GEB: SQLite in Tcl/Tk in SQLite

Abstract

GEB is a Tcl/Tk program for displaying and manipulating SQLite databases. Each of its major functions is stored in an SQLite table. It has much of the functionality of the SQLite stand-alone executable, plus spreadsheet-like table display, nearly complete ALTER TABLE functions, SQLite version 2 to version 3 conversion, the ability to execute a table as either SQL or Tcl, and a few other functions I had a need for. The name GEB was chosen because using an SQLite database to store the program which displays and modifies the SQLite database itself seemed reminiscent of the "self reference at a higher level" which was a recurring theme in the book "Gödel Escher Bach: An Eternal Golden Braid." If you insist on interpreting it as an acronym, it could stand for "Gerry's Experimental Box," but that is really a backronym.

This paper contains a brief background and history of GEB, a discussion of its current capabilities, and a list of some possible future additions.

Background

GEB began with a desire to migrate some smallish database projects from proprietary s/w (MS Access and dBase n) to FOSS. SQLite caught my eye. Neither the command-line program (CLP) nor any of the free packages for it seemed to do everything I wanted, so I decided to roll my own. My project would have to run on Linux and Windows, and I was looking at both Perl and Tcl, with the former having the edge because of its being used a lot at work. I installed both language packages on both platforms. Tcl and SQLite worked perfectly on both systems. I had problems with the perl package on one of the platforms (no idea which one by now), so I went with Tcl.

My goal was to put all the functionality I needed into my package, so I wouldn't have to switch over to the stand-alone executable or some other program for some step in the middle of a process I was undertaking.

I have tried to be extremely cautious about data integrity. Algorithms have been kept as simple as possible, to make them easier to verify. In some cases where GEB generates SQL, the user is shown the SQL and is given the chance to decline to accept it.

History (showsqlite)

The first functions I needed were displaying the tables in a file and their fields, and the contents of a table. Figure 1 shows my first version, which used text widgets. The upper window shows one table per line, with the number of entries in blue, the table name in black, and the column names in red. Clicking on a table name lists the contents in the lower window.

The display format was not appropriate for editing the table contents, so a separate editing window was created, shown in Figure 2. It was opened using the Data – Edit menu.

Since I was always thinking of more columns I needed to add to some tables, an Alter Table function was needed. This window, shown in Figure 3, was brought up by clicking on the column names of a table. Columns could be added or deleted, changed in order, renamed, or copied. All changes were made by copying to a TEMP table (possibly with changes), deleting the original table, copying tot the original name (possibly with changes), and deleting the TEMP table. Since some kinds of changes are

incompatible with other kinds (for a single pass through the alteration), some functions are disabled and their buttons grayed out when others are started. If two "incompatible" functions are needed, the user is required to make two passes through the process. The generated SQL is shown, and the user can cancel without making the changes. The example in the figure shows the result of some Move Up/Down operations. The Add, Rename, and Copy operations are incompatible and their keys have been grayed out, but Delete is still available. The SQL listing shows everything but the Commit Transaction, which is not sent until the user accepts the action.

The main window displays, and the data edit window were clunky, even by my standards, so my first upgrade after basic functionality was achieved was to switch to tkTable-based display.

Upgraded Display (showtable)

The original impetus for using tkTable was looks—getting a spreadsheet-like display. The expected side-effect was getting a much nicer format for editing and entering data. An unexpected one pleasantly took care of a potential problem with large tables. The original text-based display was populated by reading the entire table into memory and inserting it into the text widget. This worked fine with the data sets I was using, so I never got around to coding something better, but it would not have scaled well. The tkTable widget, in the command data mode, calls a user-specified proc for any cell it needs to display (or update), so the size of the table is of little importance to the display speed. I have thought a little about speeding things up by reading the whole row when the first column is asked for, and then filling in the rest of the row from cached values, but I have not been able to convince myself that some race condition could not result in an updated value being missed—and besides, the current speed is acceptable. The main window now is as shown in Figure 4, which has the standard spreadsheet-like look.

Clicking on a table name now brings up a window like Figure 5, which not only looks better, but also allows updating and adding data.

