WinRDBI

Introduction

WinRDBI is a teaching tool for use in the introductory database course at Arizona State University. It allows the user to interactively experiment with Relational Algebra, Domain Relational Calculus, Tuple Relational Calculus, and Structured Query Language expressions. WinRDBI interacts with a Prolog engine which performs the actual database manipulations.

WinRDBI is based on RDBI, which was written over a number of years by students as independent study projects. RDBI was written in Quintus Prolog and ran under UNIX or VMS. Using RDBI, students interacted with a command-line oriented user interface. This required students to use on-campus facilities, either on-site or via dial-up, to access the program. Additionally, RDBI could only be used on systems that were licensed to run Quintus Prolog. Further information and a description of how to use the original RDBI tool is described in [1].

Later, Version 1.0 of WinRDBI was written. Its interface and use is documented in [2]. In addition to providing a simpler, more intuitive Graphical User-Interface (GUI) interface with which students could interact, WinRDBI gave students who have IBM compatible computers at home the ability to run RDBI on their own computers. To help accomplish that goal, WinRDBI used the Amzi! Prolog Logic Server rather that Quintus Prolog. The Amzi! product was chosen because it has no runtime license restrictions for non-commercial use.

Version 1.0 of WinRDBI was used for several semesters at ASU. Over that time, several shortcomings were found. For example, the user could interact with only one query at a time. Once a query was completed, it had to be saved to a file before a new query could be started. The query was then no longer visible from within the user interface. If the user wanted to review a previously completed query, a separate editor or viewer program had to be used. This made it very difficult to construct the answer to a question that involved multiple sub-queries. Another related shortcoming was that any comments that were entered into the user interface as part of a query were lost when the query was saved to disk.

Relative to the database, Version 1.0 provided very minimal editing capabilities. Tuples could be inserted and deleted, but not edited. In order to change a field in a tuple, the entire tuple had to be deleted and re-entered. The only other option was to save the database as a text file of Prolog facts, call up a text editor to change the appropriate data, then reload the database into the user interface. Further, any relations that were materialized as part of a query became part of the database. After a query had been executed, it was not possible to save the original database without first deleting these extra relations.

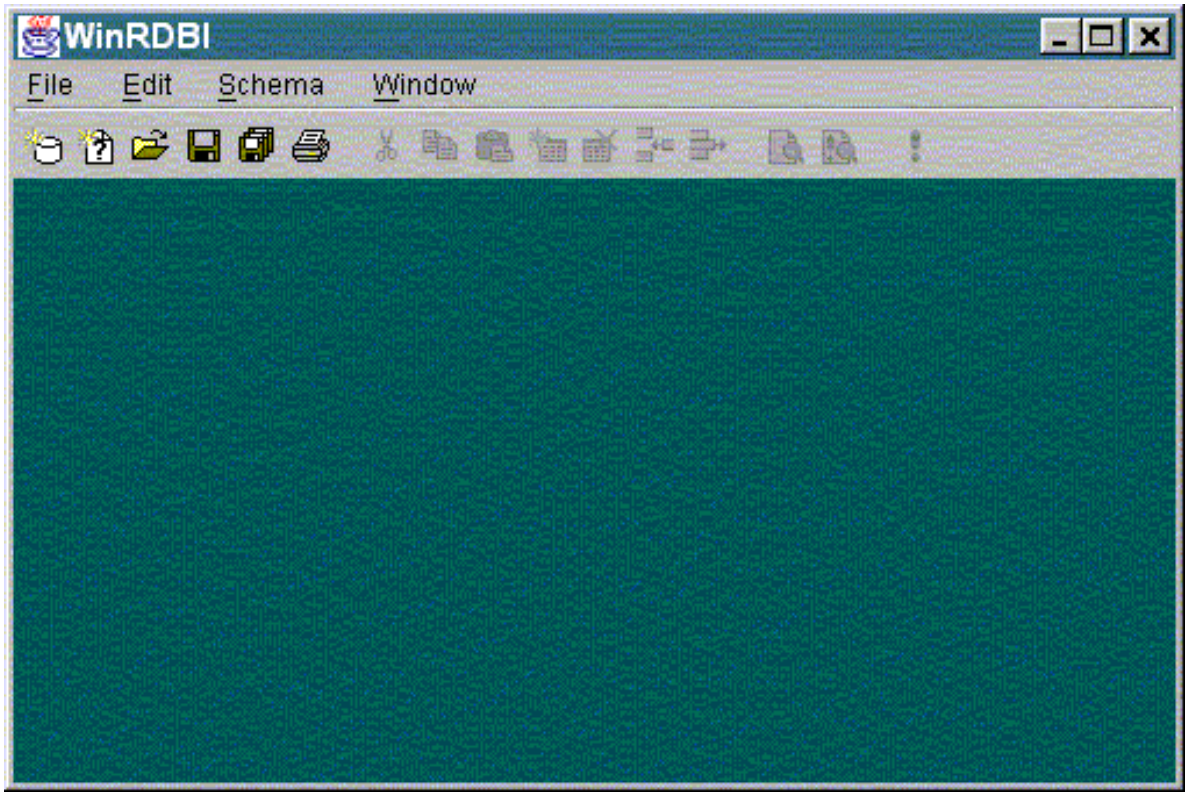
These limitations gave rise to the current version, WinRDBI 2.0. Version 2.0 is a complete redesign of the user interface of Version 1.0. Version 2.0 is written in Java and uses the Java Foundation Classes (JFC), or Swing, to provide the user interface. Version 2.0 still interacts with the Amzi! Prolog Logic Server to perform the database operations, but the user interface, described in the next section, is entirely different.

User Interface

WinRDBI 2.0 was designed as a Multiple Document Interface (MDI) application. In the general sense, a MDI application allows you to have more than one "document" open at a time, for example, more than one word-processor document or spreadsheet. In the case of WinRDBI, a document may be a database or a query file. While WinRDBI allows you to have multiple query files open simultaneously, only one database file may be open at a time.

Components

The image below shows the initial WinRDBI screen. At this point, you may open an existing query or database or create new ones using the File menu or the toolbar.

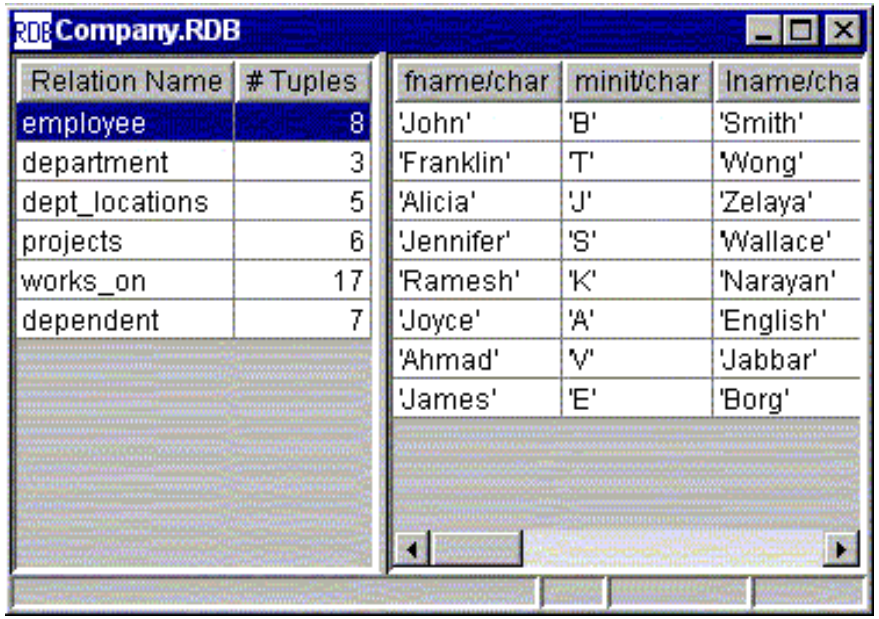


The main screen has the normal components of a Microsoft Windows application: a menu

and a toolbar

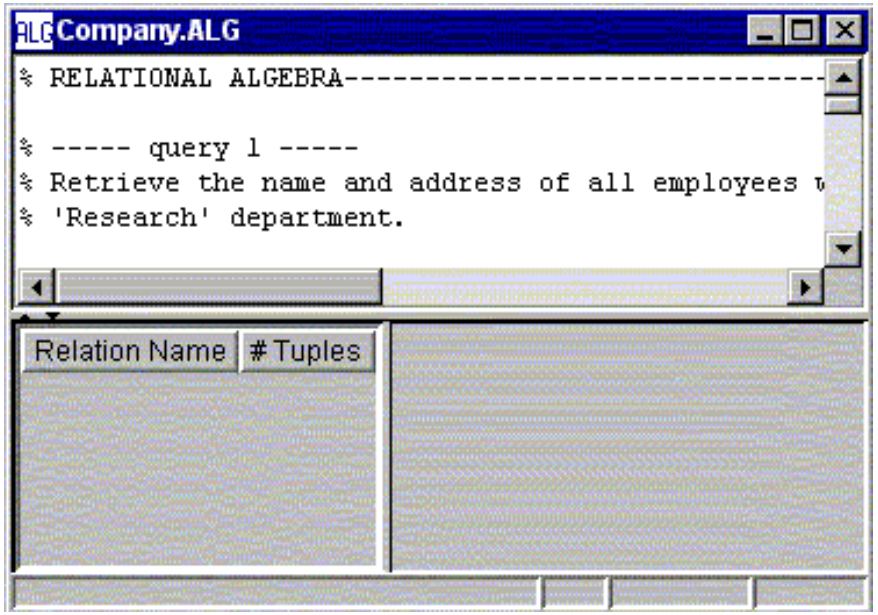
In the following sections, these components will be described in some detail.

There are two additional components which will be mentioned regularly. The first is called a SchemaPane. A SchemaPane is an internal window on the main screen that is used to contain the schema, or database, against which the queries will be executed. The figure below shows a SchemaPane.



Notice that the pane is vertically divided in two. On the left is a list of the relation names and the number of tuples in the relation. On the right is a list of the tuples in the relation. In a SchemaPane, you can add, delete, and edit relations and tuples. There may be only one SchemaPane open at a time.

The second component of which you need to be aware is called a QueryPane. A QueryPane is used to enter queries and display the results of the queries. A QueryPane is shown below.



Obviously, a QueryPane is divided into three sections. The top section is a very basic text editor and is used for entering queries in one of the four query languages. The bottom section, which is similar to a SchemaPane, is used to display the results of the queries. Note that the relations and tuples representing query results cannot be edited.

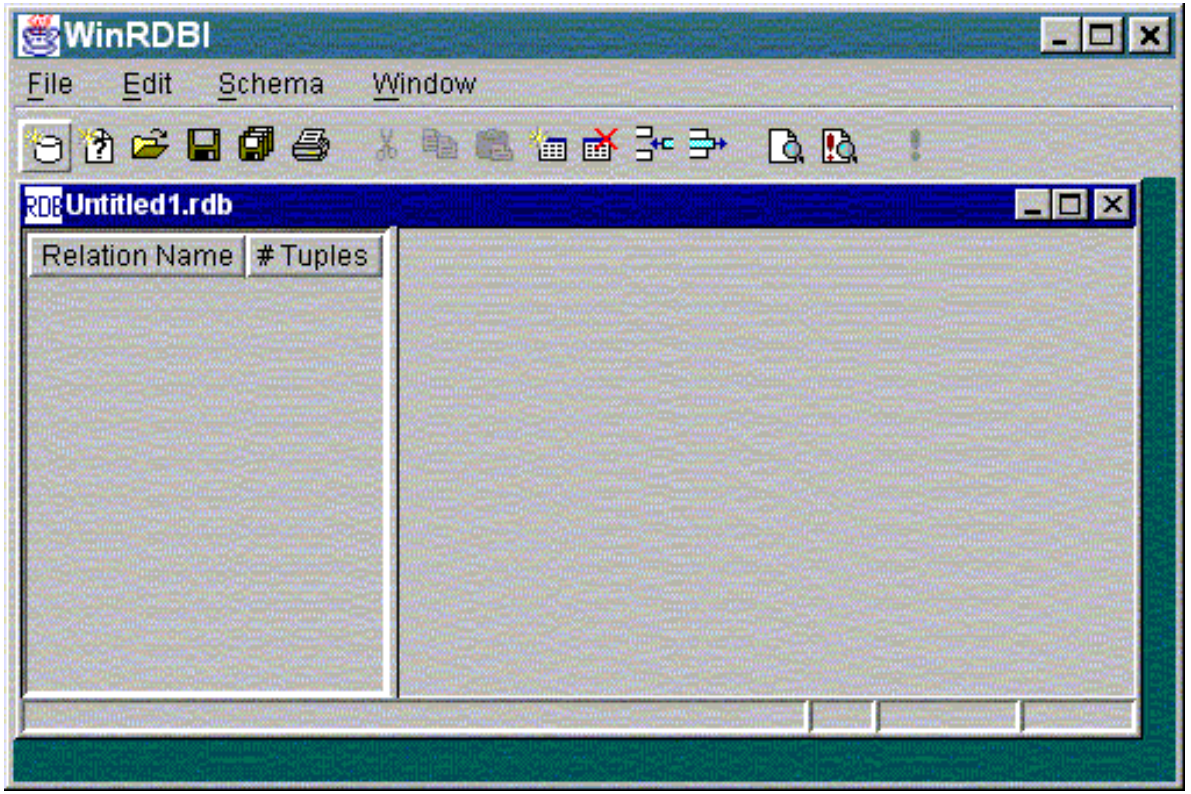
Menu Structure

This section describes the available menu items in WinRDBI 2.0.

File Menu

New Database

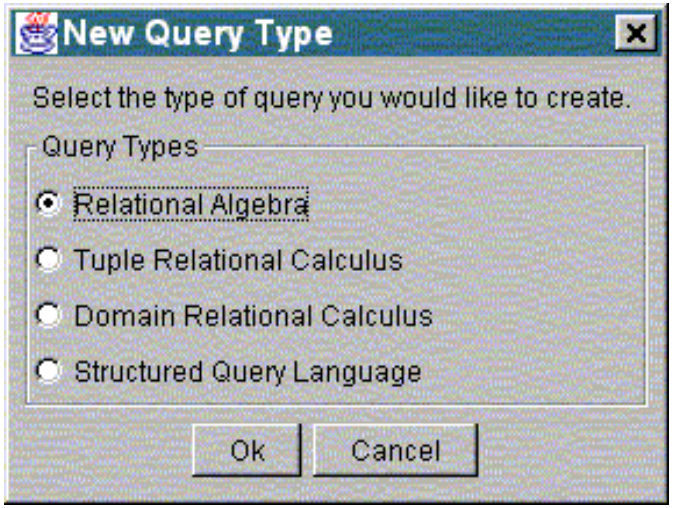
The New Database menu item allows you to create a new RDBI database. When you select this option, an empty SchemaPane will be created as shown below.



