WinRDBI: A Windows-based Relational Database Educational Tool

Suzanne W. Dietrich, Eric Eckert and Kevin Piscator
Arizona State University
Department of Computer Science and Engineering
Tempe, AZ 85287-5406

dietrich@asu.edu

Abstract

RDBI is an educational tool that provides students with the capability to test their understanding of the formal relational query languages (relational algebra, domain relational calculus and tuple relational calculus) and the industry standard query language SQL. Although RDBI is an integral part of the database management courses at a number of universities, it is unavailable to those universities that do not have a license for the software product in which RDBI is implemented. WinRDBI, a version of RDBI for Windows, removes this limitation by increasing the availability of the educational tool to a standard Windows platform. Another advantage of WinRDBI is its graphical user interface, providing the student with a more intuitive interface than the command line interface of RDBI. This paper describes the architecture and user interface of WinRDBI. The features of WinRDBI are also illustrated using nontrivial examples from a popular database text. Although formal relational query languages do not provide inherent support for aggregation, these examples illustrate how to write queries in the formal languages to support (a limited form of) counting and minimum/maximum queries.

Introduction

Relational databases are the focus of most introductory courses on database management systems. The formal relational query languages (relational algebra, domain relational calculus and tuple relational calculus) are therefore an important part of the curriculum. As a student, it is difficult to know whether your queries expressed on paper in the formal languages are correct. As an instructor, it is often difficult to grade creative queries, especially those that have not been verified by an educational tool. The goal of RDBI, the Relational DataBase Interpreter, is to provide a mechanism by which the students can explore the formal relational query languages, getting immediate feedback by seeing the answers to the posed query. Consequently, the grading process is usually eased (somewhat) since the students can submit verified answers on homework assignments.

RDBI developed over a number of years as independent study projects by students exploring the connection between databases and logic programming. RDBI is written in Quintus Prolog. The database and meta-data are stored as Prolog facts. A query in each of the query languages, including the industry standard query language SQL, is parsed and converted into Prolog code, which is then executed to evaluate the query. SQL was added to RDBI to increase its usability as an educational tool by providing an SQL interpreter for those universities that don't have the use of a relational database product with SQL. It also provides students with the capability to pose SQL queries over the same database schema and instance that they used for exploring the formal query languages.

RDBI is an integral part of the introduction to database management course at Arizona State University. RDBI is used in a sequence of assignments. In the first assignment, students define the relational schema, populate the database, and answer queries over the database in relational algebra. The subsequent assignments on relational calculus (both domain and tuple) and SQL reuse the database created and populated in the first assignment. Some of the queries change in the subsequent assignments, especially for SQL where queries are included to reinforce the concepts of aggregation and grouping in SQL.

The use of RDBI as an educational tool for relational query languages is described in [1]. Although RDBI is used at a number of universities, it is unavailable to those universities that do not have Quintus Prolog. WinRDBI, a version of RDBI for Windows, removes this limitation by increasing the availability of the educational tool to a standard Windows platform. WinRDBI is therefore available to any university running Windows and to any student for use on their own personal computer. Another advantage of WinRDBI is its graphical user interface, providing the student with a more intuitive interface than the command line interface of RDBI.

This paper describes the architecture and user interface of WinRDBI. The features of WinRDBI are also illustrated using nontrivial examples from a popular database text. Although formal relational query languages do not provide inherent support for aggregation, these examples illustrate how to write queries in the formal languages to support (a limited form of) counting and minimum/maximum queries.

Architecture

The architecture of WinRDBI, shown in Figure 1, not only provides for the increased availability of the educational tool but it also increases the maintainability of the application. The user interface and the Prolog engine of the relational database interpreter are implemented as separate components with a well-defined interface.