When updating data, the default is to require confirmation for each cell changed, but there are options for read-only (no changes allowed) or making the change immediately, without confirmation. Also, the capability to extend the number of rows in the table can be set to Never, Confirmation required, or Always done. Finally, selected rows can be deleted.

Stand-alone Functions

This section covers capabilities of GEB that do not interact (at least very much) with the basic display and edit windows that have been discussed so far. They were added over an extended period of time, but will be covered together.

Execute SQL/Tcl

There is often a need to execute a single line of SQL or Tcl, so I created windows for those functions. The result is made a little more readable by allowing the user to specify the number of items per line. Figure 6 shows an example.

Run a Table

More than once I wrote and debugged some Tcl code to process some data in an SQLite file, and then could not find the code when I needed to run it again. I hate doing things twice. A lot. So i started adding tables to the file with a column named text for storing the code snippets.

Naturally, copying the data from these tables and pasting it into one of the Execute windows got old real fast, and so the ability to edit and execute the code was added. I was really worried about the code interfering with GEB, so I experimented with executing the Tcl code in either a safe interpreter or a special namespace. I also included the ability to execute SQL code. None of these special capabilities turned out to be very useful, and this capability has not been updated, but the lessons learned led to the Tcl storing and execution that made it possible to use the database file for storing the program itself.

Import/Export

The CLP falls short of my import/export needs in a few ways. First, in some cases my data has the desires column names as the first line, and in other cases it doesn't. Including the names is an option for both import and export, as is picking the delimiter. Second, in one data set all lines contained the “essential” columns but some also had some “optional” data. Therefore the import routine optionally allows some lines to be missing columns. It would have been easy enough to handle the data manually, but why spend a few minutes doing menial work if you can avoid it with an hour or two of programming?

There was also a separate capability to import a single cell from a text file, or export it, similar to the onecolumn method, but not using it.

Convert between Version 2 and Version 3

This work started during the SQLite Version 2 days, and for a long time I refrained from using any SQLite capabilities restricted to Version 3 so that my code would work with either file format, but I eventually gave in, because of V3's ability to do its own variable substitution.

Anyhow, functions were added to convert from V2 to V3 (because it was needed), and from V3 to V2 (for completeness, and to simplify checking.

Putting the Program in the Database File

When I read DRH's paper “SQLite and Tcl” I was intrigued by the concept of storing the whole program in the database file. I was not yet familiar with Tcl's handling of unknown procs, but the wiki showed how to use and extend that. I had, or at least thought I had, all the other pieces, so the basics were soon cobbled together.

I did not exactly implement DRH's writeup, since I wanted to reduce the number of tables. Therefore I grouped the supporting procs with the main proc for each function and put them in a single table with the name of the main proc.

This had two impacts: There could not be separate columns for the header and body of a proc, and the support procs could not be called from a proc stored in a different table, unless it was known to already have been loaded. The grouping of the procs was done properly, and so the second theoretical impact has never been a problem.

Two unexpected problems did arise, though.

I had never needed to work on more than one file at a time, so I had not implemented ATTACH capabilities. The program could look at just one file, and that file was now the one containing the program itself. No place for the file with the data. A significant rewrite was needed to handle attached files, with a bunch of one-dimensional arrays describing the database schema needing to be redone as two-dimensional arrays, with the attach name as the new index.

Secondly, the simple editing support the program had was no longer enough. I could no

longer do a global search for the name of a proc or variable and see every place it was used. So the scope of the search routine was expanded to cover all tables with a column named tcl.

Figure 7 shows the main editruntcl window, and Figure 8 the search window. The two are tightly integrated, with edits in one making the appropriate changes in the other. In fact, automatic propagation of changes is something I worked hard to achieve, since the display or use of obsolete data could lead to database corruption. The one deliberate exception is the tk_optionMenu widget in the editruntcl window, whose table list must be reset manually by clicking on the button next to it. This was because updating the list affects the display, and I wanted that under user control.