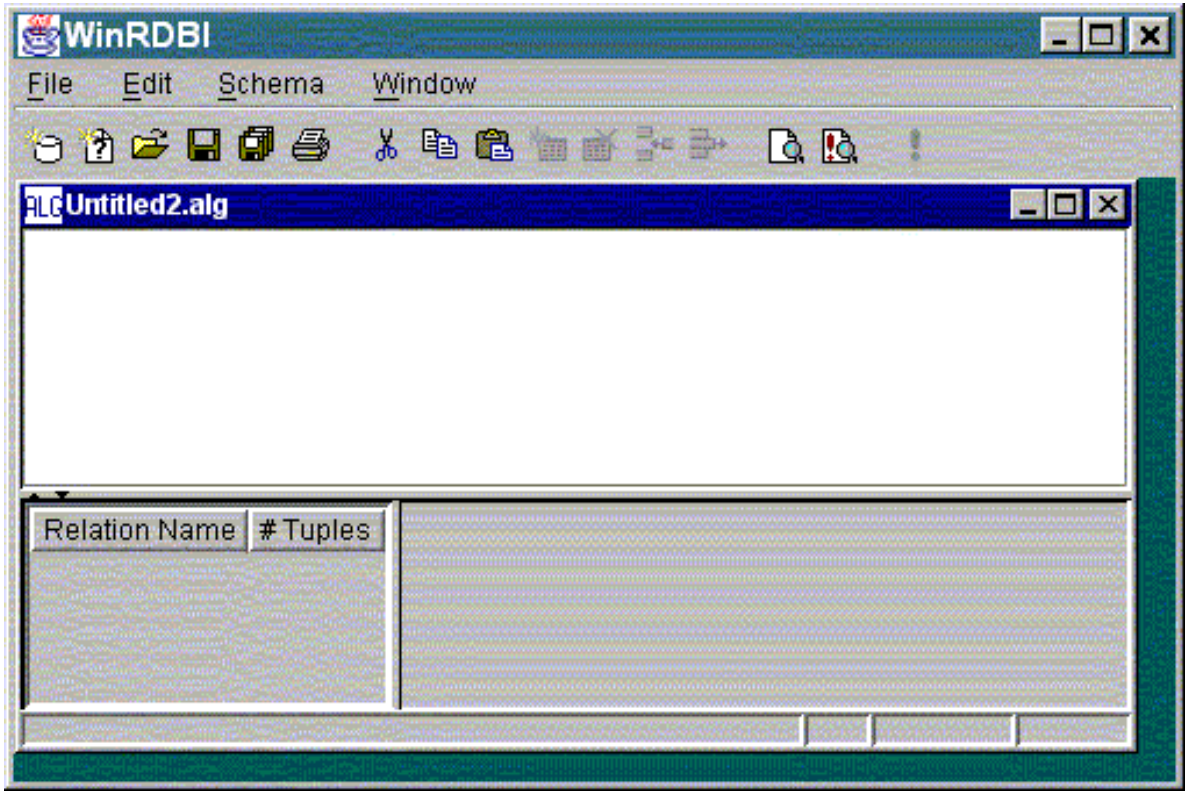
At this point, you can use various toolbar icons or items on the Schema menu to add relations and tuples to your database.

New Query

When you select the New Query item, the dialog below will be displayed, asking you what kind of queries you want to enter: Relational Algebra, Domain Relational Calculus, Tuple Relational Calculus, or Structured Query Language. The default is Relational Algebra. You must select one of the options to proceed.

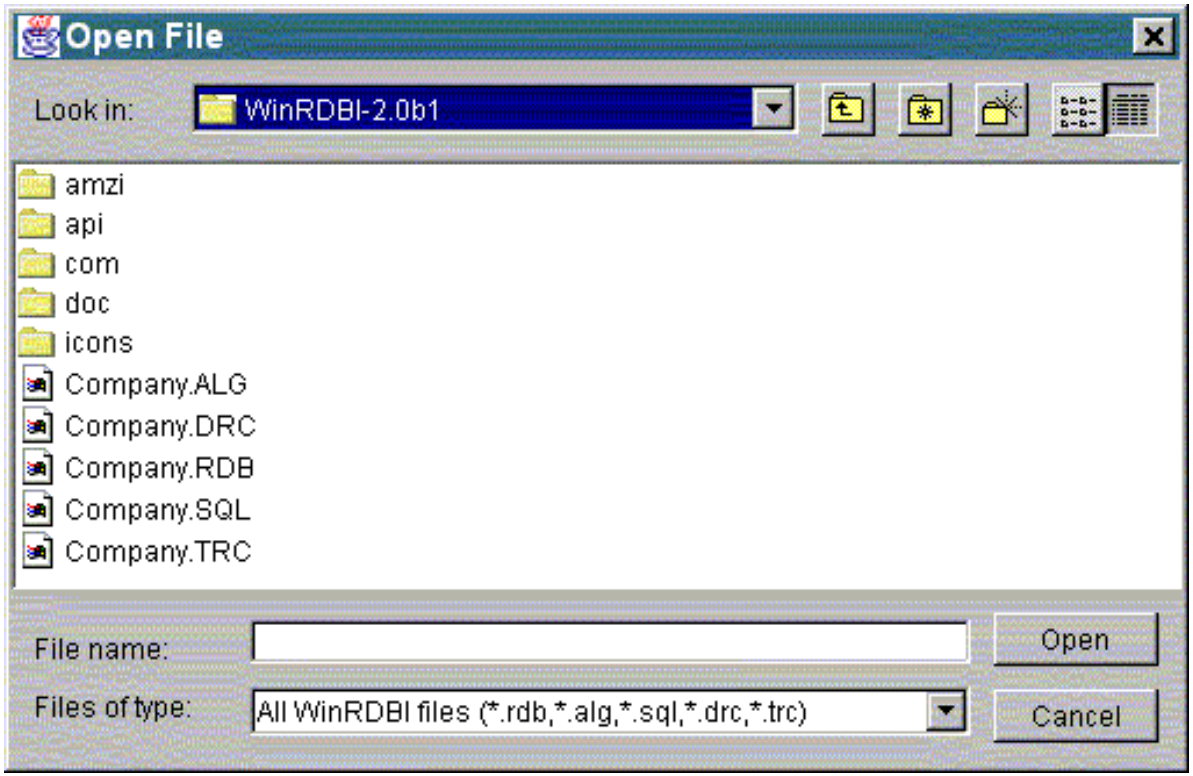


Once you have selected the appropriate type of queries, you will be given an empty QueryPane, as shown below.



Open...

If you choose "Open..." from the File menu, you will be shown the following dialog from which you can choose a query or database file to open.



Close

Choosing this item will close the currently active window. If there are unsaved changes, you will be offered a chance to save the changes.

Save

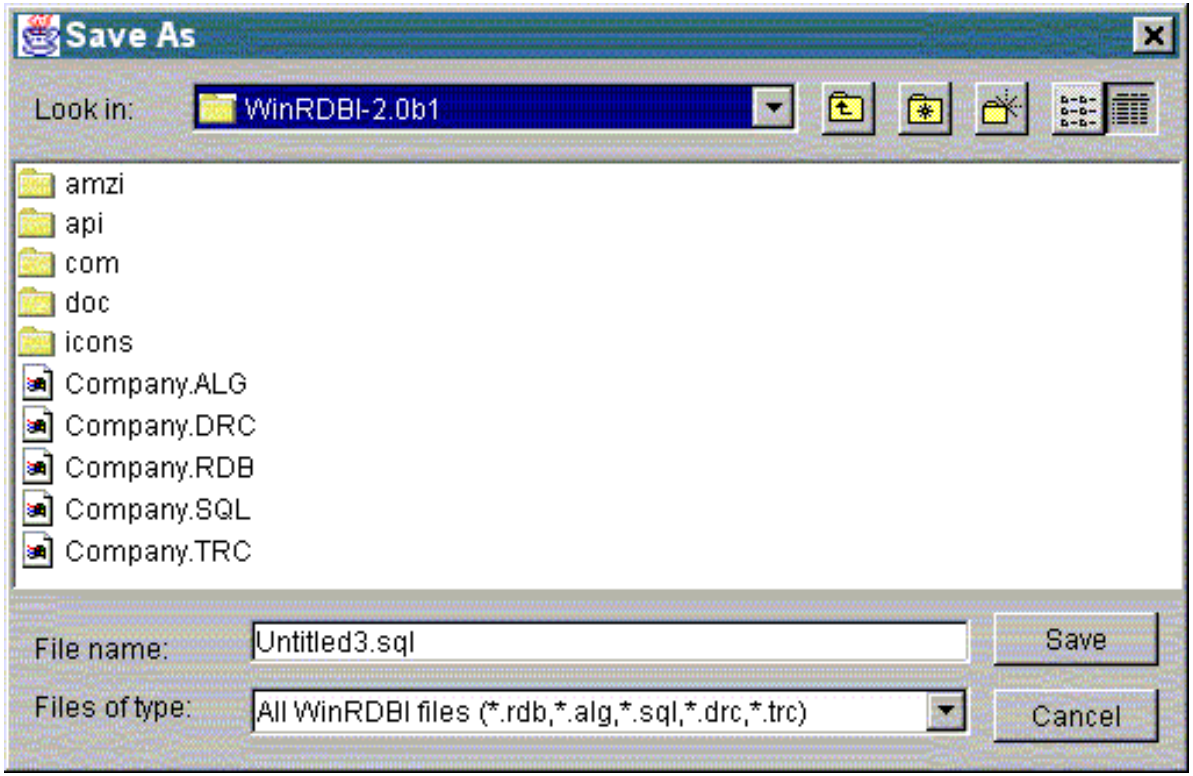
Save the contents of the currently active QueryPane or SchemaPane.

Save All

Save the contents of all open panes.

Save As...

The Save As item allows you to save the active file under a new name. If you choose this item, you will see a dialog box similar to the one below.



You would enter a new file name in the field labeled "File name:". In the figure above, this is the field that contains "Untitled1.sql". If you enter the name of a file that already exists, you will be asked if you want to overwrite the existing file. If you answer no, you will be given the opportunity to enter another file name. If you indicate that you *do* want to overwrite the existing file, then the save action will proceed normally.

Print...

Choosing the Print item will allow you to print queries, query results, or relations from the database based on the active pane. If one or more relations are selected in the results or the database, you will be given the option to print only the selected relations or all relations. Multiple non-adjacent relations may be selected by holding down the control key on the keyboard while clicking with the left mouse button. An entire region may be selected by left-clicking on the first relation, then holding down the shift key while left-clicking another relation.

Control-shift-left-click can be used to select non-adjacent regions.

Exit

Choosing "Exit" from the file menu will close all open windows and exit the program. If there are unsaved changes, you will be given the opportunity to save them.

Edit Menu

The Edit Menu contains the normal clipboard operations Cut, Copy, and Paste, along with Search and Replace operations. The clipboard operations are only available when you are editing queries or when you are editing a specific field in a tuple.

Cut

The Cut item allows you to copy selected text from a query editor or field editor to the clipboard. The selected text is then deleted from the editor.

Copy

The Copy item performs essentially the same function as the Cut item, but the selected text is ***not*** deleted from the editor.

Paste

Choosing the Paste item copies text *from* the clipboard *to* the query or field editor.

Find...

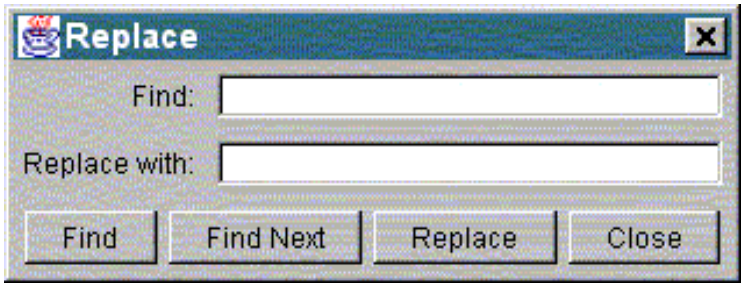
Select the Find item to search for text in the currently active Pane. The pane can be either a QueryPane or a SchemaPane. When you select the Find item, you will see the following dialog box:

Enter the text you wish to find in the field. To initiate the search, click on the "Find" button, or press enter. If the text is found, it will be selected. If it is not found, a dialog box will be displayed indicating that no matching text

was found. To find subsequent occurrences, click on the "Find Next" button. Click on the "Close" button to dismiss the dialog box.

Replace...

The Replace item allows you to search the active Pane for a string and replace it with another string. Like the Find item, Replace can be used in either a QueryPane or a SchemaPane. Note, however, that the Replace option *does not* work in the results portion of a QueryPane, as the result table is not editable. The dialog box looks like

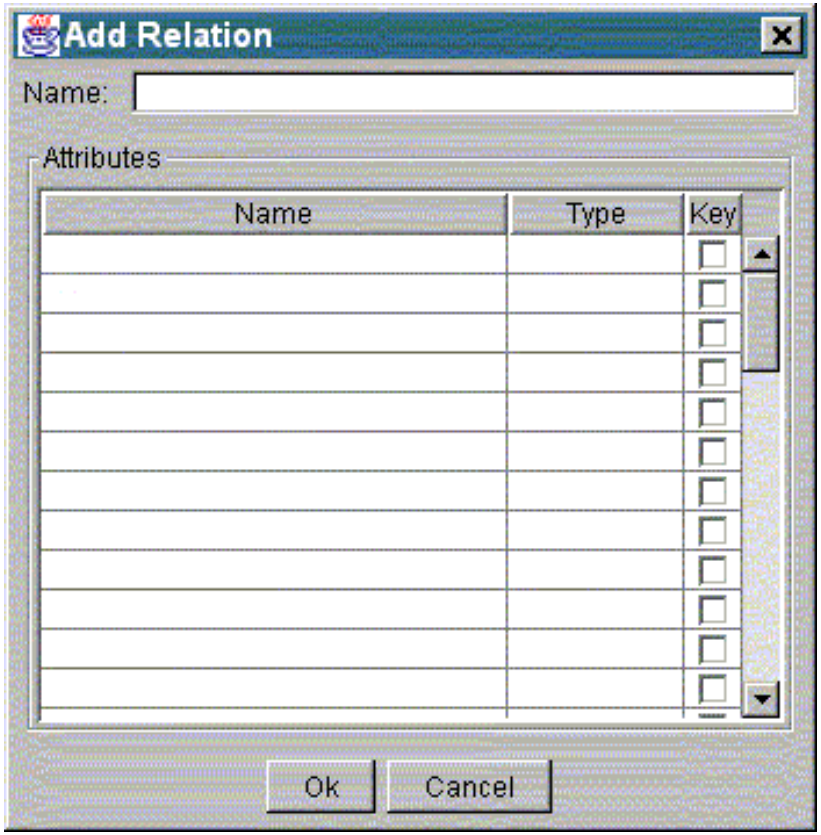


Like the Find dialog, enter the text for which you wish to search in the field labeled "Find:". Then enter the replacement text in the field labeled "Replace with:" and click the "Find" button. If the text is found, it will be highlighted. To replace the text, click on the "Replace" button. Clicking on "Replace" will cause the highlighted text to be replaced with the replacement text and will automatically search for the next occurrence of the search string. If you don't wish to replace an occurrence, just click on "Find Next". Of course, click the "Close" button to dismiss the dialog.