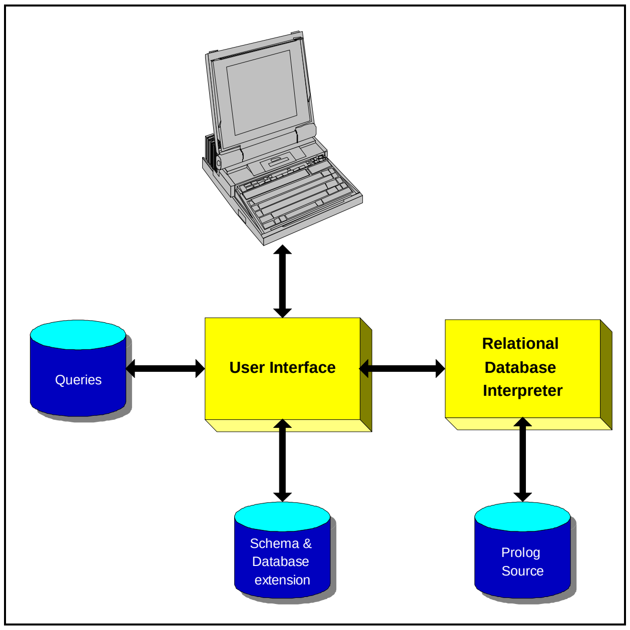


Figure 1 WinRDBI Architecture

The user interface component is written in Visual Basic 4.0, which is a well-known language for the development of graphical user interface applications. The user interface is compiled into a standard Windows executable that uses the services of a Prolog interpreter library provided as a Windows Dynamic Link Library (DLL). The user interface component also interacts with the file system, storing the queries, schema and database extension.

The relational database component is written in Amzi! Prolog 3.3, which allows the distribution of their DLLs free of charge for non-commercial applications, such as WinRDBI. The Relational Database Interpreter component also interacts with the file system upon initialization of WinRDBI by reading the Prolog source that implements the interpreter.

User Interface

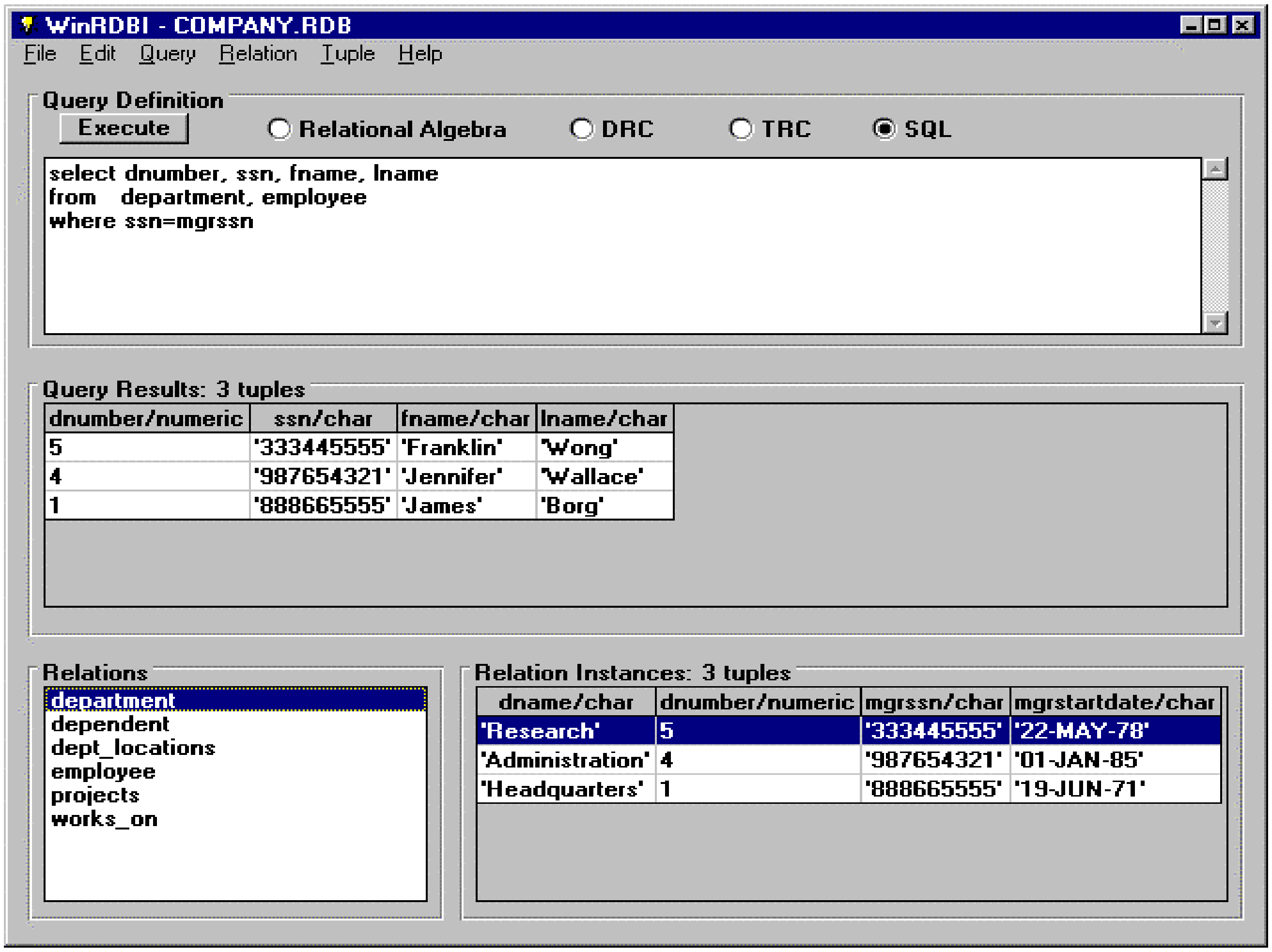
The primary WinRDBI window is shown in Figure 2. The main components are: Query Definition, Query Results, Relations and Relation Instances. The size of the window may be adjusted according to the user's preferences. The current database filename is shown in the title bar. A database file has an extension ‘RDB’ and consists of a collection of relation definitions along with their instance. The RDB file is an ASCII file of Prolog facts, where the meta-data is stored as ‘base_table’ facts and a relation instance is stored as facts given by the relation name.