Program Organization

One goal was to put as much of the startup process in the database, so that the external bootstrap would be as small as possible. As can be seen in Figure 9, that effort met with success. The last two lines are optional, and one or the other is usually commented out, depending on the work being done. For program development the editruntcl line is left uncommented, so the main editing window will come up automatically. Otherwise an attachit line brings up a display of the desired data file.

The main_attach routine controls most of the startup process. It must read in and execute two procs to set up the array of table names needed by the extended unknown handler. Refactoring the functionality could reduce the number of procs needed to one, but such strictly esthetic improvements are low priority compared to improving functionality.

Extending the capability of the unknown handler is based on http://wiki.tcl.tk/2776. Whenever an unknown proc name is encountered, the ::GEB::tcltablelist(main) list is searched for that name preceded by main. (so that currently attached files are not used for auto loading). If such a table name is found, the table is loaded. If reading in the table defines a proc with the desired name, that new proc is executed with the appropriate arguments and its result is returned. If either of these conditions is not met the original processor for unknown procs is executed.

All other functions are initiated with menus or buttons.

Future Plans

The development process so far has been so pleasant that I look forward to adding further functionality. In the past it has often taken longer to decided what I wanted to do and how I wanted it to work than it took to implement it, and I fully expect that to be as true in the future.

Two things I am planning to implement soon are allowing wildcards (of the [string match] kind to searches, and some sort of version control, at least to the point of executing a stable version while editing and testing a development version of routines. Further down the road I may look at a GUIfied way of doing Full Text Searches, and setting up tkTable displays of views.

Potential Users and Licensing

GEB is a set of Tcl routines for displaying and manipulating SQLite databases, stored in an SQLite database. As such, it could conceivable appeal to a wide range of users.

At one extreme would be a Tcl developer who didn't care about SQLite. GEB could be used by such a developer, but other environments are more powerful in many ways.

At the other extreme is an SQLite user who doesn't care about Tcl and doesn't want to learn it. This use seems like a bit better fit, and there is less competition, but most would still find other programs more to their liking.

GEB is most likely to appeal to those in the middle: those with some SQLite data and some Tcl scripts for processing the data, who want to keep, use, and maintain them together.

I intend to release GEB to the public domain, since it would feel wrong for my contribution to have a more restrictive license than SQLite itself. Therefore, GEB may be used for any purpose whatsoever. Development is encouraged, and especially if the new code is also public domain.

Support is available—inquire within.

