 500.

\[ IR_T \times IR_S = IR_{TS}(T_{A1}, T_{A2}, T_{A3}, T_{A4}, T_{A5}, T_{A6}, S_{A1}, S_{A2}, S_{A3}:50000) \]

As can be seen from this example the number of tuples for the CARTESIAN PRODUCT before applying SELECT operations was 11,000,000 million, whereas applying the SELECT operations before executing the CARTESIAN PRODUCT operation produced a relation with 50,000 thousand tuples, a significant difference.

The CR component is to “position the leaf node relations with the most restrictive SELECT operations so they are executed first in the query tree. . .”[2]. Consequently the size of the JOIN operations is reduced. Since the JOIN is one of the most resource intensive operations this can dramatically speed up query execution.
The goal of the CJ component is to combine any SELECT/CARTESIAN PRODUCT sequence of operations into a JOIN operation, but only if the condition c of the SELECT operation corresponds to a join condition. In Figure 3 there is the SELECT/CARTESIAN PRODUCT sequence of operations in which the Employee and Works_on relations are joined. I will illustrate that sequence of operations and the benefit of the CJ component in the following example:

Relation E is defined as:

\[ E(E_{a1}, E_{a2}, E_{a3}, E_{a4}, E_{a5}, E_{a6}, E_{a7}, E_{a8}, E_{a9}, E_{a10}: 10000) \]

Relation W is defined as: \[ W(W_{A1}, W_{A2}, W_{A3}, W_{A4}: 6120) \]

\[ E \times W = IR_{EW} \] defined as:

\[ IR_{EW}(E_{a1}, E_{a2}, E_{a3}, E_{a4}, E_{a5}, E_{a6}, E_{a7}, E_{a8}, E_{a9}, E_{a10}, W_{A1}, W_{A2}, W_{A3}, W_{A4}: 61200000) \]

\[ \sigma <E_{a1} = W_{A1}> (IR_{EW}) = \]

\[ IR^2_{EW}(E_{a1}, E_{a2}, E_{a3}, E_{a4}, E_{a5}, E_{a6}, E_{a7}, E_{a8}, E_{a9}, E_{a10}, W_{A1}, W_{A2}, W_{A3}, W_{A4}; 1350000) \]

When the selectivity of condition \( E_{a1} = W_{A1} \) is 1,350,000. Now combine the SELECT and CARTESIAN PRODUCT operations into a single JOIN operation.

\[ E \mid X <E_{a1} = W_{A1}> W = \]
IR₃ₓₓ₁(Eₐ₁, Eₐ₂, Eₐ₃, Eₐ₄, Eₐ₅, Eₐ₆, Eₐ₇, Eₐ₈, Eₐ₉, Eₐ₁₀, W₁, W₂, W₃, W₄:
1350000)

By combining the SELECT and CARTESIAN PRODUCT operation into a single JOIN operation the very large (61,200,000 tuples) intermediate relation caused by the CARTESIAN PRODUCT operation was avoided.

The purpose of the PDP component is to reduce the number of attributes in intermediate relations and thus reduce the total byte size of each tuple. The basic algorithm for this component is to use a post order tree traversal and at each internal node create a new PROJECT operation, in accordance with transformation rules. The operation should contain the attributes needed in the final PROJECT operation and any attributes needed in subsequent operations. The query tree in Figure 4 demonstrates the algorithm.
For example if I define the Employee relation as:

\[ E(E_{A1}(30), E_{A2}(1), E_{A3}(30), E_{A4}(9), E_{A5}(20), E_{A6}(100), E_{A7}(1), E_{A8}(10), E_{A9}(9),
\]
\[ E_{A10}(1):10000), \] where \( E_{A1} = \text{fname}, E_{A3} = \text{lname} \) and \( E_{A4} = \text{ssn} \).
If the PROJECT operation is not applied to the Employee relation, then the total number of input bytes ($R_{bs}$) for the subsequent JOIN operation is:

Record Size $R_s = 210$

Tuple Count $T_c = 10000$

Relation Byte Size $R_{bs} = R_s \times T_c = 2,100,000$

Where as if the project operation is applied first:

$\Pi_{E_{A1}, E_{A3}, E_{A4}}(E) = IR_{E_{A1(30)}, E_{A3(30)}, E_{A4(9)}:10000}$

$R_s = 69$

$T_c = 10000$

$R_{bs} = 690,000$

Thus without the PROJECT operation there are $1,410,000$ bytes of superfluous data piped into the subsequent JOIN operation. Therefore by creating additional PROJECT operations with only those attributes needed in subsequent operations, the superfluous data is eliminated and the query execution time can be decreased.

As seen from the examples the heuristic algorithm can significantly improve query execution, however in general the heuristic algorithm is not sufficient by itself. It requires another optimization algorithm to calculate cost estimates for implementing different execution plans.
CHAPTER THREE
SOFTWARE DESIGN

3.1 Introduction

A component based design approach was taken with WBQOS. This allowed components to be developed and tested individually. It also allows WBQOS to be extended in the future with additional components, such as a cost-based optimizer. The entire project was written in the Java programming language for its rich web development features.

3.2 Architecture
WBQOS utilizes the standard three-tier architecture of web applications (See Figure 5).

![Figure 5. Architecture](image-url)
3.2.1 Client Tier

The client tier consists of the user's Java-enabled web browser, which makes a request from the middle tier for the Java applet. The applet is then downloaded to the browser and executed. The user can select a database, enter a query and get both the data results and the optimization results returned.

3.2.2 Middle Tier

The middle tier consists of the web server, which serves up requests from the client. There is no server side (middle tier) execution of WBQOS code.

3.2.3 Data Tier

WBQOS uses MySQL as the back-end database, which can contain multiple databases for the application to be run against. There are two categories of data that WBQOS requests from a database. One is the result of the query and the other is a variety of statistical information about relations, attributes, etc. the optimizer needs to simulate the optimization process. (See Figure 6.)
3.3 Detailed Design

3.3.1 Language Parser

The parser design utilizes a class library developed by Steven John Metsker[4]. This library provides fundamental elements (building blocks) for the parsing process, such as sequences, alternations, and repetitions. The grammar that defines the parser is given here:

\[
\text{Query} = \text{"select" selectTerms "from" tableNames optional Where;}
\]
selectTerms = commaList(selectTerm);
selectTerm= expression;
tableNames= commaList(tableName);
tableName= Word;
columnNames = commaList(columnName);
columnName= Word;
comparisons= commaList(comparison);
commaList(p) = p (',', p)*;
optional Where= empty | "where" comparisons;
comparison= arg operator arg;
arg= expression | quoted string;
expression = term ('+' term | '-' term)*;
term = factor ('*' factor | '/' factor)*;
factor = '(' expression ')' | Number | variable;
variable= Word;
operator = "<" | "(" | "=<" | "><" | "!=";

Once the grammar and the building blocks were in place, I was able to develop the class diagram shown in Figure 7. An additional function built into the parser is the Assembler. As the parser recognizes tokens from the query it uses the Assembler to build (or assemble) the tokens into equivalent relational algebra expressions. A separate
module called "Semantic Checker" is used to validate the attributes and relations in the query. (See Figure 7.)

3.3.2 Relational Algebra

This module is used directly by the Assembler to transform the query into a relational algebra expression. The object the Assembler builds is the RAQuery shown in Figure 8. RAQuery uses instances of RAProject, RASelect, RACartesian, RAJoin, and RARElation to form the relational algebra expression. These operations are then loaded into a query tree and passed to the optimizer. (See Figure 8.)
Figure 8. Relational Algebra Class Diagram
3.3.3 Heuristic Optimizer

The Optimizer class is the driving class of the optimization module. It initiates and manages the calls to the HOptimizer class (Heuristic Optimizer) and it calculates the results of each operation in a query tree and stores them for later use (See Figure 9).

---

**Figure 9. Optimizer Class Diagram**
3.3.4 Query User Interface

The QUI went through many iterations during the life cycle of this project. It began as a stand-alone Java application and finished as a Java Applet runnable from the web. The QUI_Query implements the interface for submitting a SQL query, the QUI_HOpt class implements the heuristic optimization interface and the QUI_Pane class wraps the QUI_Query class and the QUI_HOpt class inside a JTabbedPane class (See Figure 10). There are two inner classes, GraphicPoints and DrawingPane, inside QUI_HOpt, which draw the query trees onto the component. The GraphicPoints class iterates through each query tree and calculates the geometric points for each node in the tree. The DrawingPane class uses the geometric points to draw out each node and the lines connecting them. (See Figure 10.)
Figure 10. Query Interface Class Diagram
3.4 Use Case

There is a Use Case Text and a Use Case Diagram, I have chosen to use "Use Case Text" because I feel they are clearer and more concise. The following is the Use Case for query optimization within WBQOS and comprises several scenarios.

3.4.1 Use Case Query Optimization

Scenario: Enter Query
1. User enters URL into browser.
2. User chooses a database to query.
3. User chooses optional line and node colors.
4. User types in an SQL query.
5. User clicks the submit button or if user wants to clear the query, user clicks the clear button and repeats step 4.

Alternative: Parser Error

At step 5 the WBQOS parser detects an error in query and error is returned to user. User corrects query and repeats step 5.

Scenario: Heuristic Optimizer
1. User selects Heuristic Optimizer tab after successful completion of scenario Enter Query.
2. User clicks on a node in a query tree.

Scenario: Native SQL

1. User completes steps 1 through 4 in scenario Enter Query.
2. User selects native SQL check box.
3. User enters SQL query.
4. User clicks the submit button or if user wants to clear the query, user clicks the clear button and repeats step 3.
5. Optimization module is skipped, a direct query of the database is done and results are returned to user.

Alternative: Parser Error

At step 4 the MySQL parser detects an error in query and error is return to user. User corrects query and repeats step 3.
CHAPTER FOUR

MAINTENANCE

4.1 Configuration

There is a restriction to installing WBQOS that needs to be explained. Under the default java.security model an applet can only communicate with the server from which it was downloaded. Since WBQOS needs to communicate with the database server, this requires the web server and the database server to be on the same physical machine.

4.2 Database Installation

WBQOS uses MySQL as the back-end database and a default database called "company" taken from [2]. Although "company" is the default database WBQOS has the flexibility to operate against any database. It is important to distinguish the ability to run against any database from the ability to run against any database management system. WBQOS can only run against the MySQL database management system, but the user has the option of creating multiple databases within MySQL and selecting any of those databases to simulate query optimization. In addition because WBQOS is written in Java it has the flexibility of running on Windows, Linux or any other operating system Java runs on.
To install MySQL simply download the appropriate package (depending on the operating system) and run the installer. On Windows simply click on the setup.exe and on Linux enter the following command at a shell:

rpm -i<name of package>.

Once the database is installed restore the backup copy of the company database found on the WBQOS install CD. For instructions on restoring a database see Appendix B.

4.3 Software Installation

Because WBQOS runs as a Java Applet there needs to be a functioning web server on the machine were WBQOS is installed (for instructions on configuring a web server see Appendix B). Once a web server is up and running there are two ways to install WBQOS, as a set of directories or as a single jar file. Navigate to the <install directory> on the web server. The <install directory> must be a sub directory of the web server’s html directory (See Appendix B). For this project I placed it in a directory name “queryopt”. To install as a set of directories, copy the following directories from the install CD into the <install directory>: com, mylib, org, sjm, and wbqos. Then copy the file QUI_Pane.html to the root of the <install directory>.
Using a text editor open the QUI_Pane.html file. Find the html parameter tag that identifies the server, it looks like the following:

```html
<PARAM NAME="server" VALUE="192.168.254.4">
```
Change the IP address to match the IP address of the web server and save the file.

Installing WBQOS with a jar file is similar to the previous installation. Navigate to the <install directory> and copy from the install CD the wbqos.jar file into the <install directory>. Copy the QUI_Pane.html file into the same directory as wbqos.jar file. Using a text editor open the QUI_Pane.html file and make the change for the IP address as stated above, then locate the following html line:

```html
<APPLET code="wbqos/gui/QUI_Pane.class"
width="100%" height="100%">
```
After the code attribute insert the following additional attribute: ARCHIVE="wbqos.jar". Close and save the file.

Installation is complete. To verify installation, open up a web browser and type the URL for WBQOS, a screen like Figure 11 should appear.
The user-friendly installation of WBQOS enables a user to have it up and running in a short period of time. Also, the ability to comprise the source files into one JAR file increases execution speed.

Figure 11. Query User Interface
CHAPTER FIVE
USERS MANUAL

5.1 Introduction

WBQOS is user-friendly and quite simple to use. There are two main interfaces, SQL and heuristic optimization. To begin using WBQOS simply open up a Java-enabled web browser and type the URL, which identifies the location of WBQOS.

5.2 Structured Query Language Interface

The SQL interface is the initial screen and provides the data for subsequent screens. This interface provides a user with the ability to submit queries against a database, view the results and set options for the heuristic optimization interface. Before submitting a query for optimization results a user may want to view the schema of the database. Queries for meta-data within the database are not suitable for query optimization, so this interface provides a checkbox labeled "Native SQL". When this is checked the SQL statement is sent directly to the database and the optimization process is bypassed. For example, a user could select the appropriate database from the drop-down list labeled "Databases", check the "Native SQL" checkbox, enter "show tables" into the "Enter Query" textbox
and then click submit. (See Figure 12.) To see the attributes of a given table follow the same steps, but the query would be “describe <table name>”. (See Figure 13.)

Figure 12. Show Tables
If there is an error found in the SQL statement the error will be returned inside the results area. The "Clear" button can be used to erase all information in the "Enter Query" textbox and in the results area (it will also clear all optimization results). The user can also choose the type of font color assigned to the nodes and lines on the query trees within the "Heuristic Optimization" screen.
5.3 Heuristic Optimization Interface

To view the optimization process, follow the instructions in Section 5.2 then click on the "Heuristic Optimization" tab. This interface is divided into two sections. The top section displays statistical information when an operation in a query tree is clicked, such as record count, record size, column count, etc. This information is very useful in making comparisons between operations of different query trees. (See Figure 14.) The bottom section displays the query trees that represent the heuristic optimization path for the given query. If the labels of two sibling nodes intersect each other the labels are truncated. To view the complete name of a truncated label simply hover the mouse over the node. Each node may be clicked to display the statistical information in the top section of the component.
Figure 14. Heuristic Optimization

This interface enables users to compare and contrast equivalent nodes from different trees to determine the effects of the optimization process. It is from this analysis that the user will understand the optimization algorithm.
CHAPTER SIX

CONCLUSIONS

6.1 Future Direction

There are two additional modules that can be focused on in the future, a cost-based optimization algorithm and a Query Code Generator.

A cost-based algorithm would pick up where WBQOS leaves off. It would take as input the optimized query tree from the heuristic algorithm and produce a new set of query trees showing the cost-based optimization path for a query. This would be a significant addition since in most commercial RDBMSs a heuristic optimization algorithm is used in conjunction with a cost-based optimization algorithm.

Implementing a Query Code Generator would show how the results of the optimization process are turned into executable code for retrieving the results of the query.

A foundation has been laid with the development of WBQOS, which can be supplemented with these additional modules to provide a comprehensive tool representing the entire query optimization process.
6.2 Conclusion

The WBQOS project successfully implemented an educational software tool, which simulates the complex process of query optimization in a way students can understand. Its user interface is easy to use and provides a way to measure the effects of each heuristic optimization phase.

The software is simple to install and accessible by any Java-enabled web browser, which allows instructors to use it in the classroom, in the lab or make it available to the public domain.

Since WBQOS was developed using a component-based software methodology it can be extended and can be enhanced with additional features and modules without modifying existing code.

This project has provided me with a greater understanding of query optimization, software development and relational database management systems. I hope WBQOS will be of benefit to students as they study query optimization.
APPENDIX A

PARSER GRAMMAR RULES
Query = "select" selectTerms "from" tableNames
    optional Where
selectTerms = commaList(selectTerm);
selectTerm= expression;
tableNames= commaList(tableName);
tableName= Word;
columnNames = commaList(columnName);
columnName= Word;
comparisons= commaList(comparison);
commaList(p) = p (',', p)*;
optional Where= empty | "where" comparisons;
comparison= arg operator arg;
arg= expression | quoted string;
expression = term ("+" term | "-" term)*;
term = factor ("*" factor | "/" factor)*;
factor = "(" expression ")" | Number | variable;
variable= Word;
operator = "<" | ">" | "<=" | "=" | "!"
APPENDIX B

INSTALLATION NOTES
**RESTORE DATABASE**

Since MySQL databases are implemented as files, it is very simple to restore a database. Navigate to the data directory (mysql\data for windows and var/lib/mysql on linux) and create a directory named <database name>. For our example the name would be “company”. Copy the files from the “company” directory on the install CD to the newly created directory on the database server. The database is now restored and can be used.

**WBQOS INSTALL DIRECTORY**

On every web server is the root directory where the web server starts looking for requested files. On Windows the web server is Internet Information Server and the root directory is <install drive>:\Inetpub\wwwroot. On RedHat Linux (8.0 and on) the web server is Apache and the root directory is var/www/html. The install directory for WBQOS must be a sub directory of this directory. For example for this project WBQOS was installed into var/www/html/queryopt.

**CONFIGURING WEB SERVER**

The section covers web server configuration for Windows and RedHat Linux operating systems, for other operating systems please consult the documentation for that product.

On a Windows server operating system verify that Internet Information Server (IIS) is installed. To do this, click on "Add/Remove Software" in Control Panel, then click on "Add/Remove Windows Components". Scroll down to see if "Internet Information Services" checkbox is checked. If it is, then IIS is configured and you can run WBQOS. If it is not checked, check it and click "Next" button. Follow instructions and reboot system. Now you are ready to run WBQOS.