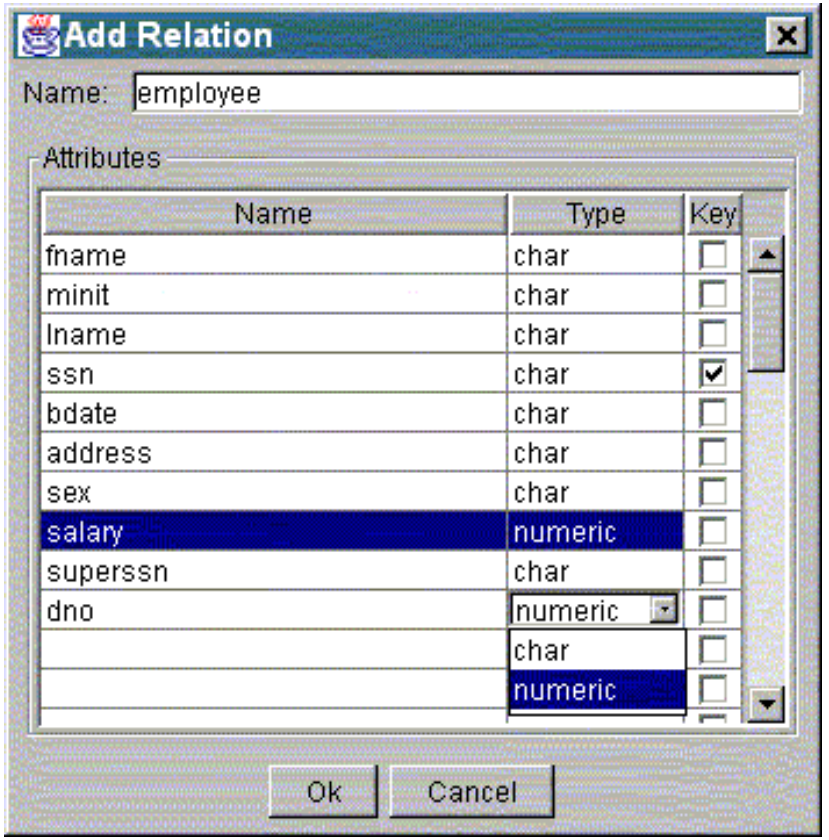
Schema Menu

Add Relation

Choose the Add Relation item to add a new relation to a database. This option is only available when a SchemaPane is active. When you choose this item, the following dialog will be displayed



The relation name should be entered in the field labeled "Name:", while the attributes should be entered in the table below. Note that relation and attribute names must be identifiers starting with a lower case letter. If an attribute is part of the key, put a check in the "Key" column by clicking on the appropriate box. Here's a more complete example.



Note that the attribute named "ssn" is the key attribute. You can create a composite key by checking more that one box. The "Type" column allows only two entries, char and numeric, which are selected from the drop-down list.

Copy Relation to Database

When a QueryPane is active, this item is available. It allows you to copy the selected relation from the query results to the SchemaPane. In this way, you can save and/or edit the results of a query. This option is only available when a QueryPane is active.

Delete Relation

The Delete Relation item deletes the selected relations from the database. This option is only available when a SchemaPane is active.

Edit Relation

Select the Edit Relation option from the Schema menu to bring up a dialog box which will allow you to edit the name and attributes of a relation. This dialog box is the same as the "Add Dialog" box. The only restriction is that you may not change the type of an attribute. You may also activate the dialog box by double-clicking the left mouse button on the relation name. This option is only available when a SchemaPane is active.

Add Tuple

The Add Tuple item adds one or more tuples to the currently selected relation. If there are multiple tuples selected, then the number of tuples added will be equal to the number of tuples selected. The new tuples will always be added to the end of the table. This option is only available when a SchemaPane is active.

Delete Tuple

Selecting the Delete Tuple item causes the selected tuples to be deleted. This option is only available when a SchemaPane is active.

Insert Tuple

Choose the Insert Tuple item to insert one of more tuples into the currently selected relation. If there are multiple tuples selected, then the number of tuples inserted will be equal to the number of tuples selected. The new tuples will be inserted above the first selected tuple. This option is only available when a SchemaPane is active.

Execute Queries

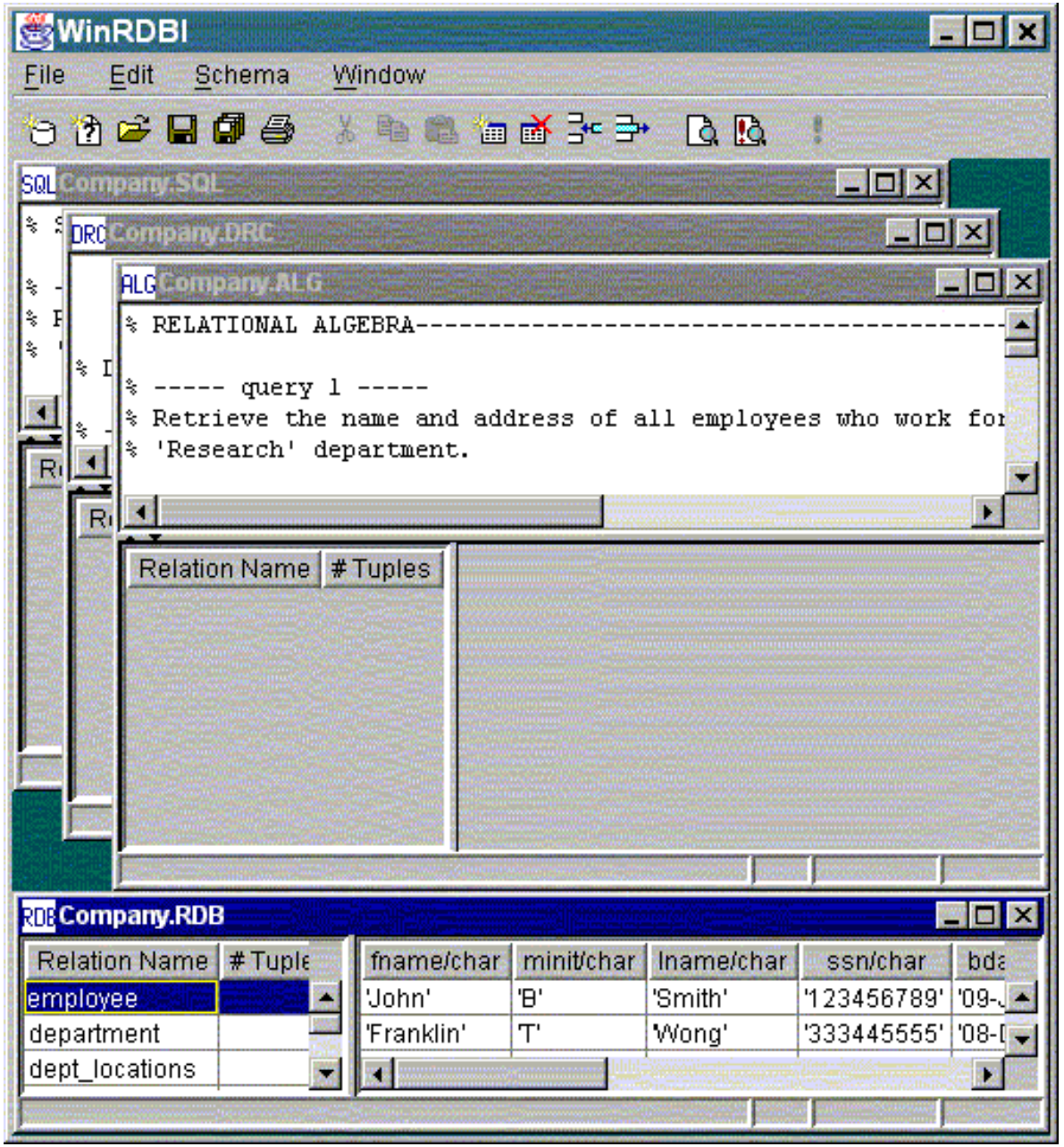
Execute the queries in the currently active QueryPane. This item is available only when a QueryPane is active *and* a database is loaded. The results will be shows in the table in the bottom half of the QueryPane.

Window Menu

The Window menu will contain the names of all open panes. If a pane is hidden, you can bring it to the front by selecting the corresponding item in the Window menu.

Arrange

In addition to an item corresponding to each pane, the Window menu contains an Arrange item. The Arrange item arranges all open panes by tiling the QueryPanes in the upper 3/4 of the main screen and puts the SchemaPane, if one is open, in the lower 1/4 of the main screen. The end result will look something like the figure below.


WinRDBI
File
Edit
Schema
Window
Company.SQL
Company.DRC
Company.ALG
% RELATIONAL ALGEBRA----------
% ----- query 1 -----
% Retrieve the name and address of all employees who work for
% 'Research' department.
Relation Name
Tuples
Company.RDB
Relation Name
Tuple
employee
department
dept_locations
fname/char
minit/char
lname/char
ssn/char
'John'
'B'
'Smith'
'123456789'
'Franklin'
'T'
'Wong'
'333445555'

Toolbar

The following table describes the action performed by each toolbar icon and lists the equivalent menu operation. The notation A → B means to select item B from menu A. More specifically, File → Open means to select the Open item from the File menu.

Icon	Description	Menu Equivalent	Keyboard Shortcut
	Create a new, empty SchemaPane.	File → New Database	Ctrl+D
	Create a new, empty QueryPane.	File → New Query	Ctrl+Q
	Open an existing file.	File → Open	Ctrl+O
	Save the contents of the currently active pane.	File → Save	Ctrl+S
	Save the contents of all open panes.	File → Save All	
	Print the currently active pane.	File → Print	Ctrl+P
	Cut the selected text to the clipboard.	Edit → Cut	Ctrl+X
	Copy the selected text to the clipboard.	Edit → Copy	Ctrl+C
	Paste the clipboard contents to the current location.	Edit → Paste	Ctrl+V
	Add a relation to the current database.	Schema → Add Relation	
	Delete the selected relations from the current database.	Schema → Delete Relation	
	Insert one or more tuples into the current relation.	Schema → Insert Tuple	
	Delete the selected tuples from the current relation.	Schema → Delete Tuple	
	Search for text in the active QueryPane or SchemaPane.	Edit → Find	Ctrl+F
	Replace text in the active QueryPane or SchemaPane.	Edit → Replace	Ctrl+R
	Execute the queries in the active QueryPane.	Schema → Execute Queries	Ctrl+E

Languages by Example

In this section, we will investigate the solutions to two example queries over the Company database example described in [3] to illustrate some of the features of the languages. The languages are described in [1] and the BNF for the languages is given as part of this User's Guide. The WinRDBI languages are case sensitive, with the exception of SQL. Relation and attribute names must be identifiers starting with a lower case letter. Constants in all languages and in the database are either (single) quoted strings or numeric constants. Variables (in DRC and TRC) are identifiers that start with an upper case letter.

Note that you can experiment with the solutions given below by loading the Company.RDB file into WinRDBI, creating a new QueryPane of the appropriate type, cutting and pasting the code into the editor portion of the QueryPane, and executing the queries. These queries are also part of the example called Company distributed with WinRDBI.

Example 1

The first example we will discuss is a very simple one that queries the Company database to find out which employees work for the department named 'Research'. The result contains the first name, last name, and address of each employee.

Relational Algebra

The relational algebra solution creates two temporary relations to solve this problem. The first relation will have a single tuple which contains the department number of the Research department. This information is necessary since only the department number, not the name, is kept in the employee record. The second temporary relation is formed by taking the Cartesian product of the first temporary relation with the employee relation, and selecting only those tuples in which the employee's department number (dno) is equal to the Research department's number (dnumber). This yields a relation that contains only those tuples for employees that work in the Research department. Finally, the attributes fname, lname and address are projected out of the second temporary relation to display only the requested information.

```
research_dept:=select dname='Research' (department);
research_dept_emps:=select dnumber=dno (research_dept product employee);
alg1:=project fname, lname, address (research_dept_emps);
```

Domain Relational Calculus

DRC is a declarative language. The solution below specifies the set of tuples for which there exists a department number DNO for a department named 'Research' and employees whose home department is the same as DNO.

```
drc1:= {FNAME, LNAME, ADDRESS |
    (exists DNO)
        (department('Research',DNO,_,_) and
             employee(FNAME,_,LNAME,_,_,ADDRESS,_,_,_,DNO))};
```

Tuple Relational Calculus

The TRC solution is very similar to the DRC solution in that it declares the employees whose home department number is the same as the Research department's number.

```
trc1:= {T.fname, T.lname, T.address |
    employee(T) and
    (exists D)
        (department (D) and D.dname='Research' and D.dnumber=T.dno)};
```

Structured Query Language

The SQL solution simply extracts the first name, last name, and address tuples from the employee relation where the employee's department number is the same as the Research department's department number.

```
sql1 := select fname,lname,address
        from employee, department
        where dname = 'Research' and dnumber = dno;
```

Example 2

The second example we will examine asks which employees in the company database have two or more dependents. Below are the solutions in all four languages supported by WinRDBI.

Relational Algebra

As pointed out in [3], relational algebra does not support aggregate functions such as max, min, or count. However, it is possible to solve this problem in relational algebra if we assume that no employee has more than one dependent with the same name. Under that assumption, a solution can be constructed as follows:

1. Create a relation where each tuple contains an employee's social security and the name of one of the

employee's dependents.

2. Make a second copy of the relation created in step 1.
3. Take the Cartesian product of the relations created in steps 1 and 2, and extract tuples in which the employee social security numbers are the same but the dependent names are different.
4. Obtain the names of the employees having more than one dependent by creating the natural join of the employee relation and the relation from step 3 and projecting out the employee's first and last names.

Here's the code:

```
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

The DRC declarative solution given below declares that there exists an employee who has two dependents whose names are not equal.