The Query Definition component is used for defining the query in any of the four supported query languages. The language choice is made by selecting one of the four option buttons labeled ‘Relational Algebra’, ‘DRC’ (Domain Relational Calculus), ‘TRC’ (Tuple Relational Calculus), or ‘SQL’ (Structured Query Language). The query can be edited using the normal Windows keystrokes, and the normal Copy, Cut and Paste functions are also supported. The ‘Execute’ button causes the current query to be evaluated. The results of the query are displayed in the Query Results component. The query may use the assignment syntax (:=) to create a new relation, which is materialized with the results of the query.

The Query Results component displays the tuples that are defined by the relational expression in the Query Definition component. The schema and count of the tuples is also displayed. The user can scroll the results horizontally or vertically but may not change the data.

The Relations component lists all of the relations that are defined in the current session. This list is vertically scrollable. The user can display the instance of a relation by selecting a relation from this list with a single mouse click. The user can display the definition of a relation by selecting a relation with a double mouse click.

The Relation Instances component shows the count, schema and tuples for the relation selected in the Relations component. The user can scroll the relation instance horizontally or vertically but may not change the data directly.



( ) Relational Algebra ( ) DRC ( ) TRC [✓ SQL]

```
select dnumber, ssn, fname, lname
from   department, employee
where ssn=mgrssn
```

Query Results: 3 tuples

dnumber/numeric	ssn/char	fname/char	lname/char
5	'333445555'	'Franklin'	'Wong'
4	'987654321'	'Jennifer'	'Wallace'
1	'888665555'	'James'	'Borg'

Relations

- department
- dependent
- dept_locations
- employee
- projects
- works_on

Relation Instances: 3 tuples

dname/char	dnumber/numeric	mgrssn/char	mgrstartdate/char
'Research'	5	'333445555'	'22-MAY-78'
'Administration'	4	'987654321'	'01-JAN-85'
'Headquarters'	1	'888665555'	'19-JUN-71'

Figure 2 WinRDBI User Interface

Menus

WinRDBI provides File, Edit, Query, Relation, Tuple and Help menus.

The File menu provides functions to create a new database schema and to open and save databases. This menu also provides the functionality to open and save queries. No changes made during the user's session are permanent until a save is performed. The Print menu item provides the capability to selectively print relations. There is an Exit menu item to exit the program.

The Edit menu provides the standard Windows features of Cut, Copy and Paste. These features work with the Query Definition component.

The Query menu provides functions to execute the current query (the same as the 'Execute' button in the Query Definition component), and to save the current query as a relation in the schema. In effect, this action causes the current query to be materialized as a relation. A Clear menu item is also available to quickly clear out the Query Definition component.