On Linux verify the Apache web server is installed by issuing the following command at a shell:

```
rpm -q <name of apache package>
```

If the package is not installed then acquire the package by downloading it from www.redhat.com or copy it from the
install CD’s for RedHat Linux (if the package is installed skip to next section). Once you have the package issue the following command from a shell:
   rpm -i <package name>

Once the package is installed you can start the web server by issuing the following command:
   /etc/rc.d/init.d/httpd start

(Note: the path to the httpd executable may differ depending on type on Linux operating system)

Once the web server is started you can now run WBQOS.
APPENDIX C

SOURCE CODE
Parser Module

/*
 *Programmer: Edwin Waite
 *Date: 9/05/02
 *Title: SQL Parser
 *Purpose: To create a Parser that recognizes the
 Structured Query Language (SQL)
 *used in many relational database systems. This class
 uses the packages in
 *sjm.*, which provide the basic tools for building Java
 Parsers.
 */

package wbqos.parsers;

import java.util.*;
import wbqos.relAlg.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

public class SQLParser {

    //private whereAssembler;
    /** Creates a new instance of SQLQuery */
    public SQLParser() {
        //whereAssembler=new whereAssembler();
    }

    /*The query method establishes the framework for
     *a syntactically correct SQL statement. Each method
     *in this class corresponds to a particular
    subParser(query())
     *being the Parent Parser) for an SQL statement. Other
    methods (i.e. subParsers)
     *are called from the query() method.
     *
     *The discard() method is used throughout the program--
    it is used to
    *keep a Terminal Parser from being pushed onto the
 assemblies stack.
    *By default when a parser matches a terminal it pushes
 it onto the assemblies stack.
     */
    public Parser query() {

    }
Sequence q=new Sequence();
q.add(new CaselessLiteral("select").discard());
q.add(commaList(selectTerm()));
q.add(new CaselessLiteral("from").discard());
q.add(fromStat());
q.add(optWhere());
q.add(optGroupBy());
q.add(optHaving());
q.add(new Symbol(';').discard());
return q;
}

/*commaList() is a subParser that recognizes a comma
list of
*the parser that is passed into it. For example
commaList(tableName())
*would recognize a comma list of table names--e.g
employee, dept, projects
*/

public Parser commaList(Parser p) {
    Sequence c=new Sequence();
c.add(p);
    Sequence n=new Sequence();
n.add(new Symbol(',').discard());
n.add(p);
c.add(new Repetition(n));
return c;
}

/*selectTerm() is a subParser used as the parent parser
for
*SQL statements involving mathematical expressions and
group functions.
*For example:
*select Fname, Lname, (salary * 10)
*from employee
*This query would return the first and last name of
each employee
*as well as the salary of each employee multiplied by
10
*This statement contains the expression (salary * 10)
The heirarchical chain of methods that represent an
expression
*are as follows:
selectTerm() --> expressions() --> term() --- > factor()

*The next three methods describe the rest of the selectTerm() hierarchy
*

//a select term can be an expression or group function
public Parser selectTerm() {
    Alternation select=new Alternation();
    select.add(groupFunction());
    select.add(expressions());
    return select;
}

/*An expression can be a repition of term (+ or -) term
*/
public Parser expressions() {

    Sequence e=new Sequence();
    e.add(term());
    Alternation a=new Alternation();
    Sequence addition=new Sequence();
    addition.add(new Symbol('+').discard());
    addition.add(term());
    Sequence minus=new Sequence();
    minus.add(new Symbol('-').discard());
    minus.add(term());
    a.add(addition);
    a.add(minus);
    e.add(new Repetition(a));
    return e;
}

public Parser term() {
    Sequence t=new Sequence();
    t.add(factor());
    Alternation f=new Alternation();
    Sequence mult=new Sequence();
    mult.add(new Symbol('*').discard());
    mult.add(factor());
    Sequence divide=new Sequence();
    divide.add(new Symbol('/').discard());
    divide.add(factor());
    f.add(mult);
    f.add(divide);
t.add(new Repetition(f));
return t;
}

public Parser factor() {
    Alternation f=new Alternation();
    f.add(new Num().discard());
    f.add(columnNames());
    return f;
}

/*A Parser recognizing a sequence of column names (or attributes) within
the select clause in sql. The attributes in a select clause
are the attributes that are associated with a relational
*algebra project clause, thus the assembler is a projectAssembler.
*/
Parser columnNames() {
    Sequence s=new Sequence();
    s.add(columnName());
    s.setAssembler(new projectAssembler());
    return s;
}

//A terminal parser for recognizing a single column (or attribute) name
Parser columnName() {
    return new Word();
}

//variable() is used to match words with the Terminal Parser Word()
public Parser variable() {
    return new Word();
}

/*A parser recognizing a sequence of table (or relation) names within
*a from clause in an sql statement.
*/
public Parser fromStat() {
    Sequence c=new Sequence();
    c.add(commaList(tableName()));
    return c;
public Parser tableName()
{
    return new Word().setAssembler(new relationAssembler());
}

/*The optWhere() method is a subParser recognizing the
optional where clause
* in an SQL query. For example
* select Fname, Lname
* from employee
* where Fname="Frank"
*
* This method also allows conjuctive and disjunctive
where statements, i.e.
* select Fname, Lname
* from employee
* where Fname="Frank"
* AND Lname="Bellows"
*/

public Parser optWhere()
{
    Alternation w=new Alternation();
    Sequence s=new Sequence();
    s.add(new CaselessLiteral("where").discard());
    s.add(comparisons());
    w.add(new Empty());
    //this is for conjunctive and disjunctive where statements
    Sequence a= new Sequence();
    a.add(new CaselessLiteral("and").setAssembler(new
    whereAssembler()));
    a.add(comparisons());
    Sequence o= new Sequence();
    o.add(new CaselessLiteral("or").setAssembler(new
    whereAssembler()));
    o.add(comparisons());
    Alternation b= new Alternation();
    b.add(a);
    b.add(o);
Repetition r = new Repetition(b);
s.add(r); // add this repetition to the where sequence
w.add(s);
return w;
}
// where expressions
public Parser whereExpressions() {
    Sequence e = new Sequence();
    e.add(whereTerm());
    Alternation a = new Alternation();
    Sequence addition = new Sequence();
    addition.add(new Symbol('+').discard());
    addition.add(whereTerm());
    Sequence minus = new Sequence();
    minus.add(new Symbol('-').discard());
    minus.add(whereTerm());
    a.add(addition);
    a.add(minus);
    e.add(new Repetition(a));
    return e;
}
// where term
public Parser whereTerm() {
    Sequence t = new Sequence();
    t.add(whereFactor());
    Alternation f = new Alternation();
    Sequence mult = new Sequence();
    mult.add(new Symbol('*').discard());
    mult.add(whereFactor());
    Sequence divide = new Sequence();
    divide.add(new Symbol('/').discard());
    divide.add(whereFactor());
    f.add(mult);
    f.add(divide);
    t.add(new Repetition(f));
    return t;
}
// where factor
public Parser whereFactor() {
    Alternation f = new Alternation();
    f.add(new Num());
    f.add(whereColumnNames());
    f.setAssembler(new whereFactorAssembler());
    return f;
}
//equivalent to columnNames, but needed for where assembler
public Parser whereColumnNames() {
    Sequence s=new Sequence();
    s.add(whereColumnName());
    s.setAssembler(new whereColumnNameAssembler());
    return s;
}
//equivalent to columnName, but needed for where assembler
public Parser whereColumnName() {
    return new Word();
}

/*optOrderBy() method is a subParser recognizing the optional Order By clause
*in an SQL statement. It takes the form of
*    Order By <column name> key word
*The key word can be empty, desc (for descending), or asc (for ascending)
*By default it is ordered ascending
*/
Parser optOrderBy() {
    Alternation a=new Alternation();
    a.add(new Empty());
    Sequence s=new Sequence();
    s.add(new CaselessLiteral("orderby").discard());
    s.add(orderByNames());
    Alternation b=new Alternation();
    b.add(new CaselessLiteral("asc"));
    b.add(new CaselessLiteral("desc"));
    b.add(new Empty());
    s.add(b);
    a.add(s);
    a.setAssembler(new orderByAssembler());
    return a;
}
//parser to recognize the column (attribute) names in an orderby clause
public Parser orderByNames() {
    Sequence s=new Sequence();
    s.add(commaList(orderByName()));
    //s.setAssembler(new orderByAssembler());
    return s;
//terminal parser for orderby attributes
public Parser orderByName() {
    return new Word().setAssembler(new orderByAssembler());
}

/*groupFunction() is a subParser recognizing a groupFunction
*statement in the select line of an SQL statement. For example
*select max(salary)
*from employee
*
*This would return maximum salary from the employee table--max(salary) being the groupFunction
*/

public Parser groupFunction() {
    Sequence s=new Sequence();
    Alternation a=new Alternation();
    a.add(new CaselessLiteral("avg"));
    a.add(new CaselessLiteral("count"));
    a.add(new CaselessLiteral("max"));
    a.add(new CaselessLiteral("min"));
    a.add(new CaselessLiteral("stddev"));
    a.add(new CaselessLiteral("sum"));
    a.add(new CaselessLiteral("variance"));
    a.setAssembler(new groupFunctionAssembler());
    s.add(a);
    s.add(new Symbol('(').discard());
    s.add(groupFunctionNames());
    s.add(new Symbol(')').discard());
    //s.setAssembler(new groupFunctionAssembler());
    return s;
}

//Parser for attributes named within a Group Function
public Parser groupFunctionNames() {
    Sequence s=new Sequence();
    s.add(columnName());
    s.setAssembler(new groupFunctionAssembler());
    return s;
}

//terminal parser for groupby
public Parser groupName() {

return new Word().setAssembler(new
groupByAssembler());
}

// A Parser that recognizes the groupby clause in SQL
query
public Parser optGroupBy() {
    Alternation a=new Alternation();
    a.add(new Empty());
    Sequence s=new Sequence();
    s.add(new CaselessLiteral("groupby").discard());
    s.add(commaList(groupName()));
    a.add(s);
    // a.setAssembler(new groupByAssembler());
    return a;
}

// A Parser that recognizes the Having clause associated
with GroupBy clause
public Parser optHaving() {
    Alternation a=new Alternation();
    a.setAssembler(new havingAssembler());
    a.add(new Empty());
    Sequence s=new Sequence();
    s.add(new CaselessLiteral("having").discard());
    s.add(havingFunction());
    a.add(s);
    return a;
}

// having functions are same as groupFunctions
public Parser havingFunction() {
    Sequence s=new Sequence();
    Alternation a=new Alternation();
    a.add(new CaselessLiteral("avg"));
    a.add(new CaselessLiteral("count"));
    a.add(new CaselessLiteral("max"));
    a.add(new CaselessLiteral("min"));
    a.add(new CaselessLiteral("stddev"));
    a.add(new CaselessLiteral("sum"));
    a.add(new CaselessLiteral("variance"));
    // a.setAssembler(new groupFunctionAssembler());
    s.add(a);
    s.add(new Symbol('['));
    s.add(havingFunctionNames());
    s.add(new Symbol(']'));
    // s.setAssembler(new groupFunctionAssembler());
    return s;
}
public Parser havingFunctionNames() {
    Sequence s=new Sequence();
    s.add(havingName());
    //s.setAssembler(new groupFunctionAssembler());
    return s;
}

public Parser havingName() {
    return new Word();
}

/*The comparisons() method is a subParser recognizing a
sequence of
*an argument followed by an operator followed by an
argument. An example of using this subParser is in
*a where clause--e.g.
*    where salary > 20000
*/

public Parser comparisons() {
    Sequence s=new Sequence();
    //Alternation a=new Alternation();
    s.add(arg());
    s.add(operator());
    s.add(arg());
    return s;
}

/*arg() method recognizes an argument, whcih can take
the form of
*an expression or a quoted string
*/

public Parser arg() {
    Alternation a=new Alternation();
    //a.add(expressions());
a.add(whereExpressions());
a.add(new QuotedString());
a.setAssembler(new argAssembler());
    return a;
}
//a subParser to recognize an mathematical operator
public Parser operator() {
    Alternation o=new Alternation();
o.add(new Symbol("<"));
o.add(new Symbol(">"));
o.add(new Symbol("<="));
o.add(new Symbol(">="));
o.add(new Symbol("!="));
o.add(new Symbol("="));
o.setAssembler(new operatorAssembler());
    return o;
}

/***************************************************************************/
/*Programmer: Edwin Waite
*Date: 8/19/03
*Title: Semantic Checker
*Purpose: This object checks the user SQL query to determine
  *if it is semantically correct, meaning do the attributes
  *(i.e. columns)
  *and relations (i.e. tables) exist in the database
*/

package wbqos.parsers;
import wbqos.relAlg.*;
import java.util.*;
import java.sql.*;
import wbqos.db.*;
import wbqos.gui.*;

public class SemanticChecker {
    private String validQuery=null;
    private Vector relations=null;  //private cs600.Database db;
    private DBConnect db;
    private Connection conn;
    /** Creates a new instance of SemanticChecker */
    public SemanticChecker() {
        DBConnect db = new DBConnect();
        conn = db.getConnection(QUIGlobalData.getDatabase());
    }
public String CheckQuery(RAQuery query) {
    String validTables=null;
    String validAttributes=null;
    boolean match;
    int i,j,k;
    //Vector relations=null;
    Vector selectElements=null;
    Vector selectAttributes=null;
    Vector projectAttributes=null;
    //validate relations against database relations (ie. tables)
    RARelation r=null;
    relations = query.getRelations();
    validTables = validateTables(relations);
    if(validTables != "ok") {
        return validTables;
    }
    //validate attributes in the query
    RAProject project = query.getProjectOper();
    projectAttributes = project.getAttributes();
    RASelect select = query.getSelectOper();
    if(select != null) {
        selectElements = select.getElements();
        selectAttributes = new Vector();
        //loop through select elements and add those elements that are attributes to queryAttributes
        for(i=0;i<selectElements.size();i++) {
            Element e = (Element)selectElements.elementAt(i);
            if(e.getType().equalsIgnoreCase("a")) {
                selectAttributes.add((Attribute)e);
            }
        }
        validAttributes = validateAttributes(selectAttributes);
        if(validAttributes != "ok") {
            return validAttributes;
        }
    }
}

/*CheckQuery() has two functions:
 *1st--validate attributes and tables against database
 *2nd--If 1st step is valid, call CollectStats on the tables and attributes
 */

/*lst--validate attributes and tables against database
*/
/*2nd--If 1st step is valid, call CollectStats on the tables and attributes*/
validAttributes = validateAttributes(projectAttributes);
if(validAttributes != "ok") {
    return validAttributes;
}
collectTableStats();
return "ok";

// check to see if relations are valid tables in database
public String validateTables(Vector relations) {
    String result="ok";
    RARelation r=null;
    // boolean validTable=false;
    String validTable;
    check_tables:
    for(int i=0;i<relations.size();i++) {
        r = (RARelation)relations.elementAt(i);
        // validTable = db.isValidTable(r.getName());
        validTable = isValidTable(r.getName());
        // if(validTable == false) {
        if(validTable != ("valid")) {
            result = "Table: " + r.getName() + " " +
        validTable;
            break check_tables;
        }
    }
    return result;
}

// check to see if the table is a valid in the current database
private String isValidTable(String t) {
    String sql,result;
    result = "invalid";
    sql = "show tables;";
    try {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql);
        while(rs.next()) {
            if(rs.getString(1).equalsIgnoreCase(t))
                result = "valid";
        }
    } finally {
        stmt.close();
    }
}
catch (SQLException se) {
    return se.toString();
}

return result;

// check to see if attributes are a valid column in table
public String validateAttributes(Vector attributes) {
    String result = "ok";
    Attribute a = null;
    String validAttribute;
    check_attributes:
        for(int i=0;i<attributes.size();i++) {
            a = (Attribute) attributes.elementAt(i);
            validAttribute = isValidAttribute(a);
            if(validAttribute != "valid") {
                result = "Attribute: " + a.getName() + " " +
                validAttribute;
                break check_attributes;
            }
        }
    return result;
}

// check if the attribute is a valid attribute of one of the relations
// Gather the statistics for each attribute and add them to the relation
public String isValidAttribute(Attribute a) {
    String result = "invalid";
    String sql = "Describe ";
    String tblAttribute;
    check_attribute:
        for(int i=0;i<relations.size();i++) {
            RARelation r =
            (RARelation)relations.elementAt(i);
            sql = sql + r.getName().toLowerCase() + ";";
            try {
                Statement stmt = conn.createStatement();
                ResultSet rs = stmt.executeQuery(sql);
                while(rs.next()) {
                    tblAttribute = rs.getString("Field");
                    if(tblAttribute.equalsIgnoreCase(a.getName())) {
                        // add table name to select condition
                        attribute
// so that in creation of initial
tree the select condition

// can be used to get the table name
the select condition attribute belongs to

a.setTableName(r.getName());
result = "valid";
break check_attribute;

}
a.setDataType(rs.getString("Type").toLowerCase());
    //find out how to get the size in bytes of the column
    a.setSize(40);
    if(rs.getString("Key") != null) {

        if(rs.getString("Key").equalsIgnoreCase("PRI")){
            a.setAsPK(true);
            //set the foreign key once you know the text stored in db
        }
        r.addAttribute(a);
    }
    stmt.close();
}catch (SQLException se) {
}

}//collect statistics on relation
public void relationStat(RARelation r) {
    String sql;
    sql = "select count(*) from " + r.getName().toLowerCase();
    try {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql);
        if(rs.next()) {
            r.setNumRows(rs.getLong(1));
            r.setHasStats(true);
        }
        stmt.close();
    }catch (SQLException se) {
        r.setHasStats(false);
    }
    /*
    if(r.getName().equalsIgnoreCase("employee")) {
        r.setNumRows(5000);
        r.setHasStats(true);
    }
    else if(r.getName().equalsIgnoreCase("department"))
    {
        r.setNumRows(100);
        r.setHasStats(true);
    }
else
if(r.getName().equalsIgnoreCase("dept_location")) {
    r.setNumRows(220);
    r.setHasStats(true);
}
else if(r.getName().equalsIgnoreCase("works_on")) {
    r.setNumRows(2000);
    r.setHasStats(true);
}
else if(r.getName().equalsIgnoreCase("project")) {
    r.setNumRows(100);
    r.setHasStats(true);
}
else if(r.getName().equalsIgnoreCase("dependent")) {
    r.setNumRows(12500);
    r.setHasStats(true);
}
else {
    //do nothing
}
*/
//collect individual statistics on a attribute
public void attributeStat(Attribute a) {
    if(a.getName().equalsIgnoreCase("fname")) {
        a.setSize(20);
    }
    else if(a.getName().equalsIgnoreCase("minit")) {
        a.setSize(2);
    }
    else if(a.getName().equalsIgnoreCase("lname")) {
        a.setSize(40);
    }
    else if(a.getName().equalsIgnoreCase("ssn") || a.getName().equalsIgnoreCase("essn")) {
        a.setSize(12);
        a.setAsPK(true);
    }
    else if(a.getName().equalsIgnoreCase("bdate")) {
        a.setSize(30);
    }
    else if(a.getName().equalsIgnoreCase("address")) {
        a.setSize(100);
    }
    else if(a.getName().equalsIgnoreCase("sex")) {

a.setSize(1);
} else if(a.getName().equalsIgnoreCase("salary")) {
    a.setSize(10);
}
else if(a.getName().equalsIgnoreCase("superssn")) {
    a.setSize(12);
    a.setAsFK(true);
}
else if(a.getName().equalsIgnoreCase("dno") || a.getName().equalsIgnoreCase("dnumber") ||
        a.getName().equalsIgnoreCase("dnum")) {
    a.setSize(4);
    if(a.getName().equalsIgnoreCase("dno") ||
        a.getName().equalsIgnoreCase("dnum"))
        a.setAsFK(true);
    else
        a.setAsPK(true);
}
else if(a.getName().equalsIgnoreCase("dname")) {
    a.setSize(40);
}
else if(a.getName().equalsIgnoreCase("mgrssn")) {
    a.setSize(12);
    a.setAsFK(true);
}
else if(a.getName().equalsIgnoreCase("mgrstartdate")) {
    a.setSize(30);
}
else if(a.getName().equalsIgnoreCase("dlocation")) {
    a.setSize(50);
}
else if(a.getName().equalsIgnoreCase("pno") || a.getName().equalsIgnoreCase("pnumber")) {
    a.setSize(5);
    a.setAsPK(true);
}
else if(a.getName().equalsIgnoreCase("hours")) {
    a.setSize(6);
}
else if(a.getName().equalsIgnoreCase("pname")) {
    a.setSize(50);
}
else if(a.getName().equalsIgnoreCase("plocation")) {
a.setSize(50);
}
else
if(a.getName().equalsIgnoreCase("dependent_name")) {  
a.setSize(50);
}
else if(a.getName().equalsIgnoreCase("gender")) {  
a.setSize(1);
}
else if(a.getName().equalsIgnoreCase("dob")) {  
a.setSize(30);
}
else
if(a.getName().equalsIgnoreCase("relationship")) {  
a.setSize(30);
}
else {
    //do nothing
}

}  
/*
//collect statistics on attributes in query
public void collectAttributeStats(Vector attributes) {
    Attribute a=null;
    for(int i=0;i<attributes.size();i++) {
        a = (Attribute)attributes.elementAt(i);
        a.setTableName(db.getTableName(a.getName()));
    }
}  
//connect to database
public void ConnectDB() {
}  
*/

/*
*Programmer: Edwin Waite
*Date: 1/13/03
*Title: Argument Assembler
*Description: This object takes in an assembly that has
*matched an Argument parser and then builds the RAQuery
*object with the matched argument/
*/
package wbqos.parsers;
import wbqos.relAlg.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

public class argAssembler extends Assembler {

    public void workOn(sjm.parse.Assembly a) {
        if(a.stackIsEmpty() == true) {
        }
    else {
        Token tok=(Token)a.pop();
        String condition=tok.sval();
        RAQuery target=(RAQuery)a.getTarget();
        target.addSelectElement(new Constant(condition));
    }
    }