```
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

The TRC solution is very similar to the DRC solution. Like DRC, TRC is a declarative language. The solution to the query reads exactly like the DRC solution.

```
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)};
```

Structured Query Language

The SQL solution makes use of SQL's built-in count function to count the number of dependents of an employee, which is stored in an intermediate relation. The intermediate relation is then used in conjunction with the employee relation to display the names of the employees with more that one dependent.

```
dep_num(ssn,depent) := 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
               depent >= 2;
```

Language Assumptions and BNF

In the following grammar, terminal symbols are in **boldface**, non-terminal symbols are in angle brackets (<>), patterns are in *italics*, and braces ({}) indicate optional items. Note that WinRDBI requires relation and attribute names to begin with a lower case letter, while variables must begin with an upper case letter. Constants must be (single) quoted strings or numeric constants.

<Query_Definition>	→	<Query> **;**
	|	<Assignment_Statement> **;**
<Assignment_Statement>	→	*Relation_Name* **:=** <Query>
	|	*Relation_Name* **(** <Attribute_List> **) :=** <Query>
<Attribute_List>	→	*Attribute_Name* **,** <Attribute_List>
	|	*Attribute_Name*

Relational Algebra

The relational algebra is a procedural language with the following assumptions regarding its operators:

- The union, difference and intersect operators require that the operand relations are compatible, i.e., the operand relations have identical schemas.
- The (Cartesian) product operator requires that the operand relations have disjoint attribute names.
- The njoin (natural join) operator does ***NOT*** require common attribute names in the operand relations; if the

operand relations have disjoint attribute names, then the njoin results in a Cartesian product.

- The θ-join and division operators are purposely not provided by the interpreter so that students investigate the equivalent definitions of these operators in terms of the fundamental relational algebra operators.

```
    <Query> → <Expression>
<Expression> → Identifier
             | ( <Expression> )
             | <Select_Expr>
             | <Project_Expr>
             | <Binary_Expr>
<Select_Expr> → select <Condition> ( <Expression> )
<Project_Expr> → project <Attributes> ( <Expression> )
<Binary_Expr> → ( <Expression> ) <Binary_Op> ( <Expression> )
  <Condition> → <And_Condition>
              | <And_Condition> or <Condition>
<And_Condition> → <Rel_Formula>
                | <Rel_Formula> and <And_Condition>
<Rel_Formula> → <Operand> <Relational_Op> <Operand>
              | ( <Condition> )
  <Binary_Op> → union
              | njoin
              | product
              | difference
              | intersect
  <Attributes> → Identifier
               | Identifier, <Attributes>
    <Operand> → Identifier
              | Constant
<Relational_Op> → =
                | >
                | <
                | <>
                | >=
                | <=
```

Domain Relational Calculus

The domain relational calculus (DRC) language is a declarative, positional language where variables range over the domain of attributes. Note that constants and anonymous variables (_) may be used within a DRC expression. For example, consider the query that returns the names and salaries of the female employees earning more than 35000:

```
{ LName, FName, Salary |
    employee(FName,_,LName,_,_,_,'F',Salary,_,_) and Salary > 35000};
```

The interpreter assumes that

- the names of the attributes in the resulting schema is a lowercase version of the variable names, which must be an identifier starting with an uppercase letter. In the above example, the result schema has the attributes lname, fname, and salary.
- the query expression is safe, assuming the left-to-right evaluation of Prolog. The above query expression is safe, however, the following logically equivalent version of the example is not allowed:

  ```
  { LName, FName, Salary |
      Salary > 35000 and employee(FName,_,LName,_,_,_,'F',Salary,_,_) };
  ```

  The rule of thumb for writing safe expressions is to make sure that a variable is bound to a value before it is used, either in a comparison or within a negation.

<Query> → { <Variables> | <Formula> }
<Formula> → <And_Formula>
| <And_Formula> **or** <Formula>
<And_Formula> → <Sub_Formula>
| <Sub_Formula> **and** <And_Formula>
<Sub_Formula> → *Identifier* (<Vars_Cons>)
| (<Formula>)
| **not** <Sub_Formula>
| <Condition>
| <Quantified_Formula>
<Quantified_Formula> → <Quantifier> (<Formula>)
| <Quantifier> <Quantified_Formula>
<Quantifier> → (**exists** <Variables>)
| (**forall** <Variables>)
<Condition> → <Operand> <Relational_Op> <Operand>
<Operand> → *Identifier*
| *Constant*
<Variables> → *Identifier*
| *Identifier*, <Variables>
<Vars_Cons> → *Identifier*
| *Identifier*, <Vars_Cons>

	Constant
	Constant , <Vars_Cons>
<Relational_Op> →	=
	>
	<
	<>
	>=
	<=

Tuple Relational Calculus

The tuple relational calculus (TRC) language is a declarative language similar to DRC except that variables range over tuples and attributes are referenced using dot notation. The interpreter only evaluates safe TRC expressions. In TRC, this means that the tuple variable must be bound to its associated relation before the values of its attributes can be accessed and before the variable is used within a negation. Here is the TRC version of the query that returns the names and salaries of the female employees earning more than 35000:

```
{ E.lname, E.fname, E.salary |
    employee(E) and E.sex='F' and E.salary>35000 };
```

The only additional assumption for TRC is a shorthand notation for selecting all attributes associated with a tuple variable. For example, the following shows the result of the above query when all employee attributes are returned:

```
{ E | employee(E) and E.sex='F' and E.salary>35000 };
```

<Query>	→	{ <Vars_Attrs> | <Formula> }
<Formula>	→	<And_Formula>
	|	<And_Formula> **or** <Formula>
<And_Formula>	→	<Sub_Formula>
	|	<Sub_Formula> **and** <And_Formula>
<Sub_Formula>	→	*Identifier* (*Variable*)
	|	(<Formula>)
	|	**not** <Sub_Formula>
	|	<Condition>
	|	<Quantified_Formula>
<Quantified_Formula>	→	<Quantifier> (<Formula>)
	|	<Quantifier> <Quantified_Formula>
<Quantifier>	→	(**exists** <Variables>)
	|	(**forall** <Variables>)
<Condition>	→	<Operand> <Relational_Op> <Operand>
<Operand>	→	*Identifier*. *Identifier*

	|	*Constant*
<Vars_Attrs>	→	*Identifier*
	|	*Identifier*, <Vars_Attrs>
	|	*Identifier*.*Identifier*
	|	*Identifier*.*Identifier*, <Vars_Attrs>
<Variables>	→	*Identifier*
	|	*Identifier*, <Variables>
<Relational_Op>	→	=
	|	>
	|	<
	|	<>
	|	>=
	|	<=

Structured Query Language

The syntax recognized by the interpreter is the original SQL standard - not SQL-92. In addition to the basic select-from-where query expression, the interpreter handles grouping, aggregation, having, ordering, and nested subqueries. The interpreter, however, has some simplifying assumptions:

- Although the language allows nested queries and aggregation, a nested subquery cannot involve aggregation; aggregation must appear in the outermost query. This is not a severe restriction. The user need only break such a query into multiples queries.

  For example, consider the query to find the employees earning the maximum salary over the COMPANY database. Although the first SQL query is a valid query in the standard, it is not allowed by the tool. But the query can be evaluated using the second formulation.

```
1. select ssn, salary
   from   employee
   where  salary = (select max(salary)
                    from   employee);

2. maxsalary(maxSal) := select max(salary)
                        from   employee;
   select ssn, salary
   from   employee, maxsalary
   where  salary=maxSal;
```

- Another restriction limits the expressibility of mathematical expressions in the SELECT clause to involve at most one attribute.

<Query> → <Query_Term>
| <Query_Term> **union [all]** <Query>
| <Query_Term> **intersect** <Query>
| <Query_Term> **except** <Query>

<Query_Term> → <Sub_Query>
| **(**<Query>**)**

<Sub_Query> → **select** <Select_Vars> **from** <Expression>
[<Group_Clause> [<Having_Clause>]] [<Order_Clause>]

<Select_Vars> → ***
| [**distinct**] <Scalar_Varlist>
| [**all**] <Scalar_Varlist>

<Scalar_Varlist> → <Scalar_Var> [, <Scalar_Varlist>]

<Scalar_Vars> → <Term>
| <Term> - <Scalar_Var>
| <Term> + <Scalar_Var>

<Term> → <Factor>
| <Factor> / <Term>
| <Factor> * **<**Term>

<Factor> → [+ | -] <Primary>

<Primary> → <Atom>
| <Aggregation_Var>
| **(**<Scalar_Var>**)**

<Atom> → *Constant*
| *Identifier*

<Aggregation_Var> → **count(*)**
| **count(distinct** <Simple_Var>**)**
| <Function> **(**[**distinct**] <Simple_Var>**)**
| <Function> **(**[**all**] <Simple_Var>**)**

<Function> → **min**
| **max**
| **avg**
| **sum**

<Simple_Var> → *Identifier*

| *Identifier.Identifier*

<Expression> → <Relation_List> [**where** <Search_Condition>]
<Relation_List> → <Relation> [**,** <Relation_List>]
<Relation> → *Identifier*
| *Identifier Alias*
<Search_Condition> → <Boolean_Term>
| <Boolean_Term> **or** <Search_Condition>
<Boolean_Term> → <Boolean_Factor>
| <Boolean_Factor> **and** <Boolean_Term>
<Boolean_Factor> → [**not**] <Boolean_Primary>
<Boolean_Primary> → <Predicate>
| **(**<Search_Condition>**)**
<Predicate> → <Exists_Predicate>
| <In_Predicate>
| <Comparison_Predicate>
<Exists_Predicate> → **exists (**<Simple_Subquery>**)**
<In_Predicate> → <Simple_Var> [**not**] **in (**<Simple_Subquery>**)**
| <Simple_Var> [**not**] **in (**<Atom_List>**)**
<Atom_List> → <Atom> [**,** <Atom_List>]
<Comparison_Predicate> → <Simple_Var><Comparison><Simple_Var>
| <Simple_Var><Comparison><Simple_Subquery>
<Comparison> → =
| <>
| <
| >
| <=
| >=
<Group_Clause> → **group by** <Group_Varlist>
<Group_Varlist> → <Simple_Var> [**,** <Group_Varlist>]
<Having_Clause> → **having** <Having_Condition>
<Having_Condition> → <Having_Term>
| <Having_Term> **or** <Having_Condition>
<Having_Term> → <Having_Factor>

| <Having_Factor> **and** <Having_Term>

<Having_Factor> → [**not**] <Having_Primary>
<Having_Primary> → <Having_Compare>
| **(**<Having_Condition>**)**
<Having_Compare> → <Aggregation_Var><Comparison><Aggregation_Var>
<Order_Clause> → **order by** <Order_Varlist> [**asc** | **desc**]
<Order_Varlist> → <Order_Var> [**,** <Order_Varlist>]
<Order_Var> → <Simple_Var>
| *Constant*
<Simple_Subquery> → <Sub_Query> without <Aggregation_Var>, <Group_Clause>, or <Order_Clause>

Known Issues

- The key constraint is not checked when tuples are inserted or edited. This was a design decision for this version. Various forms of constraint checking are planned for the next version of WinRDBI.
- A bug in Java's Swing API causes the highlighting for cut-and-paste operations to apparently stop functioning in the query editor after switching between different QueryPanes or between a QueryPane and the SchemaPane. In reality, text can still be selected, it is just not highlighted on the screen. There are two workarounds. First, try switching windows using the window names on the Window menu. If that does not work, cover the WinRDBI window with another application's window -- any application will do. Then bring the WinRDBI window back to the front. The ability to highlight will be restored.
- Occasionally, the underlying Prolog code will get into a state from which it cannot recover following a valid exception message. A "Reset" item was added to the Schema menu to try to reset the Prolog engine; however, this operation currently does not achieve the desired effect. The only known workaround is to exit and restart WinRDBI.

Acknowledgments

The author, Rick Rankin, wishes to acknowledge the original RDBI authors: Chien-Ho Ho, Ana Hun, Changguan Fan, and Sarah Simons, with Ariela Sterns. Thanks are due as well to the authors of WinRDBI version 1.0, Eric Eckert and Kevin Piscator, for their work on the first graphical user interface and establishing the communications mechanisms with the Amzi! Prolog Logic Server. Also, thanks to Dr. Suzanne Dietrich for her feedback on the WinRDBI 2.0 program and documentation.

References

1. Dietrich, S. W., An Educational Tool for Relational Query Languages, *Computer Science Education*, Vol. 4, 1993, pp. 157-184.
2. Dietrich, S. W., Eckert, E., and Piscator, K., WinRDBI: A Windows-based Relational Database Educational Tool, *SIGCSE*, 1997, pp. 126-130.
3. Elmasri, R. and Navathe, S. B., *Fundamentals of Database Systems (2nd ed.)*. Benjamin/Cummings, CA, 1994.

Introduction

WinRDBI is a teaching tool for use in the introductory database course at Arizona State University. It allows the user to interactively experiment with Relational Algebra, Domain Relational Calculus, Tuple Relational Calculus, and Structured Query Language expressions. WinRDBI interacts with a Prolog engine which performs the actual database manipulations.

WinRDBI is based on RDBI, which was written over a number of years by students as independent study projects. RDBI was written in Quintus Prolog and ran under UNIX or VMS. Using RDBI, students interacted with a command-line oriented user interface. This required students to use on-campus facilities, either on-site or via dial-up, to access the program. Additionally, RDBI could only be used on systems that were licensed to run Quintus Prolog. Further information and a description of how to use the original RDBI tool is described in [1].

Later, Version 1.0 of WinRDBI was written. Its interface and use is documented in [2]. In addition to providing a simpler, more intuitive Graphical User-Interface (GUI) interface with which students could interact, WinRDBI gave students who have IBM compatible computers at home the ability to run RDBI on their own computers. To help accomplish that goal, WinRDBI used the Amzi! Prolog Logic Server rather that Quintus Prolog. The Amzi! product was chosen because it has no runtime license restrictions for non-commercial use.

Version 1.0 of WinRDBI was used for several semesters at ASU. Over that time, several shortcomings were found. For example, the user could interact with only one query at a time. Once a query was completed, it had to be saved to a file before a new query could be started. The query was then no longer visible from within the user interface. If the user wanted to review a previously completed query, a separate editor or viewer program had to be used. This made it very difficult to construct the answer to a question that involved multiple sub-queries. Another related shortcoming was that any comments that were entered into the user interface as part of a query were lost when the query was saved to disk.

Relative to the database, Version 1.0 provided very minimal editing capabilities. Tuples could be inserted and deleted, but not edited. In order to change a field in a tuple, the entire tuple had to be deleted and re-entered. The only other option was to save the database as a text file of Prolog facts, call up a text editor to change the appropriate data, then reload the database into the user interface. Further, any relations that were materialized as part of a query became part of the database. After a query had been executed, it was not possible to save the original database without first deleting these extra relations.