The Relation menu provides functions to show (and subsequently edit) the definition of the relation selected in the Relation component, create a new relation and delete the selected relations. The Show Definition menu item pops up the Relation Editor window, which displays the relation name, its attributes and its key. The Relation Editor is used in both editing and creating a relation. When displaying an existing definition, the only operation allowed is the renaming of attributes via the Rename Attribute button, since tuples may already exist for the relation. When creating a relation, the user selects the Add Attribute button to specify the name and type (character or numeric) of each attribute. The user provides the key for the relation by specifying a comma-separated list of attributes. There is a Delete Attribute button that is only available to delete an attribute during the specification of the relation definition. Once defined, an attribute of a relation can not be deleted. The Delete menu item on the

Relation menu deletes a selected relation and its instance. A warning precedes any deletion.

The Tuple menu provides functions to insert and delete tuples of a relation. A tuple may be changed by first deleting it and then adding a new one. A tuple may be inserted into a relation by selecting the relation in the Relation component and then selecting the Insert menu item of the Tuple menu. A pop-up window appears in which to insert the attribute values, which are separated by commas. Since the database is stored as Prolog facts, values for attributes follow the Prolog convention, where numeric attribute values are Prolog numeric constants and character attribute values are quoted strings. The insert pop-up window remains open to facilitate inserting multiple tuples until the user selects the Done button. A tuple may be deleted from a relation by selecting the tuple in the Relation Instances component and then selecting the Delete menu item of the Tuple menu. Multiple tuples can be deleted by selecting contiguous tuples using the mouse. A warning precedes any deletion.

The Help menu provides abbreviated information on how to use the program and the syntax of the four query languages. An 'About Box' is also provided.

Examples

Since our introductory database management class is based on the popular Elmasri and Navathe database text [2], the WinRDBI distribution provides the company database as a detailed example. The RDB file contains the database schema (Figure 6.5 page 145) and instance (Figure 6.6 on page 146 with the exception that one tuple is added to the works_on table so that Query 3 gives a non-empty result). The seven queries from the text (Section 6.7 pages 170-172) are answered in each of the query languages. Therefore, there are four query files (with filename extensions: ALG, DRC, TRC, SQL), each containing seven queries.

A query file stores the definitions of queries in an ASCII file with each query separated by a semicolon. A query file is created using WinRDBI through the 'Save Query' and 'Save Query As' menu items on the File menu. As each query is verified, the user selects the 'Save Query As' option to save each query separately. To create a query file that is a sequence of verified queries, the user opens each query and then saves the query as an existing file, which appends the query to that query file. When a query (file) is later opened, each query is evaluated. To facilitate a query sequence, the user should use the assignment syntax provided by WinRDBI to save the query result to a materialized relation. The user can then display or print the results of the materialized query. A query file may be edited by hand.

This section provides some examples of WinRDBI's query languages using the company database enterprise. Rather than illustrate simple examples, the following uses nontrivial examples that illustrate how to support (a limited form of) counting and minimum/maximum queries in the formal relational query languages, which do not provide inherent support for aggregation.

Consider an example query, which is Query 5 from [2], that lists the names of employees with two or more dependents. As the textbook points out, relational algebra does not provide inherent support for aggregation. However, the text provides a counting solution to this query in a relational algebra that has been extended to include aggregate functions. Figure 3 shows the WinRDBI query files that provide a solution to this query without counting and assuming that an employee does not have two dependents with the same name. The relational algebra solution creates two copies of a table relating an employee ssn with the name of their dependents. These relations are then used in a Cartesian product followed by a selection to find those employees that have greater than one dependent. This intermediate result (emps_gtone_dep) is then (natural) joined with employee to display the names. The domain relational calculus and tuple relational calculus solutions are similar, declaratively specifying employees that have (at least) two dependents with different names. The SQL solution uses count to create a materialized relation having the name and schema specified to the left of the assignment. The intermediate result (dep_num) is then used to select those employees having at least 2 dependents and to join with employee to display the answer to the query.