} /*
 *Programmer: Edwin Waite
 *Date: 1/13/03
 *Title: GroupBy Assembler
 *Description: This object takes in an assembly that has
 *matched a group by parser and then builds the RAQuery
 *object with the matched argument.
 */

package wbqos.parsers;
import wbqos.relAlg.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

public class groupByAssembler extends Assembler {

    public void workOn(sjm.parse.Assembly a) {
        if(a.stackIsEmpty() == true) {
        }
    else {
        Token tok=(Token)a.pop();
        String attr=tok.sval();
        RAQuery target=(RAQuery)a.getTarget();
    }
    }

}
target.addGroupAttr(attr);

public class groupFunctionAssembler extends Assembler {

    public void workOn(sjm.parse.Assembly a) {
        if(a.stackIsEmpty() == true) {
        } else {
            Token tok=(Token)a.pop();
            String function=tok.sval();
            RAQuery target=(RAQuery)a.getTarget();
            target.addGroupFunction(function);
        }
    }

}
package wbqos.parsers;
import wbqos.relAlg.*;
import java.util.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

public class havingAssembler extends Assembler {

    /*This method receives an Assembly object and works on the Assemblies
     *stack, which contains recognized tokens of an SQL statement.
     *Its purpose is to pop off tokens that represent a where statement
     *of an SQL query. This assembler along with the other assemblers build
     *the RAQuery object.*/
    public void workOn(Assembly a) {
        String groupFunction="";
        java.util.Vector temp=new java.util.Vector();
        if(a.stackIsEmpty() == true) {
        }
        else {
            while(a.stackIsEmpty() == false) {
                Token tok=(Token)a.pop();
                String condition=tok.sval();
                temp.add(0, condition);
            }
            for(int i=0;i<temp.size();i++) {
                groupFunction+=(String)temp.elementAt(i);
            }
            RAQuery target=(RAQuery)a.getTarget();
            target.addHavingCondition(new
                Element(groupFunction, null));
        }
    }

    /*
    *Programmer: Edwin Waite
    */
}
*Date: 1/13/03
*Title: Operator Assembler
*Description: This object takes in an assembly that has
*matched an operator in an SQL
*statement and then builds the RAQuery object with the
*matched argument.
*/

package wbqos.parsers;
import wbqos.relAlg.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

public class operatorAssembler extends Assembler {
    public void workOn(sjm.parse.Assembly a) {
        if(a.stackIsEmpty() == true) {
        }
        else {
            Token tok=(Token)a.pop();
            String condition=tok.sval();
            RAQuery target=(RAQuery)a.getTarget();
            target.addSelectElement(new Operator(condition));
        }
    }
}

/*/  
*Programmer: Edwin Waite
*Date: 1/14/03
*Title: orderBy Assembler
*Description: This object takes in an assembly that has
*matched a orderby parser and then builds the RAQuery
*object with the matched argument.
*/

package wbqos.parsers;
import wbqos.relAlg.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

public class orderByAssembler extends Assembler {
    public void workOn(sjm.parse.Assembly a) {
        
    }
}
if(a.stackIsEmpty() == true) {
} else {
    Token tok=(Token)a.pop();
    String attr=tok.sval();
    if(attr.equalsIgnoreCase("asc") ||
        attr.equalsIgnoreCase("desc")) {
        RAQuery tar=(RAQuery)a.getTarget();
        tar.setSortOrder(attr);
    } else {
        RAQuery target=(RAQuery)a.getTarget();
        target.addOrderByAttr(attr);
    }
}

/package wbqos.parsers;
import wbqos.relAlg.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

public class projectAssembler extends Assembler {

    public void workOn(sjm.parse.Assembly a) {
        //System.out.println("Entered the projectAssembler
        workOn method");
        Token tok=(Token)a.pop();
        String attr=tok.sval();
        if(attr.equalsIgnoreCase("count") ||
            attr.equalsIgnoreCase("stddev") ||
            attr.equalsIgnoreCase("avg") || attr.equalsIgnoreCase("max")
    }
|| attr.equalsIgnoreCase("min") ||
| attr.equalsIgnoreCase("sum") ||
| attr.equalsIgnoreCase("variance") | {
| //do nothing
| }
| else | {
| RAQuery target=(RAQuery)a.getTarget();
| target.addAttribute(new Attribute(attr));
| }
| }

/**
 * Programmer: Edwin Waite
 * Date: 9/05/02
 * Title: Relation Assembler
 * Purpose: To assemble the relations from an SQL query to
 * a RAQuery. It adds the relations to the RAQuery
 */

package wbqos.parsers;
import wbqos.relAlg.*;
import java.util.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

public class relationAssembler extends Assembler {

/*This method receives an Assembly object and works on the
 * stack, which contains recognized tokens of an SQL
 * statement.
 * Its purpose is to pop off tokens that represent tables
 * of the database and add them to a SQLQuery object.
 * This assembler along
 * with the attributeAssembler construct an SQLQuery
 * object that can be used
 * to validate the tables and attributes being queried
 */

public void workOn(Assembly a) {
    Token tok;
    String relation;
    RAQuery target;
    tok=(Token)a.pop();

    ...
relation=tok.sval();
target=(RAQuery)a.getTarget();
target.addRelation(relation);
}

/**
*Programmer: Edwin Waite
*Date: 9/05/02
*Title: Table Assembler
*Purpose: To assemble a SQLQuery object after a query has been parsed and recognized.
*
*/
package wbqos.parsers;
import wbqos.relAlg.*;
import java.util.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

class whereAssembler extends Assembler {
     
     /*This method receives an Assembly object and works on the Assemblies
      *stack, which contains recognized tokens of an SQL statement.
      *Its purpose is to pop off tokens that represent a where statement
      *of an SQL query. This assembler along with the other assemblers build
      *the RAQuery object.
      */
     public void workOn(Assembly a) {
         if(a.stackIsEmpty() == true) {
          }
         else {
             Token tok=(Token)a.pop();
             String condition=tok.sval();
             RAQuery target=(RAQuery)a.getTarget();
     }
target.addSelectElement(new BoolOperator(condition));
}
}

package wbqos.parsers;
import wbqos.relAlg.*/;
import sjm.parse.*/;
import sjm.parse.tokens.*/;

public class whereColumnNameAssembler extends Assembler {

    public void workOn(sjm.parse.Assembly a) {
        if(a.stackIsEmpty() == true) {
        }
        else {
            Token tok=(Token)a.pop();
            String condition=tok.sval();
            RAQuery target=(RAQuery)a.getTarget();
            target.addSelectElement(new Attribute(condition));
        }
    }
}

/*
 * Programmer: Edwin Waite
 * Date: 1/13/03
 * Title: Where Column Name Assembler
 * Description: This object takes in an assembly that has
 * matched an column name in the where clause of an SQL
 * statement and then builds the RAQuery object with the
 * matched argument.
 */
package wbqos.parsers;
import wbqos.relAlg.*;
import sjm.parse.*;
import sjm.parse.tokens.*;

public class whereFactorAssembler extends Assembler {

    public void workOn(sjm.parse.Assembly a) {
        if(a.stackIsEmpty() == true) {
        }
        else {
            Token tok=(Token)a.pop();
            if(tok.isNumber() == true) {
                Double d=new Double(tok.nval());
                //System.out.println("WhereFactorAssembler output: "+d);
                RAQuery t=(RAQuery)a.getTarget();
                t.addSelectElement(new Element(d.toString(), "c"));
            }
            else {
                String condition=tok.sval();
                //System.out.println("WhereFactorAssembler output: "+condition);
                RAQuery target=(RAQuery)a.getTarget();
                target.addSelectElement(new Constant(condition));
            }
        }
    }
}

Relational Algebra Module

/*
 * Programmer: Edwin Waite
 * Date: 10/04/03
 * Title: Attribute (i.e. Column)
 * Description: This object represents an attribute (also known as field, column)
 * within a database.
 */
package wbqos.relAlg;
import java.util.*;

public class Attribute extends Element {
    private boolean pk=false;
    private boolean fk=false;
    //private Vector indices;
    private String tableName;
    private int byteSize;
    private String dataType;

    /** Creates a new instance of Attribute */
    public Attribute() {
        //indices = new Vector();
    }
    //create an attribute
    public Attribute(String name) {
        super(name, "a");
    }
    //create a deep clone copy of this object
    public Object clone() {
        //call object.clone
        Attribute cloned = (Attribute)super.clone();
        return cloned;
    }
    //set the name of the table this attribute belongs to
    public void setTableName(String n) {
        tableName=n;
    }
    //get the table name this attribute belongs to
    public String getTableName() {
        return tableName;
    }
    //set the size in bytes of this attribute
    public void setSize(int s) {
        byteSize = s;
    }
    //get size of attribute
    public int getSize() {
        return byteSize;
    }
    //set the data type of this attribute
    public void setDataType(String dt) {
public class BoolOperator extends Element {

    /** Creates a new instance of BoolOperator */
    public BoolOperator() {
    }

    //create boolean operator with value
    public BoolOperator(String n) {
        super(n, "b");
    }

    //get the data type of this attribute
    public String getDataType() {
        return dataType;
    }

    //set this attribute as a primary key
    public void setAsPK(boolean value) {
        pk = value;
    }

    //get flag indicating if this attribute is a primary key
    public boolean isPK() {
        return pk;
    }

    //set this attribute as a foreign key
    public void setAsFK(boolean value) {
        fk = value;
    }

    //get flag indicating if this attribute is a foreign key
    public boolean isFK() {
        return fk;
    }

    //create boolean operator with value
    public BoolOperator(String n) {
        super(n, "b");
    }

    /*
    *Programmer: Edwin Waite
    *Date: 10/04/03
    *Title: Boolean Operator
    *Description: This object represents a boolean operator
    *within a RASelect condition
    */

    package wbqos.relAlg;

    public class BoolOperator extends Element {

        /** Creates a new instance of BoolOperator */
        public BoolOperator() {
        }

        //create boolean operator with value
        public BoolOperator(String n) {
            super(n, "b");
        }

        //get the data type of this attribute
        public String getDataType() {
            return dataType;
        }

        //set this attribute as a primary key
        public void setAsPK(boolean value) {
            pk = value;
        }

        //get flag indicating if this attribute is a primary key
        public boolean isPK() {
            return pk;
        }

        //set this attribute as a foreign key
        public void setAsFK(boolean value) {
            fk = value;
        }

        //get flag indicating if this attribute is a foreign key
        public boolean isFK() {
            return fk;
        }
    }
}
/create a deep clone copy of this object
public Object clone() {
    //call object.clone
    BoolOperator cloned =
    (BoolOperator)super.clone();
    return cloned;
}

/*
*Programmer: Edwin Waite
*Date: 10/04/03
*Title: Condition
*Description: This object represents a condition within a
*RASelect operation
*/

package wbqos.relAlg;

public class Condition implements Cloneable {
    private Element left=null;
    private Element op=null;
    private Element right=null;
    private int selectivity;
    private String conditionType;

    /*There are three types of conditions:
    *Constant Condition = "c"
    *Join Condition = "j"
    *Boolean Condition = "b"
    */
    public Condition() {
    }
    //create Condition of type t
    public Condition(String t) {
        conditionType = t;
    }
    //create a condition with all of its elements
    public Condition(String t, Element l, Element o, Element r) {
        conditionType = t;
        left=l;
        op=o;
        right=r;
    }

//create a deep clone copy of this object
public Object clone() {
    try {
        //call object.clone
        Condition cloned = (Condition)super.clone();
        if(left != null)
            cloned.left = (Element)left.clone();
        if(op != null)
            cloned.op = (Element)op.clone();
        if(right != null)
            cloned.right = (Element)right.clone();
        return cloned;
    } catch(CloneNotSupportedException e) {return null;}
}

//get the string that represents this condition
public String getConditionString() {
    String condition="";
    if(left != null)
        condition += left.getName();
    if(op != null)
        condition += " " + op.getName();
    if(right != null)
        condition += " " + right.getName();
    return condition;
}

//set left operand
public void setLeftOperand(Element e) {
    left = e;
}

//set the operator
public void setOperator(Element e) {
    op = e;
}

//set the right operand
public void setRightOperand(Element e) {
    right = e;
}

//get the left operand
public Element getLeftOperand() {
    return left;
}

//get the operator
public Element getOperator() {
    return op;
}
//get the right operand
public Element getRightOperand() {
    return right;
}

//set type of condition
public void setType(String t) {
    conditionType = t;
}

//get type of condition
public String getType() {
    return conditionType;
}

//set the selectivity
public void setSelectivity(int s) {
    selectivity = s;
}

//get the selectivity
public int getSelectivity() {
    return selectivity;
}

/*
 *Programmer: Edwin Waite
 *Date: 10/04/03
 *Title: Constant
 *Description: This object represents any constant (number,
 *literal, etc) within a RASelect condition
 */

package wbqos.relAlg;

public class Constant extends Element {
    /** Creates a new instance of Constant */
    public Constant() {
    }

    //create constant with value
    public Constant(String n) {
        super(n, "c");
    }

    //create a deep clone copy of this object
    public Object clone() {
        //call object.clone
        Constant cloned = (Constant)super.clone();
    }
}
public class Element implements Cloneable {
    private String name="";
    private String type="";

    /*There are four types of elements:
    * "a" type = attribute
    * "c" type = constant
    * "b" type = boolean operator
    * "o" type = operator
    */

    public Element() {
    }

    //create an element with a value
    public Element(String n, String t) {
        type = t;
        name = n;
    }

    //create a deep clone copy of this object
    public Object clone() {
        try {
            //call object.clone
            Element cloned = (Element)super.clone();
            return cloned;
        } catch(CloneNotSupportedException e) {return null;}
    }

    //set the value of this element
    public void setName(String n) {
        name = n;
    }
}
/get the value of this element  
public String getName() {  
        return name;  
    }

//get the type of this element  
public String getType() {  
        return type;  
    }

//set the type of this element  
public void setType(String t) {  
        type = t;  
    }

package wbgos.relAlg;

public class Operator extends Element {

    /** Creates a new instance of Constant */
    public Operator() {
    }

    //create constant with value
    public Operator(String n) {
        super(n, "o");
    }

}

/*
*Programmer: Edwin Waite
*Date: 10/04/03
*Title: Constant
*Description: This object represents a mathematical
*operator (ie. >, <, =, etc)
*/
package wbqos.relAlg;

public class Predicate {

    private String leftOperand;
    private String operator;
    private String rightOperand;
    private int selectivity;
    private String leftRelation=null; //left meaning left
    side of the operator
    private String rightRelation=null; //right side of
    operator
    private boolean join=false;
    private String connector;

    /** Creates a new instance of Predicate */
    public Predicate() {
    }
    //set the logical operator connecting this predicate to
    another predicate
    public void setConnector(String c) {
        connector=c;
    }
    //set the left operand
    public void setLeftOperand(String o) {
        leftOperand=o;
    }
    //get the left operand of this operation
    public String getLeftOperand() {
        return leftOperand;
    }
    //set the operator of this operation
    public void setOperator(String oper) {
        operator=oper;
    }
    //get the operator of this operation
    public String getOperator() {
        return operator;
    }
    //set the right operand in this operation
    public void setRightOperand(String o) {
        rightOperand=o;
    }
    //get right operand in an operation
    public String getRightOperand() {

return rightOperand;

// set the selectivity of this predicate
public void setSelectivity(int s) {
    selectivity = s;
}

// get the selectivity of this predicate
public int getSelectivity() {
    return selectivity;
}

// set the left relation
public void setLeftRelation(String left) {
    leftRelation = left;
}

// set the right relation
public void setRightRelation(String right) {
    rightRelation = right;
}

// get the left relation
public String getLeftRelation() {
    return leftRelation;
}

// get the right relation
public String getRightRelation() {
    return rightRelation;
}

// set if this is a join predicate
public void setAsJoin(boolean value) {
    join = value;
}

// if this predicate is a join predicate return true otherwise false
public boolean isJoin() {
    if (join == true)
        return true;
    else
        return false;
}

// if this predicate has two relations return true else return false
public boolean hasTwoRelations() {
    if (rightRelation == null)
        return false;
    else
        return true;
}
public class RACartesian extends RAOperation {

    protected RAOperation rightOperand = null;
    protected RAOperation leftOperand = null;

    /** Creates new RACartesian */
    public RACartesian() {
        super("cartesian", "Χ");
    }

    public RACartesian(String n, String s) {
        super(n, s);
    }

    public Object clone() {
        RACartesian cloned = (RACartesian) super.clone();
        if (leftOperand != null) {
            cloned.leftOperand = (RAOperation) leftOperand.clone();
        }
        if (rightOperand != null) {
            cloned.rightOperand = (RAOperation) rightOperand.clone();
        }
        return cloned;
    }

    public void setRightOperand(RAOperation r) {
        rightOperand = r;
    }
}
public RAOperation getRightOperand() {
    return rightOperand;
}

public void setLeftOperand(RAOperation l) {
    leftOperand = l;
}

public RAOperation getLeftOperand() {
    return leftOperand;
}

public String getNodeString() {
    return getSymbol();
}

package wbqos.relAlg;
import wbqos.relAlg.*;

public class RAJoin extends RACartesian {

    private Condition condition;
    /** Creates a new instance of RAJoin */
    public RAJoin() {
        super("join", "|X|"),
    }

    //create clone of this object
    public Object clone() {
        RAJoin cloned = (RAJoin)super.clone();
        cloned.condition = (Condition)condition.clone();
        return cloned;
    }

    /*
    *Programmer: Edwin Waite
    *Date: 10/14/03
    *Title: Relational Algebra Join Operation
    *Description: This object represents a relational algebra
    *Join operation. It stores the relations associated with
    *the join, join condition and other necessary data on the
    *operation.
    */
public void setJoinCondition(Condition c) {
    condition = c;
}

public Condition getJoinCondition() {
    return condition;
}

public String getNodeString() {
    Element left, op, right;
    String node = super.getSymbol();
    node += " <";
    node += condition.getConditionString();
    node += ">";
    return node;
}

package wbqos.relAlg;
import java.awt.*;
import java.lang.*;

public class RAOperation implements Cloneable {

    protected String name;
    protected String symbol;
    // graphical dimensions for drawing a relation algebra operation
    protected Point startPoint; // the upper left corner of string predicate
    protected Point centerPoint; // the bottom center of string
    protected Rectangle rectangle; // rectangle encasing the predicate

    /*
     * Programmer: Edwin Waite
     * Date: 1/10/03
     * Title: Relational Algebra Operation
     * Description: This is the super class of all Relational Algebra Operations. It contains the name of the operation and its unicode symbol.
     */
}
protected Point rightChild;
protected Point leftChild;
protected Point rightCorner; //upper right corner of predicate
protected boolean truncateNode=false;
protected RARelation intermediate=null; //this represents the intermediate relation for each relational algebra operation
protected boolean hasStats=false;

/** Creates a new instance of RAOperation */
public RAOperation() {
    startPoint = new Point(0,0);
    centerPoint = new Point(0,0);
    rectangle = new Rectangle();
    rightChild = new Point(0,0);
    leftChild = new Point(0,0);
    rightCorner = new Point(0,0);
}

//creates new instance of RAOperation with specified parameters
public RAOperation(String n, String s) {
    name=n;
    symbol=s;
    startPoint = new Point(0,0);
    centerPoint = new Point(0,0);
    rectangle = new Rectangle();
    rightChild = new Point(0,0);
    leftChild = new Point(0,0);
    rightCorner = new Point(0,0);
}