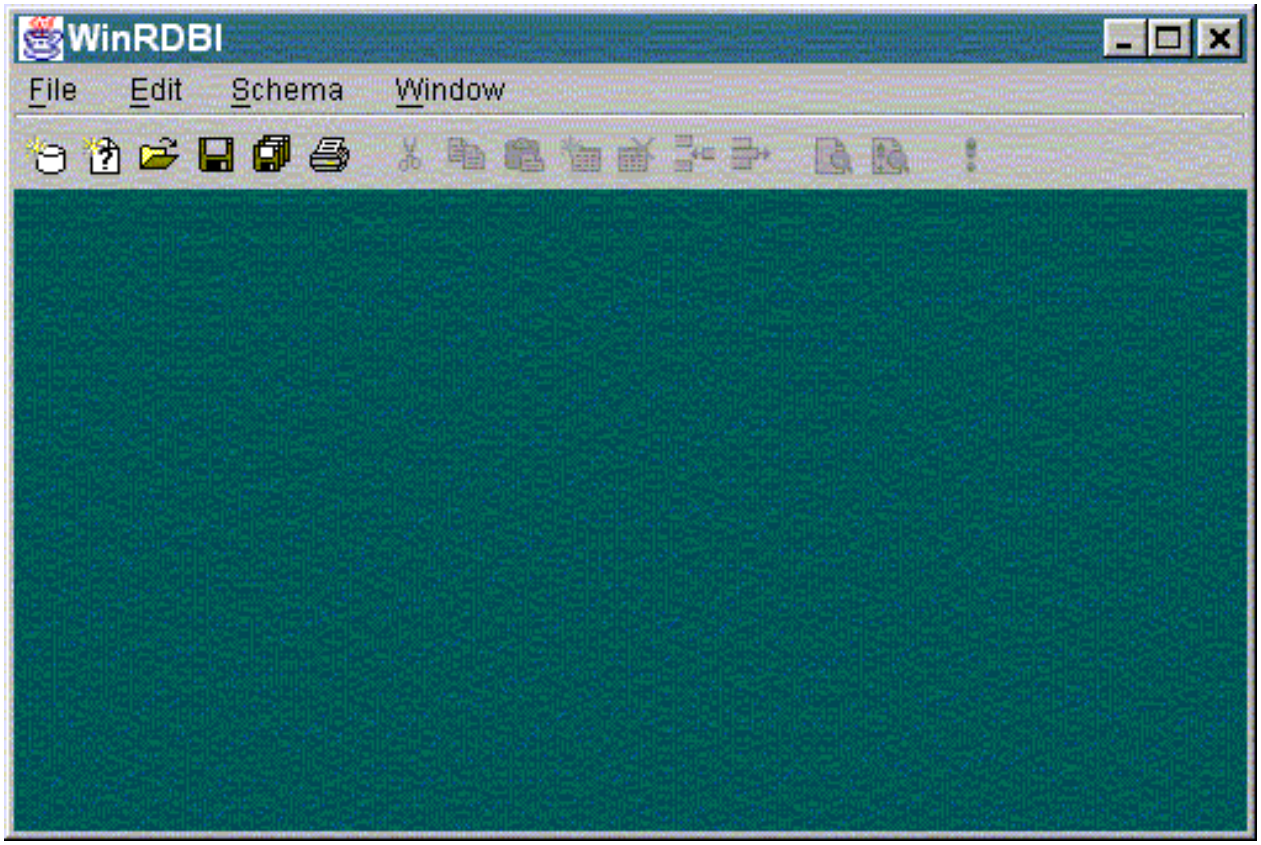
These limitations gave rise to the current version, WinRDBI 2.0. Version 2.0 is a complete redesign of the user interface of Version 1.0. Version 2.0 is written in Java and uses the Java Foundation Classes (JFC), or Swing, to provide the user interface. Version 2.0 still interacts with the Amzi! Prolog Logic Server to perform the database operations, but the user interface, described in the next section, is entirely different.

User Interface

WinRDBI 2.0 was designed as a Multiple Document Interface (MDI) application. In the general sense, a MDI application allows you to have more than one "document" open at a time, for example, more than one word-processor document or spreadsheet. In the case of WinRDBI, a document may be a database or a query file. While WinRDBI allows you to have multiple query files open simultaneously, only one database file may be open at a time.

Components

The image below shows the initial WinRDBI screen. At this point, you may open an existing query or database or create new ones using the File menu or the toolbar.

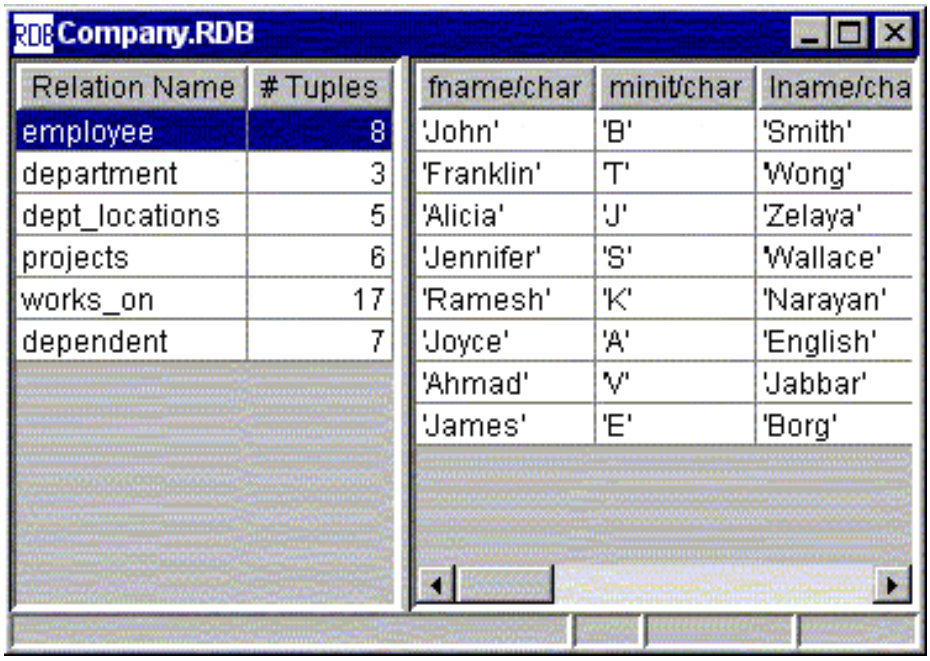


The main screen has the normal components of a Microsoft Windows application: a menu

and a toolbar

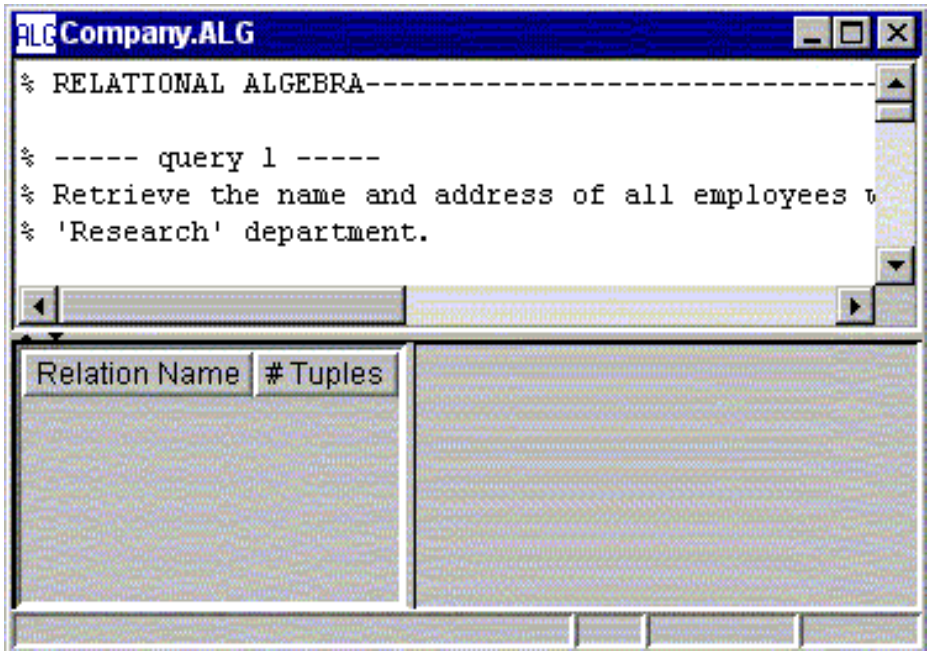
In the following sections, these components will be described in some detail.

There are two additional components which will be mentioned regularly. The first is called a SchemaPane. A SchemaPane is an internal window on the main screen that is used to contain the schema, or database, against which the queries will be executed. The figure below shows a SchemaPane.



Notice that the pane is vertically divided in two. On the left is a list of the relation names and the number of tuples in the relation. On the right is a list of the tuples in the relation. In a SchemaPane, you can add, delete, and edit relations and tuples. There may be only one SchemaPane open at a time.

The second component of which you need to be aware is called a QueryPane. A QueryPane is used to enter queries and display the results of the queries. A QueryPane is shown below.



Obviously, a QueryPane is divided into three sections. The top section is a very basic text editor and is used for entering queries in one of the four query languages. The bottom section, which is similar to a SchemaPane, is used to display the results of the queries. Note that the relations and tuples representing query results cannot be edited.

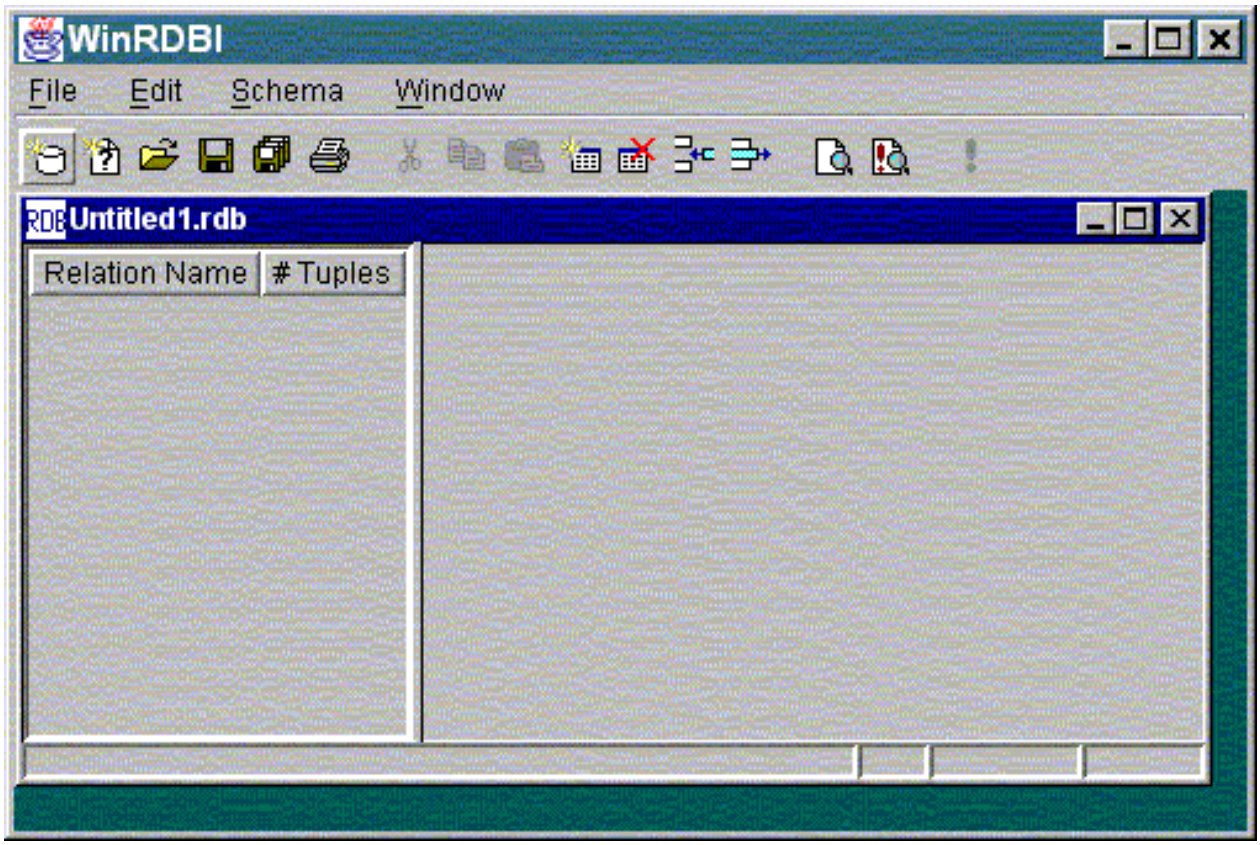
Menu Structure

This section describes the available menu items in WinRDBI 2.0.

File Menu

New Database

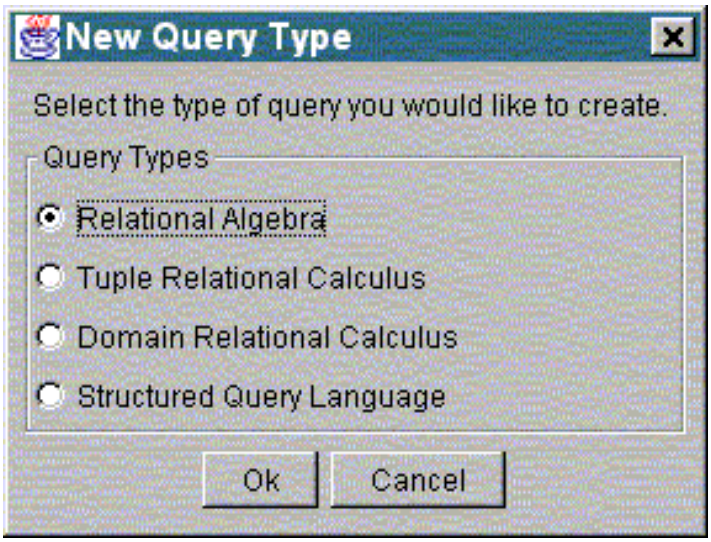
The New Database menu item allows you to create a new RDBI database. When you select this option, an empty SchemaPane will be created as shown below.