Consider another example query that finds the employees earning the maximum salary in each department. Figure 4 shows the WinRDBI query files that provide a solution to this query. The relational algebra solution creates two copies of the department salaries to use in a Cartesian product to select out those salaries that are not the maximum, i.e., there is a salary greater than it. This intermediate result (not_maxsal) is then used in a difference with all department salaries to find the maximum department salaries. A (natural) join is performed to find the employee names. The domain and tuple relational calculus solutions are again similar, declaratively specifying employees such that there does not exist another employee in the same department having a higher salary. The SQL solution uses the max aggregate function to create an intermediate result that determines the maximum salary in each department. The intermediate

```
%                 RELATIONAL ALGEBRA
empdep1(essn1, depname1) :=
   project essn, dependent_name (dependent);
empdep2(essn2, depname2) := empdep1;
emps_gtone_dep(ssn) := project essn1
   (select (essn1=essn2) and (depname1<>depname2)
      (empdep1 product empdep2));
alg5 := project lname, fname (employee njoin emps_gtone_dep);
```

```
%           DOMAIN RELATIONAL CALCULUS
drc5 :=  {LNAME,FNAME | (exists SSN)
   (employee(FNAME,_,LNAME,SSN,_,_,_,_,_,_) and
   (exists DEP1,DEP2)(dependent(SSN,DEP1,_,_,_) and
       dependent(SSN,DEP2,_,_,_) and DEP1 <> DEP2))};
```

```
%           TUPLE RELATIONAL CALCULUS
trc5 := {E.lname, E.fname | employee(E) and
   (exists D1,D2) (dependent(D1) and dependent(D2) and
       D1.essn=E.ssn and D2.essn=E.ssn and
       D1.dependent_name<>D2.dependent_name)};
```

```
%                          SQL
dep_num(ssn,depcnt) :=
   select ssn, count(*)
   from employee,dependent
   where ssn = essn
   group by ssn;
sql5 :=
   select lname,fname
   from   employee e,dep_num d
   where  e.ssn = d.ssn and
          depcnt >= 2;
```

Figure 3 Employees with 2 or more dependents

```
%                 RELATIONAL ALGEBRA
deptsal1(dno1,salary1) := project dno,salary (employee);
deptsal2(dno2,salary2) :=deptsal1;
not_maxsal := project dno1,salary1
   (select dno1=dno2 and salary1<salary2
      (deptsal1 product deptsal2));
alg_deptmaxsal(dno,salary) := deptsal1 difference not_maxsal;
alg_maxdeptsal := project fname,lname,salary,dno
   (employee njoin alg_deptmaxsal);
```

```
%           DOMAIN RELATIONAL CALCULUS
drc_maxdeptsal :=
   {FNAME, LNAME, SALARY, DNO |
      employee(FNAME, _, LNAME, _,_,_,_,SALARY,_,DNO)
      and not (exists S)
      (employee(_,_,_,_,_,_,_,S,_,DNO) and S>SALARY)};
```

```
%           TUPLE RELATIONAL CALCULUS
trc_maxdeptsal :=
   {E.fname, E.lname, E.salary, E.dno | employee(E) and
   not (exists T)
   (employee(T) and T.dno=E.dno and T.salary > E.salary)};
```

```
%                          SQL
sql_deptmaxsal(dno,maxsal) :=
   select  dno, max(salary)
   from employee
   group by dno;
sql_maxdeptsal :=
   select  fname, lname, m.dno, maxsal
   from employee e, sql_deptmaxsal m
   where e.salary = m.maxsal and e.dno=m.dno;
```

Figure 4 Maximum Department Salary Employees

result (sql_deptmaxsal) is then used to select those employees in that department that have the maximum salary.

The nontrivial examples used in this section provided an overview of the WinRDBI query languages and illustrated how to support (a limited form of) counting and minimum/maximum queries in the formal languages.

Summary

We hope that the effort to create WinRDBI by porting RDBI to Amzi! Prolog and integrating it with a graphical user interface written in Visual Basic will increase the availability of this educational tool for relational query languages.

At the time of its first release and the writing of this paper, WinRDBI had not yet been tested by creative student solutions in a class environment. However, WinRDBI had been tested on RDBI assignment solutions from the past four semesters. We expect future refinements of WinRDBI after using it in the classroom and receiving constructive feedback. We invite you to visit the RDBI and WinRDBI web page (http://www.eas.asu.edu/~cse412/rdbi.html) for additional information on these educational tools and any future enhancements.

Acknowledgments

Acknowledgments are obviously due to the authors of RDBI: Chien-Ho Ho, Ana Hun, Changguan Fan, and Sarah Simons. Thanks to Michael Tjahjadi and Anamika Chauhan as teaching assistants for the class that assisted in generating WinRDBI distribution and class materials.

References

1. Dietrich, S. W. An Educational Tool for Relational Query Languages. *Computer Science Education.* Vol 4, 1993, pp. 157-184.
2. Elmasri, R. and Navathe, S. B. *Fundamentals of Database Systems (2nd ed.).* Benjamin/Cummings, CA, 1994.