//create a deep copy of this object
public Object clone() {
    try {
        //call object.clone
        RAOperation cloned = (RAOperation)super.clone();
        cloned.startPoint = (Point)startPoint.clone();
        cloned.centerPoint = (Point)centerPoint.clone();
        cloned.rectangle = (Rectangle)rectangle.clone();
        cloned.rightChild = (Point)rightChild.clone();
        cloned.leftChild = (Point)leftChild.clone();
        cloned.rightCorner = (Point)rightCorner.clone();
        cloned.intermediate = null;
        cloned.hasStats = false;
    }
    catch(CloneNotSupportedException e) {
    }
}
}
return cloned;
    } catch(CloneNotSupportedException e) { return null; }

public void setName(String n) {
    name = n;
}

public String getName() {
    return name;
}

public void setSymbol(String s) {
    symbol = s;
}

public String getSymbol() {
    return symbol;
}

// get the string to be printed as a node on the query tree
// this returns an empty string; it should be implemented in sub classes
public String getNodeString() {
    return "";
}

// get a truncated version of the node string
// this returns an empty string; it should be implemented in sub classes
public String getTruncatedNodeString() {
    return "";
}

// a flag to determine if this node should be truncated when displaying it graphically
public boolean isTruncated() {
    return truncateNode;
}

// set truncated node flag
public void setTruncated(boolean truncate) {
    truncateNode = truncate;
}

// set the start point of predicate
public void setStartPoint(Point p) {
    startPoint = p;
}
//get the start point of predicate
public Point getStartPoint() {
    return startPoint;
}

//set the center point of predicate
public void setCenterPoint(Point c) {
    centerPoint = c;
}

//get the center point of predicate
public Point getCenterPoint() {
    return centerPoint;
}

//set the rectangle encasing the predicate, based on
//startPoint, width, height
public void setRectangleBox(int w, int h) {
    //rectangle.setBounds((int)startPoint.getX(),
    (int)startPoint.getY(), w, h);
    rectangle = new Rectangle((int)startPoint.getX(),
    (int)startPoint.getY(), w, h);
}

//get rectangle
public Rectangle getRectangleBox() {
    return rectangle;
}

//set the left child point
public void setLeftChildPoint(Point p) {
    leftChild = p;
}

//get the left child point
public Point getLeftChildPoint() {
    return leftChild;
}

//set the right child point
public void setRightChildPoint(Point p) {
    rightChild = p;
}

//get the right child point
public Point getRightChildPoint() {
    return rightChild;
}

//set the right corner point
public void setRightCornerPoint(Point p) {
    rightCorner = p;
}

//get rightCorner point
public Point getRightCornerPoint() {
    return rightCorner;
}

// set the intermediate relation
public void setIntermediateRelation(RARelation r) {
    intermediate = r;
}

// get the intermediate relation
public RARelation getIntermediateRelation() {
    return intermediate;
}

// set whether or not this RAOperation has had statistics gathered on it.
public void setHasStats(boolean value) {
    hasStats = value;
}

// get whether or not this RAOperation has had statistics gathered on it.
public boolean hasStats() {
    return hasStats;
}

package wbqos.relAlg;
import java.util.*;
import java.awt.*;

public class RAProject extends RAOperation {

    private Vector attributes;

    /** Creates a new instance of RAProject */
    public RAProject() {
        super("project", "\u03C0");
        attributes=new Vector();
    }
}
/** Creates a new instance of RAProject with a set of attributes */
public RAProject(Vector a) {
    super("project", "\003C0");
    attributes = a;
}

// create a deep copy clone of this object
public Object clone() {
    RAProject cloned = (RAProject) super.clone();
    Vector newAttributes = new Vector();
    for(int i=0; i<attributes.size(); i++) {
        Attribute a = (Attribute) attributes.elementAt(i);
        newAttributes.add(a.clone());
    }
    cloned.attributes = newAttributes;
    return cloned;
}

// add an attribute to vector of attributes
public void addAttribute(Attribute attr) {
    attributes.add(attr);
}

// return the vector of attributes
public Vector getAttributes() {
    return attributes;
}

// get the string to be printed as a node on the query tree
public String getNodeString() {
    Attribute a;
    String temp;
    String result = getSymbol() + " <";
    for(int i=0; i<attributes.size(); i++) {
        a = (Attribute) attributes.elementAt(i);
        temp = a.getName();
        // if(temp.equalsIgnoreCase("and"))
        //    result = result + ", " + temp;
        // else
        if(i == 0)
            result += temp;
        else
            result += ", " + temp;
    } // end of loop
    result = result + ">";
    return result;
}
public String getTruncatedNodeString() {
    String truncated = getNodeString();
    return truncated.substring(0, 5);
}

package wbqos.relAlg;
import java.util.*;
import sjm.utensil.*;

public class RAQuery implements PubliclyCloneable {
    private java.util.Vector relations; //holds RARelations
    private java.util.Vector cartesians; //holds RACartesian operations
    private RAProject project=null;
    private RASelect select=null;
    private RAGroupBy groupBy=null;
    private RAOrderBy orderby=null;
    private RAHaving having=null;

    /** Creates a new instance of RAQuery */
    public RAQuery() {
        relations=new java.util.Vector();
        cartesians=new java.util.Vector();
    }

    //This method creates a copy of this object
    public java.lang.Object clone() {
        try {
            return super.clone();
        }
    }

    //get a truncated node string
    public String getTruncatedNodeString() {
        String truncated = getNodeString();
        return truncated.substring(0, 5);
    }

    /*
     *Programmer: Edwin Waite
     *Date: 1/10/03
     *Title: Relational Algebra Query
     *Description: This class represents a relational algebra query. It contains all of the relational algebra operations for a single query block. It is the target object of the SQL Parser. As the SQL Parser recognizes an SQL statement it builds the RAQuery.
     */
try {
    throw new InternalError();
}

//add a relation (i.e. a database table)
public void addRelation(String r) {
    //relations.add(r);
    RARelation rel=new RARelation(r);
    relations.add(rel);
}

public java.util.Vector getRelations() {
    return relations;
}

public void addProjectAttr(Attribute attr) {
    //create project operation if it is null
    if(project == null) {
        project=new RAProject();
        project.addAttribute(attr);
    }
    else
    project.addAttribute(attr);
}

public RAProject getProjectOper() {
    return project;
}

//adds part of a select condition to the String that
//will represent the complete select conditions
public void addSelectElement(Element e) {
    if(select == null) {
        select=new RASelect();
        select.addElement(e);
    }
    else
    select.addElement(e);
}

public RASelect getSelectOper() {
    return select;
}

//add a group function
public void addGroupFunction(String f) {
    if(groupBy == null) {

groupBy=new RAGroupBy();
groupBy.addGroupFunction(f);
}
else
groupBy.addGroupFunction(f);
}
//add a group by attribute
public void addGroupAttr(String attr) {
    if(groupBy == null) {
        groupBy=new RAGroupBy();
        groupBy.addGroupByAttr(attr);
    }
    else
        groupBy.addGroupByAttr(attr);
}
//return the groupBy operation object
public RAGroupBy getGroupByOper() {
    return groupBy;
}
//add a having condition
public void addHavingCondition(Element e) {
    if(having == null) {
        having=new RAHaving();
        having.addElement(e);
    }
    else
        having.addElement(e);
}
//get the having operation
public RAHaving getHavingOper() {
    return having;
}
//add an order by attribute
public void addOrderByAttr(String attr) {
    if(orderby == null) {
        orderby=new RAOrderBy();
        orderby.addAttribute(attr);
    }
    else
        orderby.addAttribute(attr);
}
//sets the sort order for the query
public void setSortOrder(String order) {
    orderby.setSortOrder(order);
}
public RAOrderBy getOrderByOper() {
    return orderby;
}

public void createCartesian() {
    //create x number of cartesian product operations:
    //where x is one less than
    //the number of relations.
    for(int i=0;i<(relations.size() - 1);i++) {
        RACartesian c=new RACartesian();
        cartesians.add(c);
    }
}

public java.util.Vector getCartesianOperations() {
    return cartesians;
}

/package wbqos.relAlg;
import java.util.*;
import java.awt.*;

public class RARelation extends RAOperation implements Cloneable {
    private Vector relationAttributes;
    private long numRows=-1;
    private int recordSize=-1;
    private int numColumns=-1;
    private boolean inserted=false; //flag to tell if this relation has been inserted into query tree

    public RARelation() {
        relationAttributes = new Vector();
    }
}
/** Creates new Relation */
public RARelation(String n) {
    super(n, "")
    truncateNode = true;
    relationAttributes = new Vector();
}

// create a clone of this object
public Object clone() {
    RARelation cloned = (RARelation)super.clone();
    cloned.relationAttributes =
    (Vector)this.relationAttributes.clone();
    cloned.hasStats = true;
    return cloned;
}

// add a attribute to the relation
public void addAttribute(Attribute a) {
    relationAttributes.add(a);
}

// add a vector of attributes
public void addAllAttributes(Vector a) {
    relationAttributes = a;
}

// get all attributes of this relation
public Vector getAttributes() {
    return relationAttributes;
}

// remove all attributes from relation
public void removeAllAttributes() {
    relationAttributes.removeAllElements();
}

// get string representing all of the attributes in relation
public String getAttributeString() {
    String result="";
    for(int i=0;i<relationAttributes.size();i++) {
        Attribute a =
        (Attribute)relationAttributes.elementAt(i);
        if(i != relationAttributes.size())
            result += a.getName() + ",";
        else
            result += a.getName();
    }
    return result;
}

// get string to be printed as node in query tree
public String getNodeString() {
    return super.getName();
}

//get truncated version of node string
public String getTruncatedNodeString() {
    String truncated = getNodeString();
    return truncated.substring(0, 3);
}

//set the number of rows for this relation
public void setNumRows(long n) {
    numRows = n;
}

//get the number of rows
public long getNumRows() {
    return numRows;
}

//get the number of rows as a string
public String getStringNumRows() {
    return Long.toString(numRows);
}

//get the record size by summation of individual column sizes.
//this method also sets the number of columns in relation
public int getRecordSize() {
    if(numColumns != relationAttributes.size()) {
        if(relationAttributes.size() == 0) {
            recordSize = 0;
            //set number of columns
            numColumns = 0;
            return recordSize;
        }
        else {
            numColumns = 0;
            recordSize = 0;
            //loop through each attribute and sum up the byte size of each column
            for(int i=0; i<relationAttributes.size(); i++)
            {
                Attribute a = (Attribute) relationAttributes.elementAt(i);
                recordSize += a.getSize();
                numColumns++;
            }
        }
    }
    return recordSize;
}
else
    return recordSize;
}

//get the record size as string
public String getStringRecordSize() {
    return Integer.toString(getRecordSize());
}

//get the number of columns
public int getNumColumns() {
    numColumns = relationAttributes.size();
    return numColumns;
}

//get number of columns as string
public String getStringNumColumns() {
    return Integer.toString(getNumColumns());
}

/**<*
//override the super class getIntermediateRelation
public RARelation getIntermediateRelation() {
    return this;
}
/**<*/

//set whether this relation has been inserted into a query tree
public void setInserted(boolean value) {
    inserted = value;
}

//get boolean flag to tell if this relation has been inserted into query tree or not
public boolean isInserted() {
    return inserted;
}

/**<*/

/*Programmer: Edwin Waite
Date: 1/10/03
Title: Relational Algebra Select Operation
Description: This object represents a relational algebra select operation. It contains a
*java.util.java.util.Vector
*of conditions which are individual conditions within *the select operation.
package wbqos.relAlg;
import java.util.*;
import java.awt.*;

public class RASelect extends RAOperation {
    //holds individual where conditions i.e. salary > 5000
    private java.util.Vector conditions;
    private java.util.Vector elements; //holds the elements of the conditional where clause
    private boolean breakup; //when disjunctive is implemented delete this one
    private boolean disjunctive=false;

    /** Creates a new instance of RASelect */
    public RASelect() {
        super("select", "σ");
        elements=new java.util.Vector();
        conditions=new java.util.Vector();
    }
    public RASelect(String name) {
        super(name, "σ");
        elements=new java.util.Vector();
        conditions=new java.util.Vector();
    }
    //create instance of RASelect with the given condition
    public RASelect(Condition c) {
        super("select", "σ");
        elements=new Vector();
        conditions=new Vector();
        conditions.add(c);
    }
    //clone this object
    public Object clone() {
        //call object.clone
        Vector newElements = new Vector();
        Vector newConditions = new Vector();
        RASelect cloned = (RASelect)super.clone();
        for(int i=0;i<elements.size();i++) {
            Element e = (Element)elements.elementAt(i);
            newElements.add(e.clone());
        }
        for(int j=0;j<conditions.size();j++) {
            }
Condition c =
(Condition)conditions.elementAt(j);
    newConditions.add(c.clone());
} clonedelements = newElements;
cloned.conditions = newConditions;
return cloned;

/*Where conditions are received a word at a time.
*I use the elements java.util.Vector to hold all of the
words
*within the where conditional statement. Then I will
*create individual conditions out of the elements
vector and store them in the conditions vector.
*For example the elements vector holds:
*    (the | represent different indecies in the
java.util.Vector
*                salary | > | 5000 | dept_no | = | dno
*   * A single condition would be-- salary > 5000
*/
public void addElement(Element e) {
    elements.add(e);
}
//get element from front of vector
public Element getElement() {
    if(elements.size() < 1)
        return null;
    else
        return (Element)elements.elementAt(0);
}
//remove element
public Element removeElement() {
    if(elements.size() > 0)
        return (Element)elements.remove(0);
    else {
        return null;
    }
}
//returns the java.util.Vector of conditions in the
where statement
public java.util.Vector getElements() {
    return elements;
}
//get string to be printed as node in query tree
public String getNodeString() {
    if(conditions.size() > 0) {
        Condition c=null;
        String result=getSymbol() + " <";
        for(int i=0;i<conditions.size();i++) {
            c = (Condition)conditions.elementAt(i);
            if(i == 0)
                result += c.getConditionString();
            else
                result += " " + c.getConditionString();
        }
        result += " >");
        return result;
    }
    else {
        Element e=null;
        String result = getSymbol() + " <";
        for(int i=0;i<elements.size();i++) {
            e = (Element)elements.elementAt(i);
            if(i == 0)
                result += e.getName();
            else
                result += " " + e.getName();
        }
        result += " >");
        return result;
    }
}

//get a truncated version of node string
public String getTruncatedNodeString() {
    String truncated = getNodeString();
    return truncated.substring(0,5);
}

//add an AND predicate
public void addCondition(Condition c) {
    conditions.add(c);
}

//get a condition from front of vector
public Condition getCondition() {
    if(conditions.size() < 1)
        return null;
    else
        return (Condition)conditions.elementAt(0);
}
public Condition removeCondition() {
    if(conditions.size() > 0)
        return (Condition)conditions.remove(0);
    else {
        return null;
    }
}

public boolean moreConditions() {
    if(conditions.size() < 1)
        return false;
    else
        return true;
}

public java.util.Vector getConditions() {
    return conditions;
}

public void setAsDisjunctive(boolean value) {
    disjunctive = value;
}

public boolean isDisjunctive() {
    return disjunctive;
}

public void setBreakUp(boolean b) {
    breakup = b;
}

public boolean getBreakUp() {
    return breakup;
}

/*set true if the conditions of this operation can be
broken up
into individual conditions, false otherwise */
public void setBreakUp(boolean b) {
    breakup = b;
}

//get the value of breakup
public boolean getBreakUp() {
    return breakup;
}

Optimization Module

/*
*Programmer: Edwin Waite
*Date: 8/19/03
*/
*Title: Optimizer
*Purpose: This is the driving object for the optimization process. The query is parsed and optimized from this object.
*/

package wbgos.opt;
import wbgos.parsers.*;
import wbgos.relAlg.*;
import mylib.util.*;
import java.util.*;
import sjm.parse.*;
import sjm.parse.tokens.*;
import wbgos.test.*;

public class Optimizer {
    private Vector optTrees;
    private SQLParser sql;
    private RAQuery target;
    private RAQuery updated;
    private QueryTree tmpTree;
    private HOptimizer hOptimizer;
    private SemanticChecker semanticCheck;
    private testHOptimizer test;
    private BinNode tmpCurrent, startNode;

    /** Creates a new instance of Optimizer */
    public Optimizer() {
        optTrees = new Vector();
        sql = new SQLParser();
        hOptimizer = new HOptimizer();
        semanticCheck = new SemanticChecker();
        test = new testHOptimizer();
    }

    //parse user query, return false for no match, true for complete match
    public String parseQuery(String q) {
        String checkQuery;
        Assembly a = new TokenAssembly(q);
        Parser p = sql.query();
        RAQuery target = new RAQuery();
        a.setTarget(target);
        Assembly out = p.completeMatch(a);
        if (out == null) {
            return false;
        } else { //match found
            return true;
        }
    }
}
checkQuery = "Your query is not syntactically correct";
return checkQuery;
}
else {
    // get constructed target object
    updated = (RAQuery)out.getTarget();
    updated.createCartesian();
    checkQuery = semanticCheck.CheckQuery(updated);
    // checkQuery = "ok";
    // for testing the initial tree algorithm
    // testHoptimizer test = new testHoptimizer();
    // test.printTrees(optimizeQuery());
    // checkQuery = "Do Not Continue";
    return checkQuery;
}

/* start with an initial current tree. Call each optimization step with
 * a clone of the current tree (get back optimized tree)
 * store result in *optTrees vector.
 */
public Vector optimizeQuery() {
    // build initial unoptimized query tree
    tmpTree = new QueryTree();
    initialTree();
    evaluateOperations(tmpTree);
    optTrees.add(tmpTree);

    // push down select operations if there is one
    RASelect s = updated.getSelectOper();
    if(s != null) {
        tmpTree = hOptimizer.pushSelectDown((QueryTree)tmpTree.clone());
        // evaluateOperations(tmpTree.getRootNode());
        evaluateOperations(tmpTree);
        optTrees.add(tmpTree);
        tmpTree = hOptimizer.createJoins((QueryTree)tmpTree.clone());
        // evaluateOperations(tmpTree.getRootNode());
        evaluateOperations(tmpTree);
        optTrees.add(tmpTree);
    }
}
```java
public void evaluateOperations(QueryTree qTree) {
    BinNode node, leftChild, rightChild;
    RAOperation oper, childOper;
    RARelation leftIntermediate, rightIntermediate;
    QueryTreeIterator iter = qTree.queryTreeElements();
    node = (BinNode) iter.poNext();
    while (node != null) {
        oper = (RAOperation) node.getData();
        if (oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RARelation")) {
            oper.setIntermediateRelation((RARelation) oper);
        } else {
            leftChild = node.getLeftChild();
            rightChild = node.getRightChild();

            if (oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RAProject")) {
                RAProject project = (RAProject) oper;
                Vector projAttributes = project.getAttributes();
                childOper = (RAOperation) leftChild.getData();
                leftIntermediate = childOper.getIntermediateRelation();
            }
        }
    }
}
```

/*This method calculates the intermediate relation (or result) for each operation in the query tree.  
*Each node's intermediate relation is a result of the intermediate relations of its children.  
*This method uses post order traversal of the query tree to ensure that each child's intermediate relation  
*has been visited (ie. calculated) before the current node intermediate relation is calculated.  
*/
Vector leftAttributes = leftIntermediate.getAttributes();
RARelation operIntermediate = new RARelation(oper.getName() + " Intermediate");
// copy only those attributes from leftIntermediate that are a subset of project attributes
for(int i=0;i<projAttributes.size();i++)
{
    Attribute pAttrib = (Attribute)projAttributes.elementAt(i);
    for(int j=0;j<leftAttributes.size();j++) {
        Attribute lAttrib = (Attribute)leftAttributes.elementAt(j);
        if(lAttrib.getName().equalsIgnoreCase(pAttrib.getName())) {
            operIntermediate.addAttribute(lAttrib);
        }
    }
}
// copy # of rows from leftIntermediate to oper
operIntermediate.setNumRows(leftIntermediate.getNumRows());
oper.setIntermediateRelation(operIntermediate);
for(int i=0;i<leftAttributes.size();i++)
{
operIntermediate.addAttribute((Attribute)leftAttributes.elementAt(i));
}