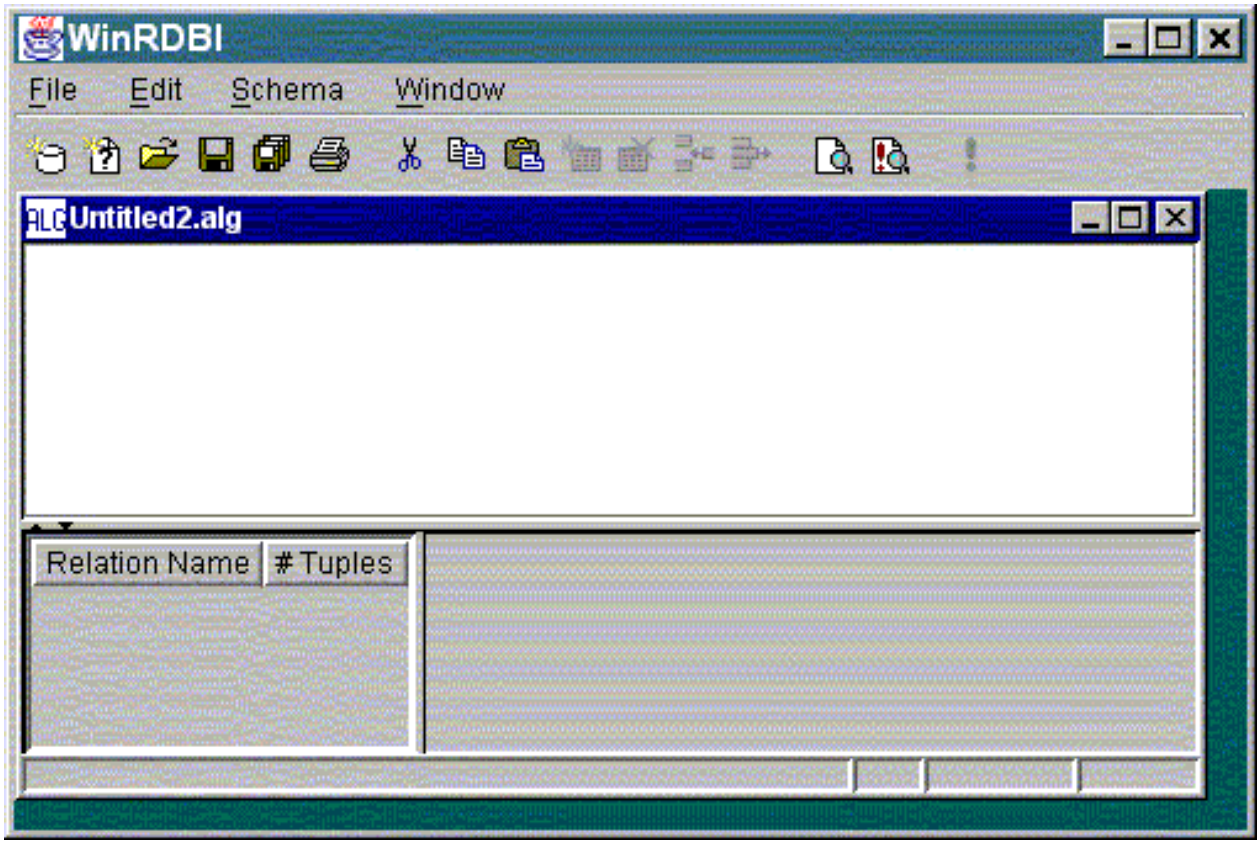
At this point, you can use various toolbar icons or items on the Schema menu to add relations and tuples to your database.

New Query

When you select the New Query item, the dialog below will be displayed, asking you what kind of queries you want to enter: Relational Algebra, Domain Relational Calculus, Tuple Relational Calculus, or Structured Query Language. The default is Relational Algebra. You must select one of the options to proceed.

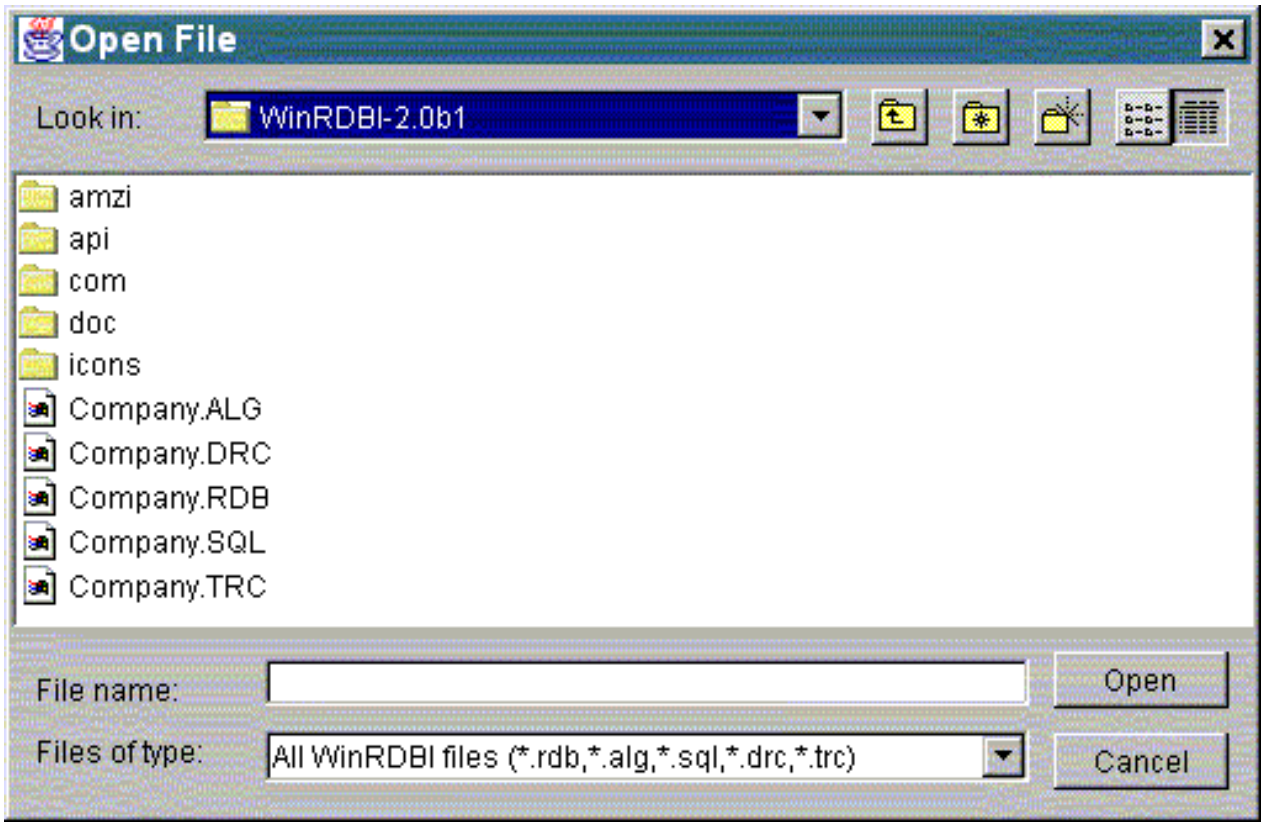


Once you have selected the appropriate type of queries, you will be given an empty QueryPane, as shown below.



Open...

If you choose "Open..." from the File menu, you will be shown the following dialog from which you can choose a query or database file to open.



Close

Choosing this item will close the currently active window. If there are unsaved changes, you will be offered a chance to save the changes.

Save

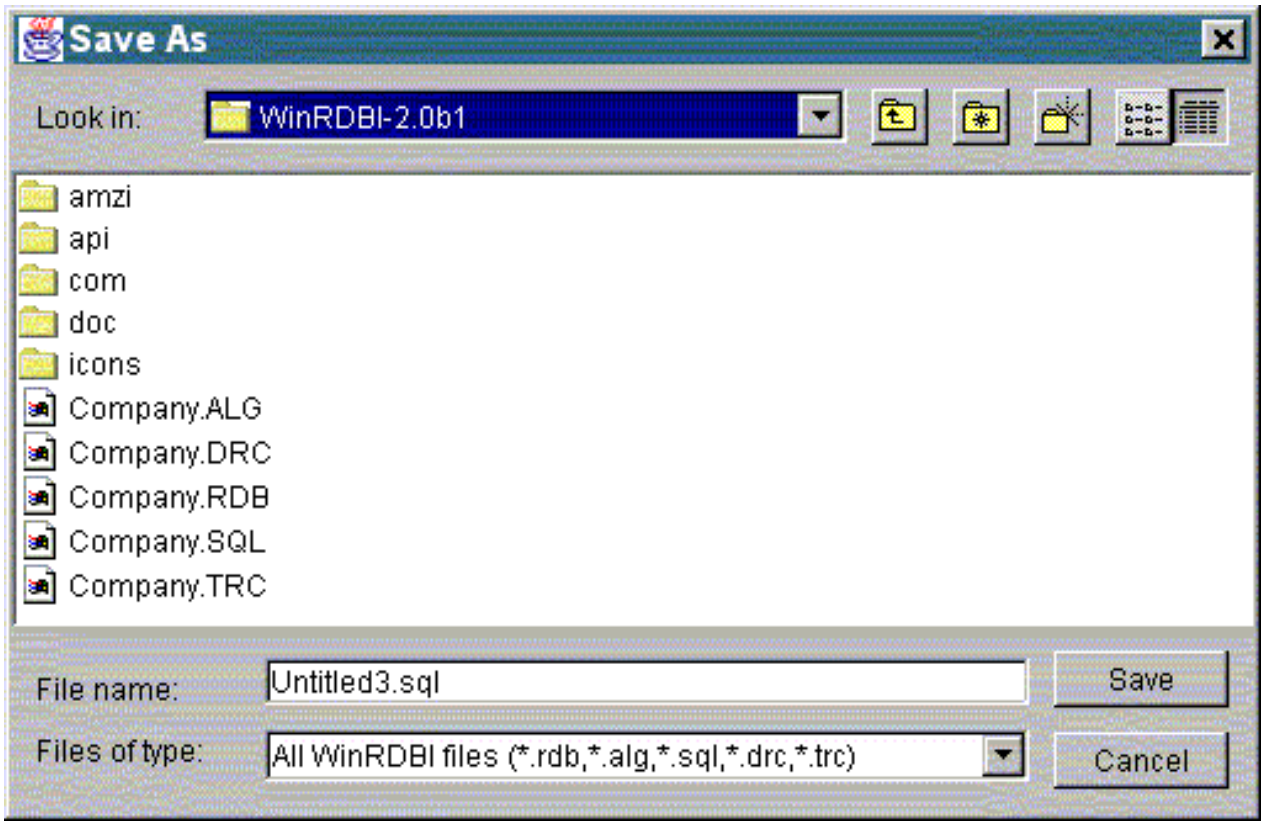
Save the contents of the currently active QueryPane or SchemaPane.

Save All

Save the contents of all open panes.

Save As...

The Save As item allows you to save the active file under a new name. If you choose this item, you will see a dialog box similar to the one below.



You would enter a new file name in the field labeled "File name:". In the figure above, this is the field that contains "Untitled1.sql". If you enter the name of a file that already exists, you will be asked if you want to overwrite the existing file. If you answer no, you will be given the opportunity to enter another file name. If you indicate that you *do* want to overwrite the existing file, then the save action will proceed normally.

Print...

Choosing the Print item will allow you to print queries, query results, or relations from the database based on the active pane. If one or more relations are selected in the results or the database, you will be given the option to print only the selected relations or all relations. Multiple non-adjacent relations may be selected by holding down the control key on the keyboard while clicking with the left mouse button. An entire region may be selected by left-clicking on the first relation, then holding down the shift key while left-clicking another relation. Control-shift-left-click can be used to select non-adjacent regions.

Exit

Choosing "Exit" from the file menu will close all open windows and exit the program. If there are unsaved changes, you will be given the opportunity to save them.

Edit Menu

The Edit Menu contains the normal clipboard operations Cut, Copy, and Paste, along with Search and Replace operations. The clipboard operations are only available when you are editing queries or when you are editing a specific field in a tuple.

Cut

The Cut item allows you to copy selected text from a query editor or field editor to the clipboard. The selected text is then deleted from the editor.

Copy

The Copy item performs essentially the same function as the Cut item, but the selected text is ***not*** deleted from the editor.

Paste

Choosing the Paste item copies text *from* the clipboard *to* the query or field editor.

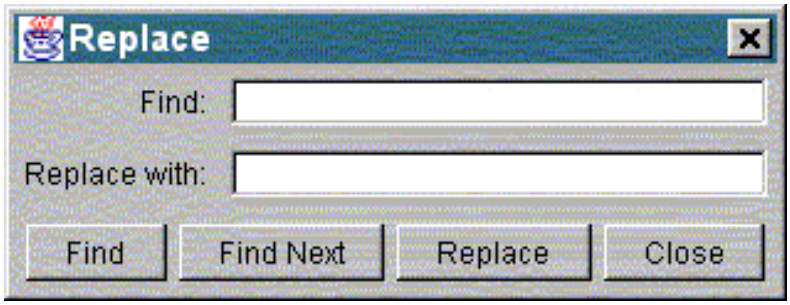
Find...

Select the Find item to search for text in the currently active Pane. The pane can be either a QueryPane or a SchemaPane. When you select the Find item, you will see the following dialog box:

Enter the text you wish to find in the field. To initiate the search, click on the "Find" button, or press enter. If the text is found, it will be selected. If it is not found, a dialog box will be displayed indicating that no matching text was found. To find subsequent occurrences, click on the "Find Next" button. Click on the "Close" button to dismiss the dialog box.

Replace...