//set #of rows based on selectivity of condition
//for now just use a fill in value
operIntermediate.setNumRows(leftIntermediate.getNumRows());
oper.setIntermediateRelation(operIntermediate);
}
else
if(oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RACartesian")) {
  int i;
  RACartesian cart = (RACartesian)oper;
  childOper =
  (RAOperation)leftChild.getData();
  leftIntermediate =
  childOper.getIntermediateRelation();
  childOper =
  (RAOperation)rightChild.getData();
  rightIntermediate =
  childOper.getIntermediateRelation();
  //copy all attributes of leftIntermediate and rightIntermediate into Intermediate Relation of RAOperation oper
  RARelation operIntermediate = new
  RARelation(oper.getName() + "_Intermediate");
  //do not use addAllAttributes because then any manipulation of the passed in vector will affect the original vector
  Vector temp =
  leftIntermediate.getAttributes();
  for(i=0;i<temp.size();i++) {
    operIntermediate.addAttribute((Attribute)temp.elementAt(i));
  }
  temp =
  rightIntermediate.getAttributes();
  for(i=0;i<temp.size();i++) {

опер进而mediate.addAttribute((Attribute)temp.elementAt(i));
}
//set the left and right operand of this
cartesian operation to the corresponding intermediate
relation
cart.setLeftOperand(leftIntermediate);
cart.setRightOperand(rightIntermediate);
// #of rows = left #of rows * right #of
rows
опер进而mediate.setNumRows(leftIntermediate.getNumRows() *
rightIntermediate.getNumRows());
cart.setIntermediateRelation(опер进而mediate);
}
else
if(опер.gettасlass().getName().equalsIgnoreCase("wbqos.relAlg.
RAJoin")) {
    int i;
    RACartesian cart = (RACartesian)опер;
    childOper =
    (RAOperation)leftChild.getData();
    leftIntermediate =
    childOper.getIntermediateRelation();
    childOper =
    (RAOperation)rightChild.getData();
    rightIntermediate =
    childOper.getIntermediateRelation();
    //copy all attributes of
leftIntermediate and rightIntermediate into Intermediate
Relation of variable опер
    RARelation опер进而mediate = new
RARelation(опер.getName() + "_Intermediate");
    //do not use addAllAttributes because
then any manipulation of the passed in vector will affect
the original vector
    Vector temp =
    leftIntermediate.getAttributes();
    for(i=0;i<temp.size();i++) {
опер进而mediate.addAttribute((Attribute)temp.elementAt(i));
}
temp =
rightIntermediate.getAttributes();
for(i=0;i<temp.size();i++) {
    operIntermediate.addAttribute((Attribute)temp.elementAt(i));
    //set the left and right operand of
    this join operation to the corresponding intermediate
    relation
    cart.setLeftOperand(leftIntermediate);
    cart.setRightOperand(rightIntermediate);
    // # of rows = selectivity of join
    condition
    operIntermediate.setNumRows(1000);
    cart.setIntermediateRelation(operIntermediate);
}
else {
    // do nothing
}
node = (BinNode)iter.poNext();
}

// create the initial unoptimized query tree
public void initialTree() {
    /* The order of a canonical or initial query tree
    (unoptimized) from top to
    * bottom is:
    * Project
    * Having
    * GroupBy
    * Select
    * Cartesian
    * Relations
    */
    Vector selectCond; // select operations
    Vector relations; // RA Relations
    int joinCount = 0; // keep count of join conditions

    RAOperation p = updated.getProjectOper();
    if(p != null) {
        tmpTree.addLeft(p);
        tmpCurrent = tmpTree.getCurrent();
    }
    RAOperation h = updated.getHavingOper();
if(h != null) {
    tmpTree.addLeft(h);
    tmpCurrent = tmpTree.getCurrent();
}
RAOperation g = updated.getGroupByOper();
if(g != null) {
    tmpTree.addLeft(g);
    tmpCurrent = tmpTree.getCurrent();
}
RAOperation s = updated.getSelectOper();
if(s != null) {
    tmpTree.addLeft(s);
    tmpCurrent = tmpTree.getCurrent();
}
relations = updated.getRelations();
RASelect selectOperation = (RASelect)s;
//set startNode as the last operation inserted
startNode = tmpCurrent;

//no where clause in the SQL query. Cartesian
Product operation is
//commutative and associative, so insertion order of
relations is not significant
if(selectOperation == null) {
    if(relations.size() == 1) { //only one relation
        insertRelations((RARelation)relations.elementAt(0), null, true);
    }
    else {
        Vector temp = new Vector();
        //loop through relations inserting two at a
time
        for(int i=0;i<relations.size();i++) {
            RARelation r =
(RARelation)relations.elementAt(i);
            if(!r.isInserted())
                temp.add(relations.elementAt(i));
            if(temp.size() == 2) {
                if(i < 2) { //this is the first
insertion of relations
                insertRelations((RARelation)temp.elementAt(0),
(RARelation)temp.elementAt(1), true);
            } else {
begin{verbatim}
    }
end{verbatim}
}
```java
    temp.removeAllElements();
    }
    else {

    insertRelations((RARelation)temp.elementAt(0),
    (RARelation)temp.elementAt(1), false);
    temp.removeAllElements();
    }
} //end of loop
//if a single relation is left insert it
if(temp.size() == 1)

insertRelations((RARelation)temp.elementAt(0), null, false);
    temp.removeAllElements();
}
}  
else {
    //break up select operation into individual
    select conditions
    hOptimizer.cascadeSelect(tmpTree);
    selectCond = selectOperation.getConditions();
    RARelation left, right;
    String tOne, tTwo; //table one and table two of
    a specific join condition
    Element e;
    Attribute a;
    //if there is only one table, there can be no
    joins
    if(relations.size() == 1) {

    insertRelations((RARelation)relations.elementAt(0), null, true);
    }
    else {
        //loop through select conditions and for
        each join operation
        //insert a RACartesian operation to join a
        single relation to existing
        //tree or to join a join operation of two
        tables to the existing tree
        int i;
        for(i=0;i<selectCond.size();i++) {
            Condition c =
            (Condition)selectCond.elementAt(i);
```
if(c.getType().equalsIgnoreCase("j")) {
    e = c.getLeftOperand();
    a = (Attribute)e;
    tOne = a.getTableName();
    e = c.getRightOperand();
    a = (Attribute)e;
    tTwo = a.getTableName();
    left = null;
    right = null;
    for(int j=0;j<relations.size();j++)
    {
        RARelation r = (RARelation)relations.elementAt(j);
        if(r.getName().equalsIgnoreCase(tOne)) {
            if(!r.isInserted())
                left = r;
        }
        if(r.getName().equalsIgnoreCase(tTwo)) {
            if(!r.isInserted())
                right = r;
        }
    } //end of relation loop
    //now insert relations into tree
    if(joinCount==0) { //if this is the first insertion
        insertRelations(left,right,true);
    } else {
        insertRelations(left,right,false);
    }
    joinCount++;
} //end of condition loop

//loop through relations and insert any that have not been inserted
Vector temp = new Vector();
for(i=0;i<relations.size();i++) {
    RARelation r = (RARelation)relations.elementAt(i);
    if(!r.isInserted())
        temp.add(r);
}
temp.add(r);
if(temp.size() == 2) {
    if(i < 2) {
        insertRelations((RARelation)temp.elementAt(0),
                        (RARelation)temp.elementAt(1), true);
        temp.removeAllElements();
    } else {
        insertRelations((RARelation)temp.elementAt(0),
                        (RARelation)temp.elementAt(1), false);
        temp.removeAllElements();
    }
}
// if a single relation is left insert it
if(temp.size() == 1) {
    insertRelations((RARelation)temp.elementAt(0), null, false);
    temp.removeAllElements();
}
}
// iterate through newly formed tree and for each RACartesian
// operation, insert references to its left and right child operations
QueryTreeIterator iter =
tmpTree.queryTreeElements();
BinNode node, leftNode, rightNode;
RAOperation oper, leftOper, rightOper;
while(iter.hasNext()) {
    node = (BinNode)iter.next();
    oper = (RAOperation)node.getData();
    if(oper.getName().equalsIgnoreCase("cartesian"))
    {
        RACartesian cart = (RACartesian)oper;
        leftNode = node.getLeftChild();
        rightNode = node.getRightChild();
        if(leftNode != null) {
            // leftOper =
            (RAOperation)leftNode.getData();
            cart.setLeftOperand((RAOperation)leftNode.getData());
        }
    }
if(rightNode != null) {
    //rightOper =
    (RAOperation)rightNode.getData();

cart.setRightOperand((RAOperation)rightNode.getData());
}
} //end of initial tree

/*Insert relations:
 * The insertion occurs after the Project,...,select operations.
 * If two relations are passed in, insert two RACartesian operations, one
 * for joining the two relations and another for joining the existing relations.
 * If one relation is passed in, insert one RACartesian joining the relation to
 * the existing relations
 */
private void insertRelations(RARelation left, RARelation right, boolean isFirst) {
    //set the current node of the tree to startNode before adding nodes to tree
    tmpTree.setCurrent(startNode);
    if(isFirst) {
        if(left != null && right != null) {
            RACartesian parent = new RACartesian();
            tmpTree.addLeft(parent);
            tmpCurrent = tmpTree.getCurrent();
            tmpTree.addLeft(left);
            tmpTree.setCurrent(tmpCurrent);
            tmpTree.addRight(right);
            left.setInserted(true);
            right.setInserted(true);
        }
        else {
            //only time a single relation would be passed in as the first insertion
            //is if there is only one relation in the From clause. No need to insert
            //a RACartesian operation if there is only one relation.
        }
if(left != null) {
    tmpTree.addLeft(left);
    left.setInserted(true);
}
if(right != null) {
    tmpTree.addLeft(right);
    right.setInserted(true);
}

else {
    if(left != null && right != null) {
        RACartesian parent = new RACartesian();
        tmpTree.addLeft(parent);
        RACartesian child = new RACartesian();
        tmpTree.addRight(child);
        tmpCurrent = tmpTree.getCurrent();
        tmpTree.addLeft(left);
        tmpTree.setCurrent(tmpCurrent);
        tmpTree.addRight(right);
        left.setInserted(true);
        right.setInserted(true);
    }
    else if(left != null && right == null) {
        RACartesian parent = new RACartesian();
        tmpTree.addLeft(parent);
        tmpTree.addRight(left);
        left.setInserted(true);
    }
    else if(left == null && right != null) {
        RACartesian parent = new RACartesian();
        tmpTree.addLeft(parent);
        tmpTree.addRight(right);
        right.setInserted(true);
    }
    else {
        //this should not happen
    }
}
}

} //end of Optimizer

/*
 * Programmer: Edwin Waite
 * Date: 2/13/03
 */
**Title: Heuristic Query Optimizer**

**Description:** This object receives an unoptimized Query Tree object and applies Heuristic rules to transform the Query represented in the Query Tree into an optimized SQL Query.

```
package wbqos.opt;
import wbqos.relAlg.*;
import mylib.util.*;
import java.util.*;

public class HOptimizer {

    private QueryTree qTree;

    /** Creates new HOptimizer */
    public HOptimizer() {
    } //create individual select conditions

    public void cascadeSelect(QueryTree qTree) {
        QueryTreeIterator iter=qTree.queryTreeElements();
        //search query tree for select operation
        //then create conditions out of the elements
        while(iter.hasNext()) {
            BinNode node=(BinNode)iter.next();
            RAOperation oper=(RAOperation)node.getData();
            if(oper.getName().equalsIgnoreCase("select")) {
                //get Elements of this select operation
                RASelect select = (RASelect)oper;
                boolean notDone = true;
                Vector temp=new Vector();
                Element left,operator,right;
                while(notDone) {
                    Element element=(Element)select.removeElement();
                    if(element == null)
                        notDone = false;
                    else {
                        if(element.getName().equalsIgnoreCase("or")) {
                            Condition c = new Condition();
                            if(element.getName().equalsIgnoreCase("or"))
                                c
                        }
                    }
                }
            }
        }
    }

```
select.setAsDisjunctive(true);  // check what type of condition needs to be created
left = (Element)temp.remove(0);
operator = (Element)temp.remove(0);
right = (Element)temp.remove(0);
if(left.getType().equalsIgnoreCase("a") && right.getType().equalsIgnoreCase("a"))
    c.setType("j");
else
    c.setType("c");
c.setLeftOperand(left);
c.setRightOperand(right);
c.setOperator(operator);
select.addCondition(c);
Condition boolCondition = new Condition("b",null,element,null);
select.addCondition(boolCondition);
if(temp.size() != 0) // this should never happen
    temp.removeAllElements();
else // element is a part of a predicate
    temp.add(element);
} // end of while
// retrieve last condition
if(temp.size() > 2) {
    Condition c = new Condition();
    left = (Element)temp.remove(0);
    operator = (Element)temp.remove(0);
    right = (Element)temp.remove(0);
    if(left.getType().equalsIgnoreCase("a")
        && right.getType().equalsIgnoreCase("a"))
        c.setType("j");
    else
        c.setType("c");
}
c.setLeftOperand(left);
c.setOperator(operator);
c.setRightOperand(right);
select.addCondition(c);

} //only supporting one select statement
currently--end iteration through tree
iter.moveToEnd();
}
} //end of while
} //end of method

/*Using equivalence rules for relation algebra push down
the
*select operations as far down the trees as the rules
allow
*/
public QueryTree pushSelectDown(QueryTree qTree) {
  //cascadeSelect(qTree);
  QueryTreeIterator iter = qTree.queryTreeElements();
  BinNode node = (BinNode) iter.find("select");
  BinNode parent;
  RAOperation oper, parentData;
  RASelect select = (RASelect) node.getData();
  Condition c=null;
  boolean notDone = true;
  boolean selectivity = true;
  /*loop through conditions and for each condition
  *move down the tree and check if the condition
  *can be commuted at each node, if true continue
  *else insert as parent (Exception: if parent is
  *an RASelect, then compare selectivity and insert
  *the operation with smallest selectivity as parent)
  */
  while(notDone) {
    c = (Condition) select.removeCondition();
    if(c == null)
      notDone = false;
    else {
      if(c.getType().equalsIgnoreCase("j") ||
        c.getType().equalsIgnoreCase("c")) {
        RASelect newSelect = new RASelect(c);
        iter.moveTo("select");
        while(iter.hasNext()) {
          node = (BinNode) iter.next();
          }
oper = (RAOperation)node.getData();
// when commute is true, continue
down tree

if(commuteSelect(oper, newSelect) == false) {
    // insert as parent unless parent is RASelect
    parent = (BinNode)node.getParent(); // get parent but don't move
    parentData = (RAOperation)parent.getData();

    if(parentData.getName().equalsIgnoreCase("select")) {
        iter.moveToParent(); // point iterator at parent; to insert new select as child
        // while the parent operation is RASelect and its selectivity < condition.selectivity
        // advance up the tree, else insert condition as a new RASelect operation
        while(selectivity) {
            /*
             * SHOULD SELECTIVITY BE STORED: RASELECT OR CONDITION************************/
             */
            if(parentData.getSelectivity() >= newSelect.getSelectivity()) {
                selectivity = false;
                iter.insertAsChild(new BinNode(newSelect));
                iter.moveToEnd();
            } else {
                (BinNode)iter.getParent();
                parentData = (RAOperation)parentData.getData();
            }
        }
    } else { // must now insert new select operation---problem:
        selectivity = false;
    }
}

118
which child to insert it as, right or left

// as you move up the

tree, you don't know which child you came from.

} }
}
} else {
  */

iter.insertAsParent(newSelect);
  iter.moveToEnd();
    //}
}
} // end of inner loop
}
} // end of else
} // end of outer loop

// if original select operation has no more

conditions, delete from tree

if(select.moreConditions() == false) {
  iter.reset();
  iter.removeOperation(select);
}

return qTree;
}

// Check if the selection condition can be commuted with the
given RAOperation

public boolean commuteSelect(RAOperation ra, RASelect select) {

if(ra.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RA
Select"))
  return true;
else
if(ra.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RA
Project")) {
  RAProject project = (RAProject)ra;
  Vector attributes = project.getAttributes();
  boolean match;
  boolean subset = true;
  boolean notDone = true;

119
Condition c=null;
Element left, right;
left = c.getLeftOperand();
right = c.getRightOperand();
Vector tempAttributes = new Vector();
if(left.getType().equalsIgnoreCase("a"))
tempAttributes.add(left);
if(right.getType().equalsIgnoreCase("a"))
tempAttributes.add(right);
while(notDone) {
    c = select.getCondition();
    if(c == null)
        notDone = false;
    else {
        is_subset:
        for(int i=0;i<tempAttributes.size();i++)
        {
            Element selectAttribute =
            (Element)tempAttributes.elementAt(i);
            match = false;
            for(int j=0;j<attributes.size();j++)
            {
                Element projectAttribute =
                (Element)attributes.elementAt(j);

                if(selectAttribute.getName().equalsIgnoreCase(projectAttribute.getName()))
                    match = true;
                }
        //if match is false one time then
        attributes in select condition are not a subset of project
        attribute list
        //and therefore the select and
        project operations are not commutable
        if(match == false) {
            subset = false;
            notDone = false;
            break is_subset; //break out of
        both loops
        }
        }
    }
}
}//end of while loop
if(subset == true)
    return true;
else
    return false;
} // end of project test
else
if(ra.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RAJoin") ||
    ra.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RACartesian")) {
    RACartesian join = (RACartesian)ra; // join can be either a cartesian join or an equiJoin
    Condition c = select.getCondition();
    if(c != null) {
        Vector leftAttributes, rightAttributes, condAttributes;
        leftAttributes = null;
        rightAttributes = null;
        Element left, right;
        left = c.getLeftOperand();
        right = c.getRightOperand();
        /* If select condition contains one Constant Attribute then it will
          * always be commutable by definition.
          */
        if(left.getType().equalsIgnoreCase("c") || right.getType().equalsIgnoreCase("c"))
            return true;
        else {
            RAOperation leftOper, rightOper;
            condAttributes = new Vector();
            condAttributes.add(left);
            condAttributes.add(right);
            leftOper = join.getLeftOperand();
            rightOper = join.getRightOperand();
        }
    }
    if(leftOper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RARelation")) {
        RARelation leftRel = (RARelation)leftOper;
        leftAttributes = leftRel.getAttributes();
    }
    if(rightOper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RARelation")) {

RARelation rightRel =
  (RARelation)rightOper;
  rightAttributes =
  rightRel.getAttributes();
}
/*If any attribute in select condition
is a member of the
  *set of attributes in tableAttributes ,
then NOT commute, otherwise commute.
*/
/*
boolean match;
boolean isMember = false;
compare_attributes:
  for(int i=0;i<condAttributes.size();i++)
  {
    Element cAttribute =
      (Element)condAttributes.elementAt(i);
    match = false;
    for(int j=0;j<tableAttributes.size();j++) {
      Attribute tAttribute =
        (Attribute)tableAttributes.elementAt(j);
      if(cAttribute.getName().equalsIgnoreCase(tAttribute.getName()))
        match = true;
    } //end of inner loop
    if(match == true) {
      isMember = true;
      break compare_attributes;
    }
  } //end of outer loop
if(isMember)
  return false;
else
  return true;
*/
/*Use logical AND test to determine if
the select operation should be commuted
*Set C = condition attributes
*Set L = attributes of leftchild
operation
*Set R = attributes of rightchild
operation
if C has elements in both L and R then
commute is false, otherwise true
*/
boolean L,R;
L = false;
R = false;
for(int i=0;i<condAttributes.size();i++)
{
    Element cAttribute =
(Element)condAttributes.elementAt(i);
    //make sure leftAttributes is not
    null
    if(leftAttributes != null) {
        for(int j=0;j<leftAttributes.size();j++) {
            Attribute tAttribute =
(Attributed)leftAttributes.elementAt(j);
            if(cAttribute.getName().equalsIgnoreCase(tAttribute.getName()));
            L = true;
        }//end of inner loop
    }
    if(rightAttributes != null) {
        //check the right attributes
        for(int k=0;k<rightAttributes.size();k++) {
            Attribute r =
(Attributed)rightAttributes.elementAt(k);
            if(cAttribute.getName().equalsIgnoreCase(r.getName()));
            R = true;
        }
    }
} //end of outer loop
if(L && R)
    return false;
else
    return true;
}
else //no condition in the new select operation
    return false;
else
if(ra.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RA Relation")) {
    /*If select condition attributes are subset of Relation attributes
     *then commute = false, otherwise true
     */
    RARelation relation = (RARelation)ra;
    boolean match;
    boolean subset = true;
    Condition c = select.getCondition();
    Element left, right;
    Vector tableAttributes = relation.getAttributes();
    Vector condAttributes = new Vector();
    left = c.getLeftOperand();
    right = c.getRightOperand();
    if(left.getType().equalsIgnoreCase("a"))
        condAttributes.add(left);
    if(right.getType().equalsIgnoreCase("a"))
        condAttributes.add(right);
    for(int i=0; i<condAttributes.size(); i++) {
        Element cAttribute =
            (Element)condAttributes.elementAt(i);
        match = false;
        for(int j=0; j<tableAttributes.size(); j++) {
            /***********************This is how it was done before changing Semantic Checker to add attributes to RARelation
            String tAttribute =
                (String)tableAttributes.elementAt(j);

            if(cAttribute.getName().equalsIgnoreCase(tAttribute))
                match = true;
            */
            Attribute tAttribute =
                (Attribute)tableAttributes.elementAt(j);

            if(cAttribute.getName().equalsIgnoreCase(tAttribute.getName()))
                match = true;
        }
        if(match == false)
            subset = false;
    }
if (subset) return false;
else return true;
}
else if (ra.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RA Union")) {
    //read up on set operation commutativity with RASelect
    return true;
}
else {
    if (ra.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RA Intersection"))
        //read up on set operation commutativity with RASelect
        return true;
    else return false;
}
/*Create join operations when a select operation with a join
*condition is preceeded by a Cartesian Cross Product operation
*/
public QueryTree createJoins(QueryTree qTree) {
    BinNode node, leftChild;
    RAOperation oper, leftOper;
    Condition c;
    QueryTreeIterator iter = qTree.queryTreeElements();
    while (iter.hasNext()) {
        node = (BinNode)iter.next();
        oper = (RAOperation)node.getData();
        if (oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RASelect")) {
            RASelect select = (RASelect)oper;
            c = select.getCondition();
            if (c != null) {
                if (c.getType().equalsIgnoreCase("j")) {
                    leftChild = node.getLeftChild();
if(leftChild != null) {
    leftOper =
    (RAOperation)leftChild.getData();

    if(leftOper.getClass().getName().equalsIgnoreCase("wbqos.rel Alg.RACartesian")) {
        RAJoin newOperation =
        new RAJoin();

        newOperation.setJoinCondition((Condition)c.clone());

        iter.insertAsParent(newOperation);
        //remove RASelect and RACartesian from tree
        iter.removeNode(node);

        iter.removeNode(leftChild);

        iter.reset();
    }
}
}

return qTree;
//end of method

/*Push project operations down the tree, keeping only those attributes
 * needed for the result and subsequent operations.
 */
public QueryTree pushDownProject(QueryTree qTree) {
    Vector subOperations=new Vector(); //holds the attributes involved in subsequent operations
    Vector prevOperations; //holds the attributes involved in previous operations
    Vector projectAttributes; //holds the attributes for the newly created project operations.

    /*At each node compare attributes from subsequent operations (symbolized by "Sa") with
     *previous operations attributes (symbolized by "Pa") for the left and right tree branches.
     *Create project operations for left and right branches of the tree, with the attributes
     *that result from (Sa Intersection Pa)
/*
QueryTreeIterator iter = qTree.queryTreeElements();
BinNode node, leftChild, rightChild;
RAOperation oper, leftOper, rightOper;

// loop through tree
while(iter.hasNext()) {
    int count = 0;
    node = (BinNode) iter.next();
    oper = (RAOperation) node.getData();

    if(! oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RARelation")) {
        // add attributes involved in this operation to the subsequent operations vector
        // thus keeping track of attributes needed in subsequent operations
        getOperationAttributes(oper, subOperations);
        // the root node is the final project list, no need to insert another project list before it
            if(! node.isRootNode()) {
                leftChild = node.getLeftChild();
                rightChild = node.getRightChild();
                if(leftChild != null) {
                    leftOper = (RAOperation) leftChild.getData();
                }
            }

            if(!(leftOper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RARelation") && oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RASelect"))) {
                prevOperations = leftOper.getIntermediateRelation().getAttributes();
                projectAttributes = intersectionOfAttributes(prevOperations, subOperations);
            // only create a project operation if there are attributes returned from intersectionOfAttributes
                // And if doing so would decrease the number of attributes
                if(projectAttributes.size() > 0 && projectAttributes.size() < prevOperations.size()) {
                    // create new project operation, add the projectAttributes, and insert it into the tree


 */
if(rightChild != null) {
    rightOper = (RAOperation) rightChild.getData();
    // project operations have > selectivity than select operations (99.9% of time), so it is practically never advantageous
    // to precede a select operation on a relation with a project operation

    if(!rightOper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RARelation") &&
        oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RASelect"))) {
        prevOperations = rightOper.getIntermediateRelation().getAttributes();
        projectAttributes = intersectionOfAttributes(prevOperations, subOperations);
        // only create a project operation if there are attributes returned from
        intersectionOfAttributes
        // And if doing so would decrease
        the number of attributes
        if(projectAttributes.size() > 0 && projectAttributes.size() < prevOperations.size()) {
            // create new project operation, add the projectAttributes, and insert it into the tree
            RAProject newProject = new RAProject(projectAttributes);
            qTree.setCurrent(node);
            qTree.addRight(newProject);
        }
    }
}
}
}
}

128
return qTree;
}  //end of pushDownProject

/* Given a Relational Algebra operation add any attributes involved in the operation * to the passed in vector. This method will keep a running total of attributes * involved in operations within a query tree */
   
   public void getOperationAttributes(RAOperation oper, Vector subsequent) {

   if(oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RAProj")) {
      RAProject p = (RAProject)oper;
      Vector pAttributes = p.getAttributes();
      for(int i=0;i<pAttributes.size();i++) {
         Attribute a = (Attribute)pAttributes.elementAt(i);
         if(!isDuplicate(subsequent, a))
            subsequent.add(a);
      }
   }
   else
   if(oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RASelect")) {
      RASelect s = (RASelect)oper;
      Vector conditions = s.getConditions();
      //loop through conditions adding attributes of conditions into subsequent
      for(int i=0;i<conditions.size();i++) {
         Condition c = (Condition)conditions.elementAt(i);
         if(c.getType().equals("c")) {
            Element e = c.getLeftOperand();
            Attribute a = (Attribute)e;
            if(!isDuplicate(subsequent, a))
               subsequent.add(a);
         }
         if(c.getType().equals("j")) {
            Element e = c.getLeftOperand();
            Attribute a = (Attribute)e;
            subsequent.add(a);
            e = c.getRightOperand();
            a = (Attribute)e;
            if(!isDuplicate(subsequent, a))
               subsequent.add(a);
         }
      }
   }
}
subsequent.add(a);

}
}

if(oper.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RAJoin")) {
RAJoin j = (RAJoin)oper;
Condition c = j.getJoinCondition();
//get attributes from c and add them to subsequent

Element e = c.getLeftOperand();
Attribute a = (Attribute)e;
if(!isDuplicate(subsequent, a))
subsequent.add(a);

e = c.getRightOperand();
a = (Attribute)e;
if(!isDuplicate(subsequent, a))
subsequent.add(a);
}
else {
// do nothing
}

/*Compute the intersection of the two vectors of attributes and return *the results.
*/

public Vector intersectionOfAttributes(Vector previous, Vector subsequent) {
Vector intersection = new Vector();
for(int i=0;i<previous.size();i++) {
    Attribute p = (Attribute)previous.elementAt(i);
    for(int j=0;j<subsequent.size();j++) {
        Attribute s = (Attribute)subsequent.elementAt(j);

        if(p.getName().equalsIgnoreCase(s.getName()))
            intersection.add(p);
    }
}
return intersection;
}

// check to see if attribute is already in the vector
public boolean isDuplicate(Vector v, Attribute a) {
boolean result = false;
for(int i=0;i<v.size();i++) {
    Attribute e = (Attribute)v.elementAt(i);
    if(a.getName().equalsIgnoreCase(e.getName()))
        result = true;
}
return result;
}

package wbqos.opt;
import mylib.util.*;
import java.util.*;
import java.lang.*;
import wbqos.relAlg.*;
public class QueryTree implements Cloneable {
    protected BinNode root=null;
    protected BinNode current=null;
    protected int height=0;
    private Queue queue=null;
    /** Creates a new instance of QueryTree */
    public QueryTree() {
        queue = new Queue();
    }
    //over-ride the clone method and create a deep copy of this object
    public Object clone() {
        try {
            //call object.clone
            QueryTree cloned = (QueryTree)super.clone();
            /*cloning the root node starts a chain reaction
             *where each node clones it children until all
             */
            return result;
        }
    }
}

/*
*Programmer: Edwin Waite
*Date: 5/31/03
*Title: Query Tree
*Purpose: A binary tree that represents an sql query in relational
*algebra form
*/

package wbqos.opt;
import mylib.util.*;
import java.util.*;
import java.lang.*;
import wbqos.relAlg.*;
public class QueryTree implements Cloneable {
    protected BinNode root=null;
    protected BinNode current=null;
    protected int height=0;
    private Queue queue=null;
    /** Creates a new instance of QueryTree */
    public QueryTree() {
        queue = new Queue();
    }
    //over-ride the clone method and create a deep copy of this object
    public Object clone() {
        try {
            //call object.clone
            QueryTree cloned = (QueryTree)super.clone();
            /*cloning the root node starts a chain reaction
             *where each node clones it children until all
             */
            return result;
        }
    }
}

package wbqos.opt;
import mylib.util.*;
import java.util.*;
import java.lang.*;
import wbqos.relAlg.*;
public class QueryTree implements Cloneable {
    protected BinNode root=null;
    protected BinNode current=null;
    protected int height=0;
    private Queue queue=null;
    /** Creates a new instance of QueryTree */
    public QueryTree() {
        queue = new Queue();
    }
    //over-ride the clone method and create a deep copy of this object
    public Object clone() {
        try {
            //call object.clone
            QueryTree cloned = (QueryTree)super.clone();
            /*cloning the root node starts a chain reaction
             *where each node clones it children until all
             */
            return result;
        }
    }
}

package wbqos.opt;
import mylib.util.*;
import java.util.*;
import java.lang.*;
import wbqos.relAlg.*;
public class QueryTree implements Cloneable {
    protected BinNode root=null;
    protected BinNode current=null;
    protected int height=0;
    private Queue queue=null;
    /** Creates a new instance of QueryTree */
    public QueryTree() {
        queue = new Queue();
    }
    //over-ride the clone method and create a deep copy of this object
    public Object clone() {
        try {
            //call object.clone
            QueryTree cloned = (QueryTree)super.clone();
            /*cloning the root node starts a chain reaction
             *where each node clones it children until all
             */
            return result;
        }
    }
}

package wbqos.opt;
import mylib.util.*;
import java.util.*;
import java.lang.*;
import wbqos.relAlg.*;
public class QueryTree implements Cloneable {
    protected BinNode root=null;
    protected BinNode current=null;
    protected int height=0;
    private Queue queue=null;
    /** Creates a new instance of QueryTree */
    public QueryTree() {
        queue = new Queue();
    }
    //over-ride the clone method and create a deep copy of this object
    public Object clone() {
        try {
            //call object.clone
            QueryTree cloned = (QueryTree)super.clone();
            /*cloning the root node starts a chain reaction
             *where each node clones it children until all
             */
            return result;
        }
    }
}

package wbqos.opt;
import mylib.util.*;
import java.util.*;
import java.lang.*;
import wbqos.relAlg.*;
public class QueryTree implements Cloneable {
    protected BinNode root=null;
    protected BinNode current=null;
    protected int height=0;
    private Queue queue=null;
    /** Creates a new instance of QueryTree */
    public QueryTree() {
        queue = new Queue();
    }
    //over-ride the clone method and create a deep copy of this object
    public Object clone() {
        try {
            //call object.clone
            QueryTree cloned = (QueryTree)super.clone();
            /*cloning the root node starts a chain reaction
             *where each node clones it children until all
             */
            return result;
        }
    }
}
*nodes in the query tree have been copied. The clone method of the BinNode class, only creates clones for the *leftChild and rightChild BinNode, it does not clone the parent *or the data encapsulated by the BinNode. Below I loop through *the tree and create clones of the data and set each nodes parent *pointer. */

```java
if (root != null)
    cloned.root = (BinNode)root.clone();
cloned.current = cloned.root;
cloned.queue = (Queue)queue.clone();
BinTreeIterator iter = cloned.elements();
while (iter.hasNext()) {
    BinNode current, leftChild, rightChild;
    RAOperation oper;
    current = (BinNode)iter.next();
    oper = (RAOperation)current.getData();
    current.setData(oper.clone());
    if (current.hasLeftChild()) {
        leftChild = current.getLeftChild();
        leftChild.setParent(current);
    }
    if (current.hasRightChild()) {
        rightChild =
        (BinNode)current.getRightChild();
        rightChild.setParent(current);
    }
}
return cloned;
} catch(CloneNotSupportedException e) {return null;}
```

//update nodes when a new node is added as an internal node

```java
public void updateNodes(BinNode newNode, int which) {
    BinNode leftChild, rightChild;
    // we are adding a left child
    if (which == 0) {
        leftChild = current.getLeftChild();
        newNode.setParent(current);
        current.setLeftChild(newNode);
    }
```
leftChild.setParent(newNode);
newNode.setLeftChild(leftChild);
newNode.setChildType(0);
newNode.setDepth(leftChild.getDepth());
// set current to the node just added
current = newNode;
updateDepth(newNode, true);

} //we are adding a right child
else {
    rightChild = current.getRightChild();
    newNode.setParent(current);
    current.setRightChild(newNode);
    rightChild.setParent(newNode);
    // when adding a new node into an internal
    node
    // connect the sub tree to its left side
    newNode.setLeftChild(rightChild);
    newNode.setChildType(1);
    newNode.setDepth(rightChild.getDepth());
    // set current to the node just added
    current = newNode;
    // it is now a left child
    rightChild.setChildType(0);
    updateDepth(newNode, true);
    }

    } //increment the depth on all children of start node
public void updateDepth(BinNode start, boolean increment) {
    Queue updateQ = new Queue();
    BinNode next;
    boolean notDone = true;
    // push left and right child of start node onto queue
    if (start.hasLeftChild() == true)
        updateQ.push(start.getLeftChild());
    if (start.hasRightChild() == true)
        updateQ.push(start.getRightChild());
    // make sure the queue has a node in it to start with
    if (updateQ.isEmpty() == true)
        notDone = false;
    while (notDone) {
        next = (BinNode) updateQ.pop();

//increment or decrement depth of node
if (increment)
    next.setDepth(next.getDepth() + 1);
else
    next.setDepth(next.getDepth() - 1);
if (next.hasLeftChild() == true)
    updateQ.push(next.getLeftChild());
if (next.hasRightChild() == true)
    updateQ.push(next.getRightChild());
//object into next.leftChild or
next.rightChild
if (updateQ.isEmpty() == true)
    notDone = false;
}//end of while

updateQ.removeAll();

//return the level of the last child of passed in node
public int lastChildLevel(BinNode start) {
    Queue levelQ = new Queue();
    boolean notDone = true;
    BinNode next = null;
    levelQ.push(start);

    while (notDone) {
        next = (BinNode) levelQ.pop();
        if (next.hasLeftChild() == true)
            levelQ.push(next.getLeftChild());
        if (next.hasRightChild() == true)
            levelQ.push(next.getRightChild());
        //if queue is empty we have last child node
        if (levelQ.isEmpty() == true)
            notDone = false;
    } //end of while
    //remove all elements of queue
    levelQ.removeAll();
    //return the level of the last
    return (height - next.getDepth());
}

//add a left child to current node
public void addLeft(Object obj) {
    /*There are two reasons the height of the tree needs to be
    * incremented when adding a new node to the current
    node, they are:
1. Current node is at level 0
2. A node in the sub tree of current node is at level 0

BinNode newNode = new BinNode(obj);
if(root == null) {
    root = newNode;
    root.setChildType(-1);
    root.setDepth(0);
    root.setFarLeft(true);
    current = root;
} else {
    if(current.hasLeftChild() == true) {
        if(lastChildLevel(current.getLeftChild()) == 0)
            height++;
        if(current.isFarLeft() == true)
            newNode.setFarLeft(true);
        //add new node into tree and update the depth of all descendants
        updateNodes(newNode, 0);
    } //current node does not have a left child
    else {
        //if current node is at level 0 then increment height
        if(height - current.getDepth() == 0) {
            height++;
        }
        if(current.isFarLeft() == true)
            newNode.setFarLeft(true);
        //add new node as left child of current node
        current.setLeftChild(newNode);
        newNode.setParent(current);
        newNode.setDepth(current.getDepth() + 1);
        newNode.setChildType(0); //left child
        current = newNode;
    }
} //end of method
//add a right child to current node--this is for only adding a right child
//where there are no existing right children.
public void addRight(Object obj) {

BinNode newNode = new BinNode(obj);
//could put exception in if root == null
//assumption is that a right child would never be added
//unless a left child exists.
if(current.hasRightChild() == true) {
    if(lastChildLevel(current.getRightChild()) == 0)
        height++;
    updateNodes(newNode, 1);
} else {
    //if current node is at level zero then increase height
    if(height - current.getDepth() == 0)
        height+=1;
    current.setRightChild(newNode);
    newNode.setParent(current);
    newNode.setDepth(current.getDepth() + 1);
    newNode.setChildType(1); //right child
    current = newNode;
}
//remove the current node from the tree
//user must make sure current node is set to the node to be removed
public boolean remove() {
    if(current.isRootNode()) {
        if(current.hasLeftChild() &&
            !(current.hasRightChild())) {
            BinNode child;
            child = root.getLeftChild();
            child.setParent(null);
            child.setChildType(-1);
            child.setDepth(child.getDepth() - 1);
            root.finalize();
            root = child;
            current = root;
            //when removing root node height of tree must be decremented
            height--;
            updateDepth(child, false);
            return true;
        }
    } else //cannot remove if root node has two children

return false;
}
else if(current.hasLeftChild() &&
!(current.hasRightChild())) {
    // if current is left child
    if(current.getChildType() == 0) {
        BinNode parent, leftChild;
        // check to see if height of tree needs to be changed
        if(decrementHeight())
            height--;
        parent = current.getParent();
        leftChild = current.getLeftChild();
        parent.setLeftChild(leftChild);
        leftChild.setParent(parent);
        // finalize current
        current.finalize();
        current = leftChild;
        // decrement current.leftchild's depth
        leftChild.setDepth(leftChild.getDepth() - 1);
        updateDepth(current, false);  // for descendants of decrement the depth
        return true;
    }
    // current is right child
    else {
        BinNode parent, child;
        // check to see if height of tree needs to be changed
        if(decrementHeight())
            height--;
        parent = current.getParent();
        child = current.getLeftChild();
        parent.setRightChild(child);
        child.setParent(parent);
        child.setChildType(1);
        child.setDepth(child.getDepth() - 1);
        current.finalize();
        current = child;
        updateDepth(current, false);
        return true;
    }
}
else if(current.isLeafNode()) {