The Replace item allows you to search the active Pane for a string and replace it with another string. Like the Find item, Replace can be used in either a QueryPane or a SchemaPane. Note, however, that the Replace option *does not* work in the results portion of a QueryPane, as the result table is not editable. The dialog box looks like



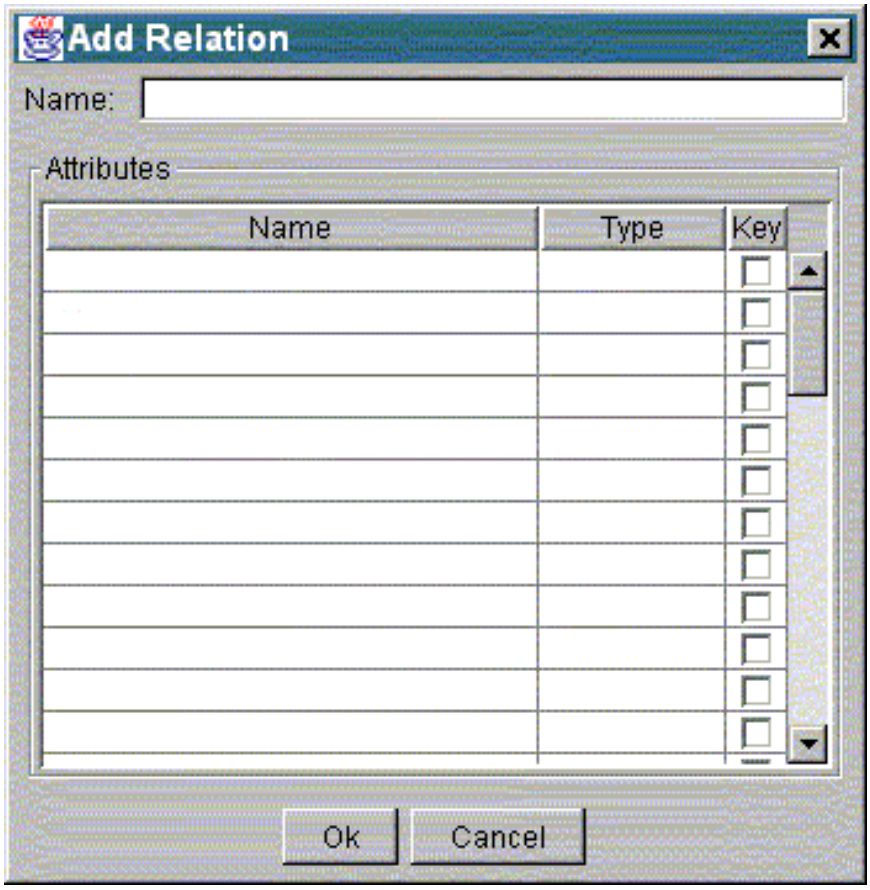
Like the Find dialog, enter the text for which you wish to search in the field labeled "Find:". Then enter the replacement text in the field labeled "Replace with:" and click the "Find" button. If the text is found, it will be highlighted. To replace the text, click on the "Replace" button. Clicking on "Replace" will cause the

highlighted text to be replaced with the replacement text and will automatically search for the next occurrence of the search string. If you don't wish to replace an occurrence, just click on "Find Next". Of course, click the "Close" button to dismiss the dialog.

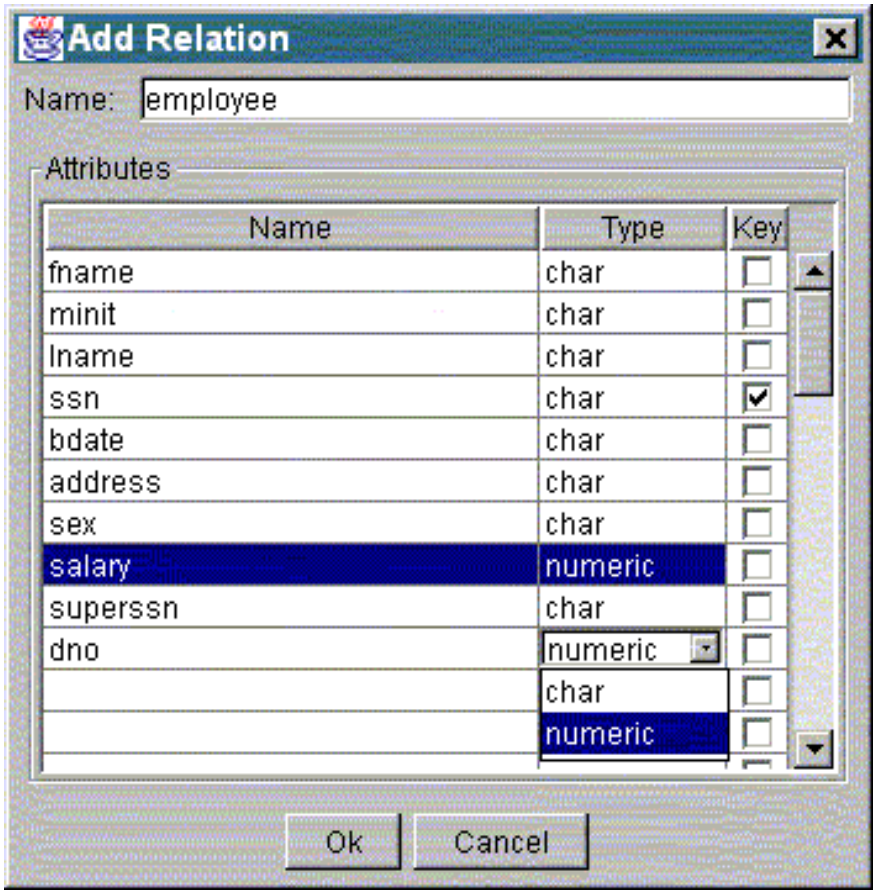
Schema Menu

Add Relation

Choose the Add Relation item to add a new relation to a database. This option is only available when a SchemaPane is active. When you choose this item, the following dialog will be displayed



The relation name should be entered in the field labeled "Name:", while the attributes should be entered in the table below. Note that relation and attribute names must be identifiers starting with a lower case letter. If an attribute is part of the key, put a check in the "Key" column by clicking on the appropriate box. Here's a more complete example.



Note that the attribute named "ssn" is the key attribute. You can create a composite key by checking more that one box. The "Type" column allows only two entries, char and numeric, which are selected from the drop-down list.

Copy Relation to Database

When a QueryPane is active, this item is available. It allows you to copy the selected relation from the query results to the SchemaPane. In this way, you can save and/or edit the results of a query. This option is only available when a QueryPane is active.

Delete Relation

The Delete Relation item deletes the selected relations from the database. This option is only available when a SchemaPane is active.

Edit Relation

Select the Edit Relation option from the Schema menu to bring up a dialog box which will allow you to edit the name and attributes of a relation. This dialog box is the same as the "Add Dialog" box. The only restriction is that you may not change the type of an attribute. You may also activate the dialog box by double-clicking the left mouse button on the relation name. This option is only available when a SchemaPane is active.

Add Tuple

The Add Tuple item adds one or more tuples to the currently selected relation. If there are multiple tuples selected, then the number of tuples added will be equal to the number of tuples selected. The new tuples will always be added to the end of the table. This option is only available when a SchemaPane is active.

Delete Tuple

Selecting the Delete Tuple item causes the selected tuples to be deleted. This option is only available when a

SchemaPane is active.

Insert Tuple

Choose the Insert Tuple item to insert one of more tuples into the currently selected relation. If there are multiple tuples selected, then the number of tuples inserted will be equal to the number of tuples selected. The new tuples will be inserted above the first selected tuple. This option is only available when a SchemaPane is active.

Execute Queries

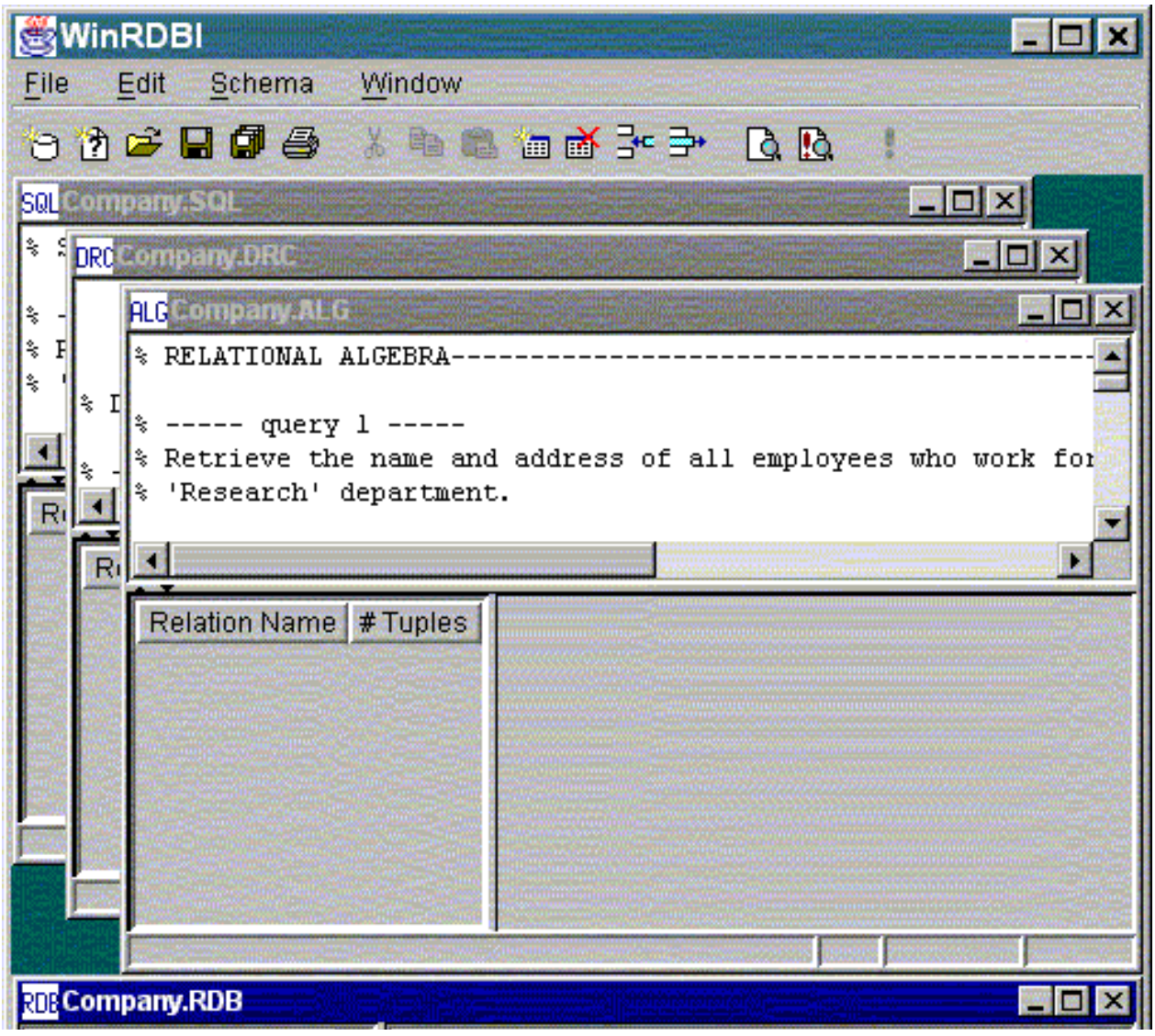
Execute the queries in the currently active QueryPane. This item is available only when a QueryPane is active *and* a database is loaded. The results will be shows in the table in the bottom half of the QueryPane.

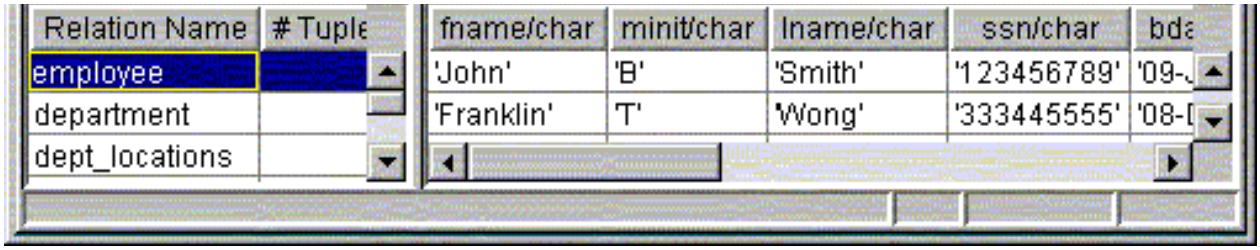
Window Menu

The Window menu will contain the names of all open panes. If a pane is hidden, you can bring it to the front by selecting the corresponding item in the Window menu.

Arrange

In addition to an item corresponding to each pane, the Window menu contains an Arrange item. The Arrange item arranges all open panes by tiling the QueryPanes in the upper 3/4 of the main screen and puts the SchemaPane, if one is open, in the lower 1/4 of the main screen. The end result will look something like the figure below.





Toolbar

The following table describes the action performed by each toolbar icon and lists the equivalent menu operation. The notation A → B means to select item B from menu A. More specifically, File → Open means to select the Open item from the File menu.

Icon	Description	Menu Equivalent	Keyboard Shortcut
	Create a new, empty SchemaPane.	File → New Database	Ctrl+D
	Create a new, empty QueryPane.	File → New Query	Ctrl+Q
	Open an existing file.	File → Open	Ctrl+O
	Save the contents of the currently active pane.	File → Save	Ctrl+S
	Save the contents of all open panes.	File → Save All	
	Print the currently active pane.	File → Print	Ctrl+P
	Cut the selected text to the clipboard.	Edit → Cut	Ctrl+X
	Copy the selected text to the clipboard.	Edit → Copy	Ctrl+C
	Paste the clipboard contents to the current location.	Edit → Paste	Ctrl+V
	Add a relation to the current database.	Schema → Add Relation	
	Delete the selected relations from the current database.	Schema → Delete Relation	
	Insert one or more tuples into the current relation.	Schema → Insert Tuple	
	Delete the selected tuples from the current relation.	Schema → Delete Tuple	
	Search for text in the active QueryPane or SchemaPane.	Edit → Find	Ctrl+F
	Replace text in the active QueryPane or SchemaPane.	Edit → Replace	Ctrl+R
	Execute the queries in the active QueryPane.	Schema → Execute Queries	Ctrl+E

Languages by Example

In this section, we will investigate the solutions to two example queries over the Company database example described in [3] to illustrate some of the features of the languages. The languages are described in [1] and the BNF for the languages is given as part of this User's Guide. The WinRDBI languages are case sensitive, with the exception of SQL. Relation and attribute names must be identifiers starting with a lower case letter. Constants in all languages and in the database are either (single) quoted strings or numeric constants. Variables (in DRC and TRC) are identifiers that start with an upper case letter.

Note that you can experiment with the solutions given below by loading the Company.RDB file into WinRDBI, creating a new QueryPane of the appropriate type, cutting and pasting the code into the editor portion of the QueryPane, and executing the queries. These queries are also part of the example called Company distributed with WinRDBI.

Example 1

The first example we will discuss is a very simple one that queries the Company database to find out which employees work for the department named 'Research'. The result contains the first name, last name, and address of each employee.