137
BinNode parent;
// if current is left child
if (current.getChildType() == 0) {
    // check to see if height of tree needs to be changed
    if (decrementHeight())
        height--;
    parent = current.getParent();
    parent.setLeftChild(null);
    current.setParent(null);
    // finalize current
    current.finalize();
    current = parent;
    return true;
}
else { // current is rightchild
    if (decrementHeight())
        height--;
    parent = current.getParent();
    parent.setRightChild(null);
    current.setParent(null);
    // finalize current
    current.finalize();
    current = parent;
    return true;
}

// current has two children
else {
    BinNode parent, leftChild, rightChild;
    parent = current.getParent();
    if (parent.hasLeftChild() && parent.hasRightChild())
        return false; // cannot remove this node
    else {
        // check to see if height of tree needs to be changed
        if (decrementHeight())
            height--;
        leftChild = current.getLeftChild();
        rightChild = current.getRightChild();
        parent.setLeftChild(leftChild);
        parent.setRightChild(rightChild);
        leftChild.setParent(parent);
        rightChild.setParent(parent);
    }
}
leftChild.setDepth(leftChild.getDepth() - 1);
rightChild.setDepth(rightChild.getDepth() - 1);

// finalize current
current.finalize();
current = leftChild;
updateDepth(leftChild, false);
updateDepth(rightChild, false);
return true;

// check to see if height needs to be decremented after removing current node
public boolean decrementHeight() {
    boolean decrement = true;
    if(lastChildLevel(current) != 0)
        return false;
    else {
        boolean notDone = true;
        BinNode next;
        queue.push(root);
        // iterate through tree, except descendants of current node
        // if a node is at level zero, then don't decrement tree
        while(notDone) {
            next = (BinNode)queue.pop();
            if(next.hasLeftChild()) {
                if(!next.equals(current))
                    queue.push(next.getLeftChild());
            }
            if(next.hasRightChild()) {
                if(!next.equals(current))
                    queue.push(next.getRightChild());
            }
            if(queue.isEmpty())
                notDone = false;
            if((height - next.getDepth()) == 0) {
                if(!next.equals(current)) {
                    decrement = false;
                    notDone = false;
                }
            }
        }
    }
}
private int decrement();

//insert a new child-- level order insert
public void insert(Object obj) {
  boolean notDone=true;
  BinNode next, tmpcurrent;
  // check to see if this is first node
  if(root == null) {
    next = new BinNode(obj);
    next.setDepth(0);
    next.setFarLeft(true);
    next.setChildType(-1);
    current = root = next;
  }
  else {
    // push each node on to queue in level order starting from
    // current node. Current node is the starting point
    // for insertion
    // into the tree. For example, if the root node is a
    // Project and its
    // left child is a Select, no nodes should be
    // inserted as right children
    // to these operations, so the starting point would
    // be the left child of the
    // select operation, typically a RACartesian.
    /*
     queue.push(current);
     while(notDone) {
       next=(BinNode)queue.pop();
       if(next.hasLeftChild() == true)
         queue.push(next.getLeftChild());
       if(next.hasRightChild() == true)
         queue.push(next.getRightChild());
       //if this condition is true then we should
       insert the
       // object into next.leftChild or
       next.rightChild
       if(next.hasLeftChild() == false ||
       next.hasRightChild() == false) {
         notDone=false;
         if(next.hasLeftChild() == false) {
           //
           //
//hold original current
tmpcurrent = current;
//temporarily assign current as next
so addLeft can be done
current = next;
addLeft(obj);
//reset current to original current
current = tmpcurrent;
}
else {
    tmpcurrent = current;
current = next;
addRight(obj);
current = tmpcurrent;
}
} //end of if
} //end of while
//remove all elements of queue
queue.removeAll();
} //end of else
} //end of method
//set current node
public void setCurrent(BinNode c) {
current = c;
}
public BinNode getCurrent() {
    return current;
}
//increment the height of the tree
public void incrementHeight() {
    height++;
}
//get the height of tree
public int getHeight() {
    return height;
}
//get the root node
public BinNode getRootNode() {
    return root;
}
//returns an new Binary Tree Iterator
public BinTreeIterator elements() {
    return new BinTreeIterator(this);
}
//return a new Query Tree Iterator

141
public QueryTreeIterator queryTreeElements() {
    return new QueryTreeIterator(this);
}

}//end of QueryTree

Query User Interface Module

/*
*Programmer: Edwin Waite
*Date: 3/16/2004
*Title: Query User Interface for SQL Queries
*Purpose: Provide a graphical user interface for entering SQL queries for WBQOS.
*/

package wbqos.gui;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import wbqos.db.*;
import java.sql.*;
import java.util.*;
import mylib.sql.*;

public class QUI_Query extends JPanel {

    private JTextArea resultArea;
    private JTextArea queryArea;
    private Box resultBox;
    private DBConnect db;
    private JComboBox cboxDB;
    private JComboBox cboxLColor;
    private JComboBox cboxNColor;
    private JCheckBox cbNativeSQL;
    private QUI_Hopt hoptimizer;

    public QUI_Query() {
        db = new DBConnect();
        // instantiate resultArea now, so it can be used
        // for displaying errors that may occur in this
        resultArea = new JTextArea();
this.setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
JSplitPane pane = new JSplitPane();
pane.setOrientation(JSplitPane.VERTICAL_SPLIT);
pane.setDividerSize(5);
pane.setDividerLocation(250);
pane.setLeftComponent(createTopBox());
pane.setRightComponent(createBottomBox());
this.add(pane);

//This method constructs the top gui
public Box createTopBox() {
    Box top = Box.createHorizontalBox();
    Box spacer = Box.createHorizontalBox();
    spacer.add(Box.createHorizontalStrut(20));
    Box glueSpace = Box.createHorizontalBox();
    glueSpace.add(Box.createHorizontalGlue());
    top.add(spacer);
    top.add(createQueryBox());
    //top.add(Box.createHorizontalStrut(20));
    top.add(Box.createHorizontalGlue());
    top.add(createSubmitBox());
    //top.add(Box.createHorizontalStrut(50));
    top.add(Box.createHorizontalGlue());
    top.add(createPreferenceBox());
    top.add(glueSpace);
    return top;
}

//create the bottom gui
public Box createBottomBox() {
    return createResultsBox();
}

//create the box that will contain the area where sql
//queries are entered
public Box createQueryBox() {
    Box q = Box.createVerticalBox();
    q.add(Box.createVerticalStrut(20));
    JLabel lblQuery = new JLabel("Enter Query", SwingConstants.CENTER);
    q.add(lblQuery);
    queryArea = new JTextArea(100,50);
    JScrollPane queryScroll = new JScrollPane(queryArea);
queryArea.setBorder(new BevelBorder(BevelBorder.LOWERED));
q.add(queryScroll);
q.add(Box.createVerticalStrut(20));
return q;
}

//create the submit box, which contains the submit and clear
//buttons for submitting the query
public Box createSubmitBox() {
    Box container = Box.createHorizontalBox();
    Box s = Box.createVerticalBox();
    Box c = Box.createVerticalBox();
    //set the size of the box to as tall as queryBox.
    JButton submit = new JButton("Submit");
    //add action listener to button
    submit.addActionListener(new ButtonListener());
    JButton clear = new JButton("Clear");
    //add action listener to button
    clear.addActionListener(new ButtonListener());
    s.add(Box.createVerticalGlue());
    s.add(submit);
    s.add(Box.createVerticalStrut(20));
    c.add(Box.createVerticalGlue());
    c.add(clear);
    c.add(Box.createVerticalStrut(20));
    container.add(s);
    container.add(Box.createHorizontalStrut(10));
    container.add(c);
    return container;
}

//create the preferences box, which gives the user
//the ability to set preferences for WBQOS.
public Box createPreferenceBox() {
    Box p = Box.createVerticalBox();
    Box h = Box.createHorizontalBox();
    cbNativeSQL = new JCheckBox();
    JLabel lblNativeSQL = new JLabel("Native SQL: ");
    h.add(lblNativeSQL);
    h.add(Box.createHorizontalStrut(10));
    h.add(cbNativeSQL);
    h.add(Box.createHorizontalGlue());
    JLabel lblDb = new JLabel("Database",
    SwingConstants.LEADING);
}

144
JLabel lcolor = new JLabel("Line Color", SwingConstants.LEADING);
JLabel ncolor = new JLabel("Node Color", SwingConstants.LEADING);
Dimension d = new Dimension(150, 30);
cboxDB = new JComboBox();
cboxLColor = new JComboBox();
cboxNColor = new JComboBox();
cboxDB.setMaximumSize(d);
cboxLColor.setMaximumSize(d);
cboxNColor.setMaximumSize(d);
    //fill the comboboxes
String query = "Show Databases;"
fillComboBox(cboxDB, query, "Database");
Vector colors = getColors();
fillComboBox(cboxLColor, colors);
fillComboBox(cboxNColor, colors);
p.add(Box.createVerticalStrut(20));
p.add(h);
p.add(lbldb);
p.add(cboxDB);
p.add(Box.createVerticalGlue());
p.add(lcolor);
p.add(cboxLColor);
p.add(Box.createVerticalGlue());
p.add(ncolor);
p.add(cboxNColor);
p.add(Box.createVerticalStrut(20));
return p;
}

//parse a multi lined query an remove the carriage returns
public String parseQuery(String q) {
    StringTokenizer tok=new StringTokenizer(q, "\n");
    String token,results;
    results = "";
    while(tok.hasMoreTokens()) {
        token=tok.nextToken();
        results=results + " " + token;
    }
    return results.trim();
}

//create the query results component of the gui
public Box createResultsBox() {
    resultBox = Box.createHorizontalBox();
}
// set size of box
// r.add(resultArea);
return resultBox;
}

// populate Combo Box with data from database
private void fillComboBox(JComboBox cb, String sql, String fldName) {
    Connection conn;
    conn = db.getConnection(QUIGlobalData.getDatabase());
    if(conn == null) {
        displayError("Error making connection to database");
    } else {
        try {
            Statement st = conn.createStatement();
            ResultSet rs = st.executeQuery(sql);
            // loop through record set loading values
            while(rs.next()) {
                cb.addItem(rs.getString(fldName));
            }
            st.close();
            conn.close();
        } catch (SQLException se) {
            displayError(se.toString());
        }
    }
}

// populate Combo Box with array of data
private void fillComboBox(JComboBox cb, Vector data) {
    for(int i=0;i<data.size();i++)
        cb.addItem(data.elementAt(i));
}

// return a vector of colors
private Vector getColors() {
    Vector colors = new Vector();
    /*
    colors.add(Color.red.toString());
    colors.add(Color.black.toString());
    colors.add(Color.blue.toString());
    colors.add(Color.green.toString());
    colors.add(Color.magenta.toString());
    */
    return colors;
}
colors.add(Color.yellow.toString());
colors.add(Color.cyan.toString());
*/
colors.add("black");
colors.add("red");
colors.add("blue");
colors.add("green");
colors.add("magenta");
colors.add("yellow");
colors.add("cyan");
return colors;
} //display errors that occur to user
public void displayError(String error) {
    resultArea.append(error + "\n");
}
//display the query results in resultsArea
public void displayQueryResults(String q) {
    Connection conn;
    Statement st;
    ResultSet rs;
    ResultSetMetaData metaData;
    String tabSpace = "    ";
    conn = db.getConnection(QUIGlobalData.getDatabase());
    try {
        st = conn.createStatement();
        rs = st.executeQuery(q);
        metaData = rs.getMetaData();
        while(rs.next()) {
            for(int i=1;i<=metaData.getColumnCount();i++) {
                resultArea.append(rs.getString(i) +
tabSpace);
            }
            resultArea.append("\n");
        }
        st.close();
        conn.close();
    } catch (SQLException se) {
        displayError(se.toString());
    }
} //return a result set for the query
public ResultSet getResultSet(String query) {
    Statement stmt;
    ResultSet rs;
    Connection conn;
    //conn =
    try {
        stmt = conn.createStatement();
        rs = stmt.executeQuery(query);
        return rs;
    } catch (SQLException se) {
        return null;
    }
}

//set the Hueristic Optimizer
public void setHOptimizer(QUI_Hopt opt) {
    hoptimizer = opt;
}

//This private class handles the events for the submit and clear buttons
private class ButtonListener implements java.awt.event.ActionListener {
    private String usrQuery;
    private String action;
    private boolean submitBtn=false;
    private boolean clearBtn = true;
    public ButtonListener() {
    }
    public String parseQuery(String q) {
        StringTokenizer tok=new StringTokenizer(q, "\n");
        String token,results;
        String token,results;
        String results;
results = "";
while(tok.hasMoreTokens()) {
    token = tok.nextToken();
    results = results + " " + token;
}
return results.trim();

// method called when button is clicked
public void actionPerformed(java.awt.event.ActionEvent evt) {
    action = evt.getActionCommand();
    if (action.equalsIgnoreCase("clear")) {
        queryArea.setText("");
        resultBox.removeAll();
        resultBox.repaint();
        hoptimizer.clearOptimization();
    }
    if (action.equalsIgnoreCase("submit")) {
        usrQuery = queryArea.getText();
        if (usrQuery == "") {
            queryArea.setText("You must enter a query before clicking the submit button");
        } // end of if
        else {
            // set all global variables
            QUIGlobalData.setDatabase((String)cboxDB.getSelectedItem());
            QUIGlobalData.setLineColor((String)cboxLColor.getSelectedItem());
            QUIGlobalData.setNodeColor((String)cboxNColor.getSelectedItem());
            // clear previous components from the result area
            resultBox.removeAll();
            if (cbNativeSQL.isSelected()) {
                // do not use WBQOS to parse and optimize the query
                // simply pass it to the backend db
                RSTableModel model = new RSTableModel(getResultSet(usrQuery));
                JTable resultTable = new JTable(model);
            }
        }
    }
}
resultTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

JScrollPane scrollResult = new JScrollPane(resultTable);

resultBox.add(scrollResult);
resultBox.validate();

} else {
    // use WBQOS to parse and optimize query
    String optResults;
    usrQuery = parseQuery(usrQuery);
    optResults = hoptimizer.startOptimization(usrQuery);

    if(optResults.equalsIgnoreCase("ok")) {
        // pass query to database to get query results
        // displayQueryResults(queryArea.getText());
        RSTableModel model = new RSTableModel(getResultSet(usrQuery));
        JTable resultTable = new JTable(model);

        resultTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        JScrollPane scrollResult = new JScrollPane(resultTable);

        resultBox.add(scrollResult);
        resultBox.validate();
    } else {
        RSTableModel model = new RSTableModel(optResults);
        JTable resultTable = new JTable(model);

        resultTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        JScrollPane scrollResult = new JScrollPane(resultTable);

        resultBox.add(scrollResult);
        resultBox.validate();
    }
}
// end of if
// end of actionPerformed
// end of inner class
}//end of QUI_Query

.isDefined...
pane = new JSplitPane();
top = new JPanel();
top.setLayout(new BoxLayout(top, BoxLayout.X_AXIS));
topMainBox = Box.createHorizontalBox();
top.add(topMainBox);
//top.setBounds(0,0,setWidth,200);
top.setMaximumSize(new Dimension(setWidth, 150));
// = new Vector();
dpanel = new DrawingPane();
optimizer = new Optimizer();
drawingArea = new Dimension(0,0);
font = new Font("SansSerif", Font.BOLD, 14);
dpanel.setFont(font);
//dpanel.setPreferredSize(new Dimension(setWidth, getHeight()
//panelGraphics = this.getGraphics();
//panelGraphics.setFont(font);
gp = new GraphicPoints();
dpanel.setLayout(null);
drawScroll = new JScrollPane(dpanel, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
drawScroll.setPreferredSize(new Dimension(setWidth, getHeight() - 50));
pane.setOrientation(JSplitPane.VERTICAL_SPLIT);
pane.setDividerLocation(150);
pane.setDividerSize(5);
pane.setLeftComponent(top);
pane.setRightComponent(drawScroll);
this.add(pane, BorderLayout.CENTER);
}
//start optimization of sql query
public String startOptimization(String sqlQuery) {
    String parseResult = optimizer.parseQuery(sqlQuery);
    if(parseResult.equalsIgnoreCase("ok")) {
        optTrees = optimizer.optimizeQuery();
        gp.setPoints();
        addQueryNodes();
        dpanel.revalidate();
        return 'parseResult;
    } else {
        return parseResult;
    }
}
public void clearOptimization() {
    dpanel.removeAll();
    if (optTrees != null)
        optTrees.removeAllElements();
    dpanel.repaint();
    gp.startPoint.setLocation(50, 50);
    gp.largestX = -1;
    gp.largestY = 50;
}

public void addQueryNodes() {
    /*Iterate through optimized query trees. For each tree, add each node to the drawing panel, if its sibling node will overlap it, then truncate the node*/
    Queue queue = new Queue();
    QueryTree tmpTree;
    BinNode sibling, node;
    RAOperation nodeData, siblingData;
    String predicate;
    Rectangle rec;
    boolean toolTip = false;
    boolean truncate = false;
    for (int i = 0; i < optTrees.size(); i++) {
        tmpTree = (QueryTree) optTrees.elementAt(i);
        BinTreeIterator iter = tmpTree.elements();
        while (iter.hasNext()) {
            node = (BinNode) iter.next();
            nodeData = (RAOperation) node.getData();
            //check if nodes with intersect
            checkTruncation(nodeData, (BinNode) iter.getSibling(), tmpTree.getHeight(), node.getDepth());
            predicate = nodeData.getNodeString();
            rec = nodeData.getRectangleBox();
            //JLabel label = new JLabel(predicate);
            //RALabel label = new RALabel(predicate,
            JLabel label = new JLabel(predicate);
            JLabel label = new JLabel(predicate);
            label.addMouseListener(new MyMouseListener(nodeData));
        }
    }
}
label.setFont(font);

//this.getGraphics().setColor(QUIGlobalData.getNodeColor());

//label.getComponentGraphics(this.getGraphics()).setColor(QUIGlobalData.getNodeColor());

//label.setBackgroundColor(QUIGlobalData.getNodeColor());

if(nodeData.isTruncated())
    label.setToolTipTextText(nodeData.getNodeString());

dpanel.add(label);
label.setBounds(rec);
dpanel.scrollRectToVisible(rec);
if(rec.getX() + rec.width >
    drawingArea.width)
    drawingArea.width = (int)rec.getX() +
    rec.width;
if(rec.getY() + rec.height >
    drawingArea.height)
    drawingArea.height = (int)rec.getY() +
    rec.height;
    dpanel.setPreferredSize(drawingArea);
    dpanel.revalidate();

//add some space after last node in last query tree
    drawingArea.setSize(drawingArea.width + 100,
    drawingArea.height + 100);
    dpanel.setPreferredSize(drawingArea);
}

//check if two nodes will intersect and need to be truncated
    private void checkTruncation(RAOperation oper, BinNode sibling, int int height, int depth) {
        /*For all RAoperations except Relations, check if sibling node
            *will intersect with operation, if so set each operation as truncated
            *and resize rectangle based on Pythagoreau theorem.
            */
        if(sibling != null) {
            RAOperation siblingData =
            (RAOperation)sibling.getData();
    }
Rectangle operRectangle, siblingRectangle;
operRectangle = oper.getRectangleBox();
siblingRectangle =
siblingData.getRectangleBox();
boolean intersects =
operRectangle.intersects(siblingRectangle);
boolean truncateSibling;
if(intersects) {
    double hypotenuse;
    Rectangle intersection =
    operRectangle.intersection(siblingRectangle);
    oper.setTruncated(true);
    siblingData.setTruncated(true);
    if(siblingRectangle.getWidth() >
        operRectangle.getWidth())
        truncateSibling = true;
    else
        truncateSibling = false;
    if(truncateSibling) {
        siblingRectangle.setRect(oper.getGetStartPoint().getX() +
        operRectangle.getWidth() + 10, siblingRectangle.getY(),
        siblingRectangle.getWidth() - intersection.getWidth(),
        siblingRectangle.getHeight());

        siblingData.getGetStartPoint().setLocation((int)siblingRectangl
        e.getX(),(int)siblingRectangle.getY());
    } else {

        operRectangle.setRect(operRectangle.getX(), operRectangle.get
        Y(), operRectangle.getWidth() - (intersection.getWidth() +
        10), operRectangle.getHeight());

        oper.getRightCornerPoint().setLocation((int)operRectangle.ge
        tX() + (int)operRectangle.getWidth(),
        (int)operRectangle.getY());
    }
    //int x, resizeX;
    /*If a node in the query tree has a
    sibling node, two adjacent right triangles
    *are implicitly formed. Using the
    Pythagorean theorem calculate the X axis of
    *both triangles, then resize the
    rectangle's of each node to be X in length from
*the node center point, thus eliminating overlap between the two node strings.

*hypotenuse formula (height of tree - depth of node) * halfinch in pixels

*/

/**
 hypotenuse = (height - depth) *

(72*0.5);

x = (int)Math.cos(45.0) *

(int)hypotenuse;

resizeX =
(int)oper.getCenterPoint().getX() + (x-5);

operRectangle.setRect(operRectangle.getX(),
operRectangle.getY(), (double)resizeX,
operRectangle.getHeight());

oper.getRightCornerPoint().setLocation(resizeX,
oper.getRightCornerPoint().getY());

    //do the same for the sibling node
    resizeX =
(int)siblingData.getCenterPoint().getX() - (x-5);

siblingRectangle.setRect((double)resizeX,
siblingRectangle.getY(), siblingRectangle.getWidth(),
siblingRectangle.getHeight());

siblingData.getStartPoint().setLocation(resizeX,
(int)siblingData.getStartPoint().getY());

*/

} } }

/*******************
 * This inner class listens for the mouse over and mouse click on nodes
 * within the query tree.
 *******************/

 private class MyMouseListener extends java.awt.event.MouseAdapter {

 RAOperation oper;

}
public MyMouseListener(RAOperation ra) {
    oper = ra;
}
// when a node in the query tree is clicked add a JTable
// with the Intermediate Relation statistics to the top of the gui
public void mouseClicked(MouseEvent e) {
    if (e.getClickCount() > 0) {
        IRTTableModel model = new IRTTableModel(oper.getIntermediateRelation());
        JTable irTable = new JTable(model);
        irTable.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);
        /*
         * TableColumnModel columnModel =
         * irTable.getColumnModel();
         * TableColumn colOne = columnModel.getColumn(0);
         * colOne.setWidth(150);
         * TableColumn colTwo = columnModel.getColumn(1);
         * colTwo.setWidth(200);
         * /
         * JScrollPane scroll = new JScrollPane(irTable);
         * scroll.setMaximumSize(new Dimension(250, 200));
         * Box irBox = Box.createVerticalBox();
         * irBox.add(scroll);
         * top.add(irBox);
         * top.revalidate();
    }
}
// when the mouse enters a node on the query tree change the
// mouse cursor to a hand cursor
public void mouseEntered(MouseEvent e) {
    setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
}
// when the mouse exits a node in query tree change mouse cursor to default
public void mouseExited(MouseEvent e) {
    setCursor(Cursor.getDefaultCursor());
}
private class DrawingPane extends JPanel {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        //draw lines connecting relational algebra expressions
        Rectangle rect;
        Class nodeClass;
        RARelation relation = new RARelation();
        BinNode node;
        RAOperation data=null;
        BinTreeIterator iter;
        Point sPoint, ePoint;
        QueryTree qTree;
        //g.setColor(Color.black);
        g.setColor(QUIGlobalData.getLineColor());
        g.setFont(new Font("SansSerif", Font.BOLD, 24));
        //loop through array of optimized trees, drawing a line between parent and children
        for(int i=0;i<optTrees.size();i++) {
            qTree = (QueryTree)optTrees.elementAt(i);
            iter = qTree.elements();
            while(iter.hasNext()) {
                node = (BinNode)iter.next();
                data = (RAOperation)node.getData();
                if(node.hasLeftChild() == true) {
                    sPoint = data.getCenterPoint();
                    ePoint = data.getLeftChildPoint();
                    //cast point methods as int
                    g.drawLine((int)sPoint.getX(), (int)sPoint.getY(), (int)ePoint.getX(), (int)ePoint.getY());
                }
                if(node.hasRightChild() == true) {
                    sPoint = data.getCenterPoint();
                }
            }
        }
    }
}
ePoint = data.getRightChildPoint();
g.drawLine((int)sPoint.getX(), (int)sPoint.getY(), (int)ePoint.getX(), (int)ePoint.getY());
*/
//if node is a relation draw oval around it
nodeClass = data.getClass();
if(nodeClass.isInstance(relation)) {
   rect = data.getRectangleBox();
g.drawOval((int)rect.getX(), (int)rect.getY(), (int)rect.getWidth(), (int)rect.getHeight);
}
} //end of while
this.revalidate();
} //end of class DrawingPane

/**
 * This inner class sets the graphical points of each node in a query tree, so they can be drawn by DrawingPane class.
 */
private class GraphicPoints {

   private Point orgPoint;
   private Point farRight = null;
   private double largestX = -1;
   private double largestY = 50;
   private Point startPoint;
   private double halfInch; // in pixels
   private int inch; // in pixels
   private int thrqtrInch; // in pixels
   private double hypotenuse;
   private int height; // height of a given tree
   private int fontSize;
   private QueryTree tmpTree;
   private FontMetrics fm;
/** Creates a new instance of GraphicPoints */
/*NOTE: When using the Point class you need to construct
a point with int
*but thereafter use double to set location
*/
public GraphicPoints() {
    orgPoint = new Point(50, 50);
    startPoint = new Point(50, 50);
    halfinch = InchesToPixels(0.5);
    inch = InchesToPixels(1.0);
    thrqtrInch = InchesToPixels(.75);
    hypotenuse = 0;
    tmpTree = null;
    fm = dpanel.getFontMetrics(QUI_Hopt.this.font);
    //gets the dimensions of strings painted in the Graphic context
}

//set the graphic points for each node
//might need to cast all doubles as int
public void setPoints() {
    int i, gap;
    //gap = 5;
    //QueryTree tmpTree;
    BinNode node = null;
    BinNode parent = null;
    RAOperation nodeData = null;
    RAOperation parentData = null;
    String predicate; //holds the predicate of each node
    in each tree
    Rectangle2D dim; //holds the dimensions of each
    predicate string to be printed
    Point tmpPoint;
    Point child;
    for(i = 0; i < optTrees.size(); i++) {
        tmpTree = (QueryTree) optTrees.elementAt(i);
        height = tmpTree.getHeight();
        //fontSize = tmpTree.getFontSize();
        setstartPoint();
        BinTreeItera}tor iter = tmpTree.elements();
        while(iter.hasNext()) {
            Point center = new Point(0, 0);
            Point leftCorner = new Point(0, 0);
            Point rightCorner = new Point(0, 0);
            node = (BinNode) iter.next();
            if(node.isRootNode() == false) {
parent = (BinNode)node.getParent();
parentData =
(RAOperation)parent.getData();
}
nodeData = (RAOperation)node.getData();
//get the dimensions of the predicate to be printed

//if(nodeData.getClass().getName().equalsIgnoreCase("wbqos.relAlg.RARelation"))
    // predicate =
nodeData.getTruncatedNodeString();
//else
    predicate = nodeData.getNodeString();
    //dim = fm.getStringBounds(predicate,
panelGraphics);
    dim = fm.getStringBounds(predicate,
QUI_Hopt.this.getGraphics());
    //get child point of parent node which is
    //the top center point of the predicate
    //then reset center to be the bottom center
    point of predicate
    if(node.isLeftChild() == true) {
        tmpPoint =
(Point)parentData.getLeftChildPoint();
        center.setLocation(tmpPoint.getX(),
(tmpPoint.getY() + dim.getHeight()));
    } else {
        if(node.isRootNode() == false) {
            tmpPoint =
(Point)parentData.getRightChildPoint();
            center.setLocation(tmpPoint.getX(),
(tmpPoint.getY() + dim.getHeight()));
        } else //this is root node
            center.setLocation(startPoint.getX() + 
(dim.getWidth() / 2.0), startPoint.getY() + 
dim.getHeight());
    }
    if(node.isRootNode() == true) //this is root
node
    nodeData.setStartPoint(startPoint);
else {
leftCorner setLocation(center.getX() - (dim.getWidth() / 2.0), center.getY() - dim.getHeight());
    nodeData.setStartPoint(leftCorner);
}
    //reset the frame of rectangle bounding the predicate based on startPoint, width and height

nodeData.setRectangleBox((int)dim.getWidth(), (int)dim.getHeight());
    nodeData.setCenterPoint(center);
    //if node has a right child it must have a left child
    if(node.hasRightChild() == true)
        nodeData.setRightChildPoint(getChildPoint(1, center, node.getDepth(), false));
            if(node.hasLeftChild() == true)
        nodeData.setLeftChildPoint(getChildPoint(0, center, node.getDepth(), node.hasRightChild()));

rightCorner setLocation(center.getX() + (dim.getWidth() / 2.0), center.getY());
    nodeData.setRightCornerPoint(rightCorner);
    //update largestX and largestY
    if(largestX > 0) {
        //this is not the first node of first
        tree
            if(rightCorner.getX() > largestX)
                largestX = rightCorner.getX();
            if(rightCorner.getY() > largestY)
                largestY = rightCorner.getY();
    } else {
        if(rightCorner.getX() > largestX)
            largestX = rightCorner.getX();
        largestY = rightCorner.getY();
    }
} //end of while loop
} //end of for

    //returns the point of child node
    public Point getChildPoint(int childType, Point c, int depth, boolean rightChild) {
        double x, y;
Point child = new Point(0, 0);
hypotenuse = (height - depth) * halfinch;
//hypotenuse = inch;
y = Math.sin(45.0) * hypotenuse;
x = Math.cos(45.0) * hypotenuse;
if(childType == 0) {
    //its a left child point
    if(rightChild == true)
        child.setLocation(c.getX() - x, c.getY() + y);
    else {
        //this is an only child, so set directly
        below its parent node a distance of halfinch
        child.setLocation(c.getX(), c.getY() + inch);
    }
} else
    child.setLocation(c.getX() + x, c.getY() + y);

return child;

//returns the startPoint point for tree passed in
public void setStartPoint() {
    //check if current startPoint point will position
tree so no overlapping of trees occurr
    double x, y, diff;
    int i,tmpWidth,largestWidth;
    largestWidth = -1;
    int leftMargin = InchesToPixels(.25);
    String predicate;
    Rectangle2D strDim;
    /*Calculate the distance of base of the triangle
    formed by the query tree.
    *Calculate the width of each node. Pick a starting
    point and substrack from
    *it the greater of the two values. If the result
    is less than the left margin
    *of the screen, then readjust the starting point.
    */
    QueryTreeIterator iter =
tmpTree.queryTreeElements();
    while(iter.hasNext()) {
        BinNode node = (BinNode)iter.next();
        RAOperation oper = (RAOperation)node.getData();
predicate = oper.getNodeString();
strDim = fm.getStringBounds(predicate, panelGraphics);
strDim = fm.getStringBounds(predicate, QUI_Hopt.this.getGraphics());
tmpWidth = (int)strDim.getWidth() / 2;
if(tmpWidth > largestWidth)
largestWidth = tmpWidth;
if(node.isFarLeft()) {
    if(node.hasRightChild() &&
    node.hasLeftChild())
        hypotenuse = hypotenuse + (halfinch *
    node.getDepth());
}

//add space for predicates in hypotenuse
//hypotenuse += fontSize * height;
y = Math.sin(45.0) * hypotenuse;
x = Math.cos(45.0) * hypotenuse;
if(largestWidth > x)
x = largestWidth;

//this is the first tree of a row, place start point so that it does not go
//beyond the left margin.
if(largestX < 0) {
diff = orgPoint.getX() - x;
    if(diff < leftMargin) {
        //the farleft point of tree is off the
        left side of screen
        startPoint.setLocation(startPoint.getX() + Math.abs(diff), startPoint.getY());
        orgPoint.setLocation(startPoint.getX(), orgPoint.getY());
    }
}
else {
    if((largestX + (x + halfinch) >
    screenWidth)) {
        startPoint.setLocation(orgPoint.getX(),
largestY + inch);
        //largestX = startPoint.getX();
largestX = -1;
diff = startPoint.getX() - x;
    if(diff < leftMargin)
startPoint.setLocation(startPoint.getX() + Math.abs(diff),
startPoint.getY());
}
else {
    // not at end of screen
    startPoint.setLocation(largestX + (x + halfinchest), startPoint.getY());
    //no need to set Y coordinate; its in proper position
}
};//end of loop
} ///comment inches to pixels--using 72 dpi
public int InchesToPixels(double inch) {
    double tmp = 72*inch;
    int result = (int)tmp;
    return result;
}
};//end of GraphicPoints
};//end of QUI

/*
 *Programmer: Edwin Waite
 *Date: 03/22/04
 *Title: QUI Main Page
 *Description: This is the main applet for WBQOS. It displays a
 *JTabbedPane with the different components of WBQOS on separate tabs.
 */

package wbqos.gui;
import javax.swing.*;
import java.awt.*;

public class QUI_Pane extends JApplet {
    private JTabbedPane tabs;
    private QUI_Query queryInterface;
    private QUI_Hopt hOpt;
    /** Creates a new instance of QUI_Pane */
    public QUI_Pane() {
    }
    //applet initialization
public void init() {
    QUIGlobalData.setServer(getParameter("server"));
    QUIGlobalData.setScreenWidth(Integer.parseInt(getParameter("width")));
    QUIGlobalData.setScreenHeight(Integer.parseInt(getParameter("height")));
    QUIGlobalData.setDatabase(getParameter("initdb"));
    tabs = new JTabbedPane();
    hOpt = new QUI_Hopt();
    queryInterface = new QUI_Query();
    queryInterface.setHOptimizer(hOpt);
    tabs.add("SQL", queryInterface);
    tabs.add("Heuristic Optimizer", hOpt);
    this.getContentPane().setLayout(new BorderLayout());
    this.getContentPane().add(tabs, BorderLayout.CENTER);
}

package wbqos.gui;
import java.awt.*;

public class QUIGlobalData {

    private static int screenWidth;
    private static int screenHeight;
    private static String database;
    private static String nodeColor;
    private static String lineColor;
    private static String dbserver;

    /** Creates a new instance of QUIGlobalData */
    public QUIGlobalData() {

//set the database server
public static void setServer(String ip) {
    dbserver = ip;
}

//get the database server
public static String getServer() {
    return dbserver;
}

//set screen (that the applet is viewed in) width
public static void setScreenWidth(int w) {
    screenWidth = w;
}

//get screen width
public static int getScreenWidth() {
    return screenWidth;
}

//set screen height
public static void setScreenHeight(int h) {
    screenHeight = h;
}

//get screen height
public static int getScreenHeight() {
    return screenHeight;
}

//set database name that the applet is pointing at
public static void setDatabase(String db) {
    database = db;
}

//get database the applet is pointing at
public static String getDatabase() {
    return database;
}

//set node color
public static void setNodeColor(String s) {
    nodeColor = s;
}

//set the line color
public static void setLineColor(String s) {
    lineColor = s;
}

//get the node color
public static Color getNodeColor() {
    return getColor(nodeColor);
}
//get the line color
public static Color getLineColor() {  
    return getColor(lineColor);
}

private static Color getColor(String c) {  
    if(c.equalsIgnoreCase("black"))  
        return Color.black;
    else if(c.equalsIgnoreCase("red"))  
        return Color.red;
    else if(c.equalsIgnoreCase("blue"))  
        return Color.blue;
    else if(c.equalsIgnoreCase("green"))  
        return Color.green;
    else if(c.equalsIgnoreCase("magenta"))  
        return Color.magenta;
    else if(c.equalsIgnoreCase("yellow"))  
        return Color.yellow;
    else if(c.equalsIgnoreCase("cyan"))  
        return Color.cyan;
    else
        return Color.black;
}

/*
*Programmer: Edwin Waite
*Date: 4/05/04
*Title: Intermediate Relation Table Model
*Description: This table model holds the data
*for an intermediate relation within the query tree.
*/

package wbqos.gui;
import wbqos.relAlg.*;

public class IRTTableModel extends javax.swing.table.AbstractTableModel {  

    private RARelation ir;
    /** Creates a new instance of IRTTableModel */
    public IRTTableModel(RARelation r) {  
        ir = r;
    }
    //get column count
    public int getColumnCount() {

//there will always be two columns, a
//label column and data column
return 2;
}

//get column name
public String getColumn_name(int c) {
    if(c == 0)
        return "Label";
    else if(c == 1)
        return "Data";
    else
        return "";
}

//get row count
public int getRow_count() {
    //hard coded for the amount
    //of data in a Intermediate Relation
    return 5;
}

//get value at specific cell
public Object getValueAt(int r, int c) {
    if(r == 0 && c == 0)
        return "Name";
    else if(r == 1 && c == 0)
        return "Attributes";
    else if(r == 2 && c == 0)
        return "Record Count";
    else if(r == 3 && c == 0)
        return "Record Size";
    else if(r == 4 && c == 0)
        return "Column Count";
    else if(r == 0 && c == 1)
        return ir.getName();
    else if(r == 1 && c == 1)
        return ir.getAttributeString();
    else if(r == 2 && c == 1)
        return ir.getStringNumRows();
    else if(r == 3 && c == 1)
        return ir.getStringRecordSize();
    else if(r == 4 && c == 1)
        return ir.getStringNumColumns();
    else
        return null;
}
package wbqos.gui;
import wbqos.relAlg.*;

class IRTableModel extends javax.swing.table.AbstractTableModel {

    private RARelation ir;
    /** Creates a new instance of IRTableModel */
    public IRTableModel(RARelation r) {
        ir = r;
    }

    //get column count
    public int getColumnCount() {
        //there will always be two columns, a
        //label column and data column
        return 2;
    }

    //get column name
    public String getColumnName(int c) {
        if(c == 0)
            return "Label";
        else if(c == 1)
            return "Data";
        else
            return "";
    }

    //get row count
    public int getRowCount() {
        //hard coded for the amount
        //of data in a Intermediate Relation
        return 5;
    }

    //get value at specific cell
    public Object getValueAt(int r, int c) {

}
if(r == 0 && c == 0)
    return "Name";
else if(r == 1 && c == 0)
    return "Attributes";
else if(r == 2 && c == 0)
    return "Record Count";
else if(r == 3 && c == 0)
    return "Record Size";
else if(r == 4 && c == 0)
    return "Column Count";
else if(r == 0 && c == 1)
    return ir.getName();
else if(r == 1 && c == 1)
    return ir.getAttributeString();
else if(r == 2 && c == 1)
    return ir.getStringNumRows();
else if(r == 3 && c == 1)
    return ir.getStringRecordSize();
else if(r == 4 && c == 1)
    return ir.getStringNumColumns();
else
    return null;
REFERENCES