Relational Algebra

The relational algebra solution creates two temporary relations to solve this problem. The first relation will have a single tuple which contains the department number of the Research department. This information is necessary since only the department number, not the name, is kept in the employee record. The second temporary relation is formed by taking the Cartesian product of the first temporary relation with the employee relation, and selecting only those tuples in which the employee's department number (dno) is equal to the Research department's number (dnumber). This yields a relation that contains only those tuples for employees that work in the Research department. Finally, the attributes fname, lname and address are projected out of the second temporary relation to display only the requested information.

```
research_dept:=select dname='Research' (department);
research_dept_emps:=select dnumber=dno (research_dept product employee);
alg1:=project fname, lname, address (research_dept_emps);
```

Domain Relational Calculus

DRC is a declarative language. The solution below specifies the set of tuples for which there exists a department number DNO for a department named 'Research' and employees whose home department is the same as DNO.

```
drc1:= {FNAME, LNAME, ADDRESS |
    (exists DNO)
        (department('Research',DNO,_,_) and
             employee(FNAME,_,LNAME,_,_,ADDRESS,_,_,_,DNO))};
```

Tuple Relational Calculus

The TRC solution is very similar to the DRC solution in that it declares the employees whose home department number is the same as the Research department's number.

```
trc1:= {T.fname, T.lname, T.address |
    employee(T) and
    (exists D)
        (department (D) and D.dname='Research' and D.dnumber=T.dno)};
```

Structured Query Language

The SQL solution simply extracts the first name, last name, and address tuples from the employee relation where the employee's department number is the same as the Research department's department number.

```
sql1 := select fname,lname,address
        from employee, department
        where dname = 'Research' and dnumber = dno;
```

Example 2

The second example we will examine asks which employees in the company database have two or more dependents. Below are the solutions in all four languages supported by WinRDBI.

Relational Algebra

As pointed out in [3], relational algebra does not support aggregate functions such as max, min, or count. However, it is possible to solve this problem in relational algebra if we assume that no employee has more than one dependent with the same name. Under that assumption, a solution can be constructed as follows:

1. Create a relation where each tuple contains an employee's social security and the name of one of the employee's dependents.
2. Make a second copy of the relation created in step 1.
3. Take the Cartesian product of the relations created in steps 1 and 2, and extract tuples in which the employee social security numbers are the same but the dependent names are different.
4. Obtain the names of the employees having more than one dependent by creating the natural join of the employee relation and the relation from step 3 and projecting out the employee's first and last names.

Here's the code:

```
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

The DRC declarative solution given below declares that there exists an employee who has two dependents whose names are not equal.

```
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

The TRC solution is very similar to the DRC solution. Like DRC, TRC is a declarative language. The solution to the query reads exactly like the DRC solution.

```
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)};
```

Structured Query Language

The SQL solution makes use of SQL's built-in count function to count the number of dependents of an employee, which is stored in an intermediate relation. The intermediate relation is then used in conjunction with the employee relation to display the names of the employees with more that one dependent.

```
dep_num(ssn,depent) := 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
               depent >= 2;
```

Language Assumptions and BNF

In the following grammar, terminal symbols are in **boldface**, non-terminal symbols are in angle brackets (<>), patterns are in *italics*, and braces ({}) indicate optional items. Note that WinRDBI requires relation and attribute names to begin with a lower case letter, while variables must begin with an upper case letter. Constants must be (single) quoted strings or numeric constants.

<Query_Definition> → <Query> **;**
| <Assignment_Statement> **;**
<Assignment_Statement> → *Relation_Name* **:=** <Query>
| *Relation_Name* **(** <Attribute_List> **) :=** <Query>
<Attribute_List> → *Attribute_Name* **,** <Attribute_List>
| *Attribute_Name*

Relational Algebra

The relational algebra is a procedural language with the following assumptions regarding its operators:

- The union, difference and intersect operators require that the operand relations are compatible, i.e., the operand relations have identical schemas.
- The (Cartesian) product operator requires that the operand relations have disjoint attribute names.
- The njoin (natural join) operator does ***NOT*** require common attribute names in the operand relations; if the operand relations have disjoint attribute names, then the njoin results in a Cartesian product.
- The θ-join and division operators are purposely not provided by the interpreter so that students investigate the equivalent definitions of these operators in terms of the fundamental relational algebra operators.

<Query> → <Expression>
<Expression> → *Identifier*
| (<Expression>)
| <Select_Expr>
| <Project_Expr>
| <Binary_Expr>
<Select_Expr> → **select** <Condition> (<Expression>)
<Project_Expr> → **project** <Attributes> (<Expression>)
<Binary_Expr> → (<Expression>) <Binary_Op> (<Expression>)
<Condition> → <And_Condition>

	|	<And_Condition> or <Condition>
<And_Condition>	→	<Rel_Formula>
	|	<Rel_Formula> and <And_Condition>
<Rel_Formula>	→	<Operand> <Relational_Op> <Operand>
	|	(<Condition>)
<Binary_Op>	→	**union**
	|	**njoin**
	|	**product**
	|	**difference**
	|	**intersect**
<Attributes>	→	*Identifier*
	|	*Identifier*, <Attributes>
<Operand>	→	*Identifier*
	|	*Constant*
<Relational_Op>	→	=
	|	>
	|	<
	|	<>
	|	>=
	|	<=

Domain Relational Calculus

The domain relational calculus (DRC) language is a declarative, positional language where variables range over the domain of attributes. Note that constants and anonymous variables (_) may be used within a DRC expression. For example, consider the query that returns the names and salaries of the female employees earning more than 35000:

```
{ LName, FName, Salary |
    employee(FName,_,LName,_,_,_,'F',Salary,_,_) and Salary > 35000};
```

The interpreter assumes that

- the names of the attributes in the resulting schema is a lowercase version of the variable names, which must be an identifier starting with an uppercase letter. In the above example, the result schema has the attributes lname, fname, and salary.
- the query expression is safe, assuming the left-to-right evaluation of Prolog. The above query expression is safe, however, the following logically equivalent version of the example is not allowed:

  ```
  { LName, FName, Salary |
      Salary > 35000 and employee(FName,_,LName,_,_,_,'F',Salary,_,_) };
  ```

  The rule of thumb for writing safe expressions is to make sure that a variable is bound to a value before it is used, either in a comparison or within a negation.

<Query>	→	{ <Variables> | <Formula> }
<Formula>	→	<And_Formula>
	|	<And_Formula> **or** <Formula>
<And_Formula>	→	<Sub_Formula>
	|	<Sub_Formula> **and** <And_Formula>
<Sub_Formula>	→	*Identifier* (<Vars_Cons>)
	|	(<Formula>)
	|	**not** <Sub_Formula>
	|	<Condition>
	|	<Quantified_Formula>
<Quantified_Formula>	→	<Quantifier> (<Formula>)
	|	<Quantifier> <Quantified_Formula>

```
    <Quantifier> → ( exists <Variables> )
                 |  ( forall <Variables> )
     <Condition> → <Operand> <Relational_Op> <Operand>
       <Operand> → Identifier
                 |  Constant
     <Variables> → Identifier
                 |  Identifier, <Variables>
    <Vars_Cons>  → Identifier
                 |  Identifier, <Vars_Cons>
                 |  Constant
                 |  Constant , <Vars_Cons>
<Relational_Op>  → =
                 |  >
                 |  <
                 |  <>
                 |  >=
                 |  <=
```

Tuple Relational Calculus

The tuple relational calculus (TRC) language is a declarative language similar to DRC except that variables range over tuples and attributes are referenced using dot notation. The interpreter only evaluates safe TRC expressions. In TRC, this means that the tuple variable must be bound to its associated relation before the values of its attributes can be accessed and before the variable is used within a negation. Here is the TRC version of the query that returns the names and salaries of the female employees earning more than 35000:

```
{ E.lname, E.fname, E.salary |
    employee(E) and E.sex='F' and E.salary>35000 };
```

The only additional assumption for TRC is a shorthand notation for selecting all attributes associated with a tuple variable. For example, the following shows the result of the above query when all employee attributes are returned:

```
{ E | employee(E) and E.sex='F' and E.salary>35000 };
```

```
            <Query> → { <Vars_Attrs> | <Formula> }
          <Formula> → <And_Formula>
                    |  <And_Formula> or <Formula>
      <And_Formula> → <Sub_Formula>
                    |  <Sub_Formula> and <And_Formula>
      <Sub_Formula> → Identifier ( Variable )
                    |  ( <Formula> )
                    |  not <Sub_Formula>
                    |  <Condition>
                    |  <Quantified_Formula>
<Quantified_Formula> → <Quantifier> ( <Formula> )
                    |  <Quantifier> <Quantified_Formula>
      <Quantifier> → ( exists <Variables> )
                    |  ( forall <Variables> )
       <Condition> → <Operand> <Relational_Op> <Operand>
         <Operand> → Identifier. Identifier
                    |  Constant
       <Vars_Attrs> → Identifier
                    |  Identifier, <Vars_Attrs>
                    |  Identifier.Identifier
                    |  Identifier.Identifier, <Vars_Attrs>
       <Variables> → Identifier
```

| *Identifier*, <Variables>

<Relational_Op> → =
| >
| <
| <>
| >=
| <=

Structured Query Language

The syntax recognized by the interpreter is the original SQL standard - not SQL-92. In addition to the basic select-from-where query expression, the interpreter handles grouping, aggregation, having, ordering, and nested subqueries. The interpreter, however, has some simplifying assumptions:

- Although the language allows nested queries and aggregation, a nested subquery cannot involve aggregation; aggregation must appear in the outermost query. This is not a severe restriction. The user need only break such a query into multiples queries.

  For example, consider the query to find the employees earning the maximum salary over the COMPANY database. Although the first SQL query is a valid query in the standard, it is not allowed by the tool. But the query can be evaluated using the second formulation.

```
1. select ssn, salary
   from   employee
   where  salary = (select max(salary)
                    from   employee);
2. maxsalary(maxSal) := select max(salary)
                        from   employee;
   select ssn, salary
   from   employee, maxsalary
   where  salary=maxSal;
```

- Another restriction limits the expressibility of mathematical expressions in the SELECT clause to involve at most one attribute.

<Query> → <Query_Term>
| <Query_Term> **union [all]** <Query>
| <Query_Term> **intersect** <Query>
| <Query_Term> **except** <Query>

<Query_Term> → <Sub_Query>
| **(**<Query>**)**

<Sub_Query> → **select** <Select_Vars> **from** <Expression>
[<Group_Clause> [<Having_Clause>]] [<Order_Clause>]

<Select_Vars> → ***
| [**distinct**] <Scalar_Varlist>
| [**all**] <Scalar_Varlist>

<Scalar_Varlist> → <Scalar_Var> [, <Scalar_Varlist>]

<Scalar_Vars> → <Term>
| <Term> - <Scalar_Var>
| <Term> + <Scalar_Var>

<Term> → <Factor>

| <Factor> / <Term>

| <Factor> * <Term>

<Factor> → [+ | -] <Primary>

<Primary> → <Atom>
| <Aggregation_Var>
| **(**<Scalar_Var>**)**

<Atom> → *Constant*
| *Identifier*

<Aggregation_Var> → **count(*)**
| **count(distinct** <Simple_Var>**)**
| <Function> **(**[**distinct**] <Simple_Var>**)**
| <Function> **(**[**all**] <Simple_Var>**)**

<Function> → **min**
| **max**
| **avg**
| **sum**

<Simple_Var> → *Identifier*
| *Identifier*.*Identifier*

<Expression> → <Relation_List> [**where** <Search_Condition>]

<Relation_List> → <Relation> [**,** <Relation_List>]

<Relation> → *Identifier*
| *Identifier Alias*

<Search_Condition> → <Boolean_Term>
| <Boolean_Term> **or** <Search_Condition>

<Boolean_Term> → <Boolean_Factor>
| <Boolean_Factor> **and** <Boolean_Term>

<Boolean_Factor> → [**not**] <Boolean_Primary>

<Boolean_Primary> → <Predicate>
| **(**<Search_Condition>**)**

<Predicate> → <Exists_Predicate>
| <In_Predicate>
| <Comparison_Predicate>

<Exists_Predicate> → **exists (**<Simple_Subquery>**)**

<In_Predicate> → <Simple_Var> [**not**] **in (**<Simple_Subquery>**)**
| <Simple_Var> [**not**] **in (**<Atom_List>**)**

<Atom_List> → <Atom> [**,** <Atom_List>]

<Comparison_Predicate> → <Simple_Var><Comparison><Simple_Var>
| <Simple_Var><Comparison><Simple_Subquery>

<Comparison> → =
| <>
| <
| >
| <=
| >=

<Group_Clause> → **group by** <Group_Varlist>
<Group_Varlist> → <Simple_Var> [**,** <Group_Varlist>]
<Having_Clause> → **having** <Having_Condition>
<Having_Condition> → <Having_Term>
| <Having_Term> **or** <Having_Condition>
<Having_Term> → <Having_Factor>
| <Having_Factor> **and** <Having_Term>
<Having_Factor> → [**not**] <Having_Primary>
<Having_Primary> → <Having_Compare>
| **(**<Having_Condition>**)**
<Having_Compare> → <Aggregation_Var><Comparison><Aggregation_Var>
<Order_Clause> → **order by** <Order_Varlist> [**asc** | **desc**]
<Order_Varlist> → <Order_Var> [**,** <Order_Varlist>]
<Order_Var> → <Simple_Var>
| *Constant*
<Simple_Subquery> → <Sub_Query> without <Aggregation_Var>, <Group_Clause>, or <Order_Clause>

Known Issues

- The key constraint is not checked when tuples are inserted or edited. This was a design decision for this version. Various forms of constraint checking are planned for the next version of WinRDBI.
- A bug in Java's Swing API causes the highlighting for cut-and-paste operations to apparently stop functioning in the query editor after switching between different QueryPanes or between a QueryPane and the SchemaPane. In reality, text can still be selected, it is just not highlighted on the screen. There are two workarounds. First, try switching windows using the window names on the Window menu. If that does not work, cover the WinRDBI window with another application's window -- any application will do. Then bring the WinRDBI window back to the front. The ability to highlight will be restored.
- Occasionally, the underlying Prolog code will get into a state from which it cannot recover following a valid exception message. A "Reset" item was added to the Schema menu to try to reset the Prolog engine; however, this operation currently does not achieve the desired effect. The only known workaround is to exit and restart WinRDBI.

Acknowledgments

The author, Rick Rankin, wishes to acknowledge the original RDBI authors: Chien-Ho Ho, Ana Hun, Changguan Fan, and Sarah Simons, with Ariela Sterns. Thanks are due as well to the authors of WinRDBI version 1.0, Eric Eckert and Kevin Piscator, for their work on the first graphical user interface and establishing the communications mechanisms with the Amzi! Prolog Logic Server. Also, thanks to Dr. Suzanne Dietrich for her feedback on the WinRDBI 2.0 program and documentation.

References

1. Dietrich, S. W., An Educational Tool for Relational Query Languages, *Computer Science Education*, Vol. 4, 1993, pp. 157-184.
2. Dietrich, S. W., Eckert, E., and Piscator, K., WinRDBI: A Windows-based Relational Database Educational Tool, *SIGCSE*, 1997, pp. 126-130.
3. Elmasri, R. and Navathe, S. B., *Fundamentals of Database Systems (2nd ed.)*. Benjamin/Cummings, CA, 1994.