Gerald C. Snyder
mesmerizerfan@gmail.com

Figures

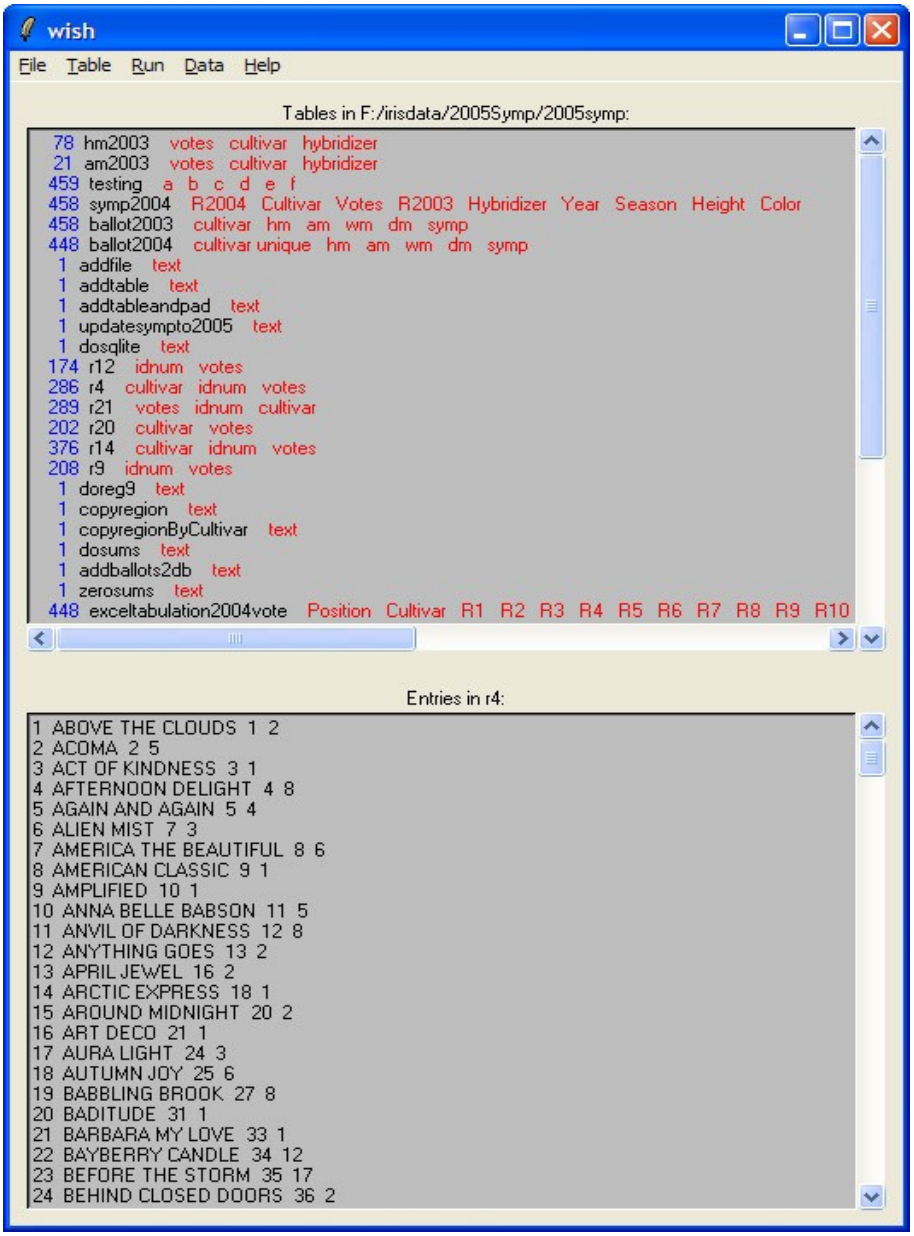

Figure 1. Pre-tkTable Main Display Window

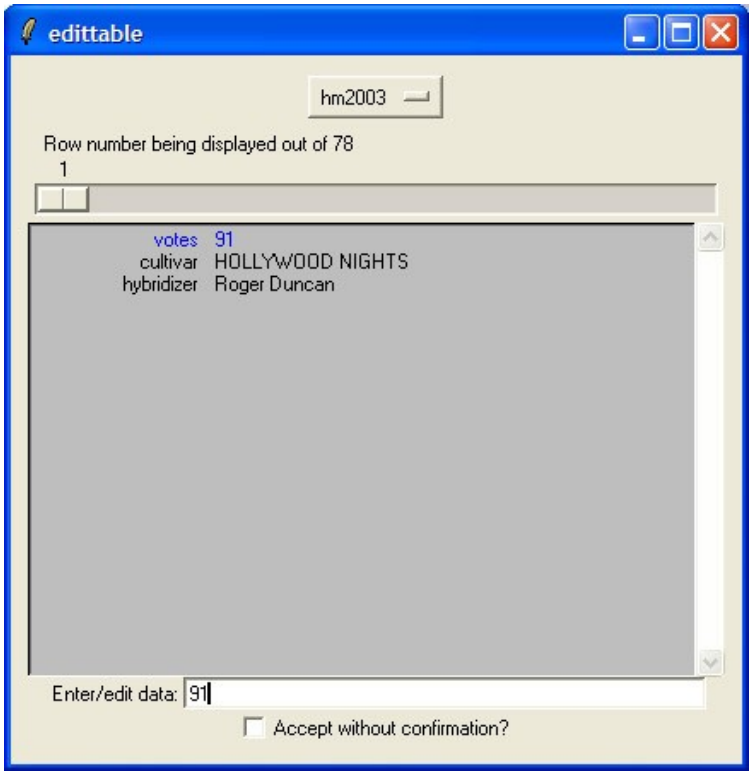

Figure 2. Pre-tkTable Edit Window

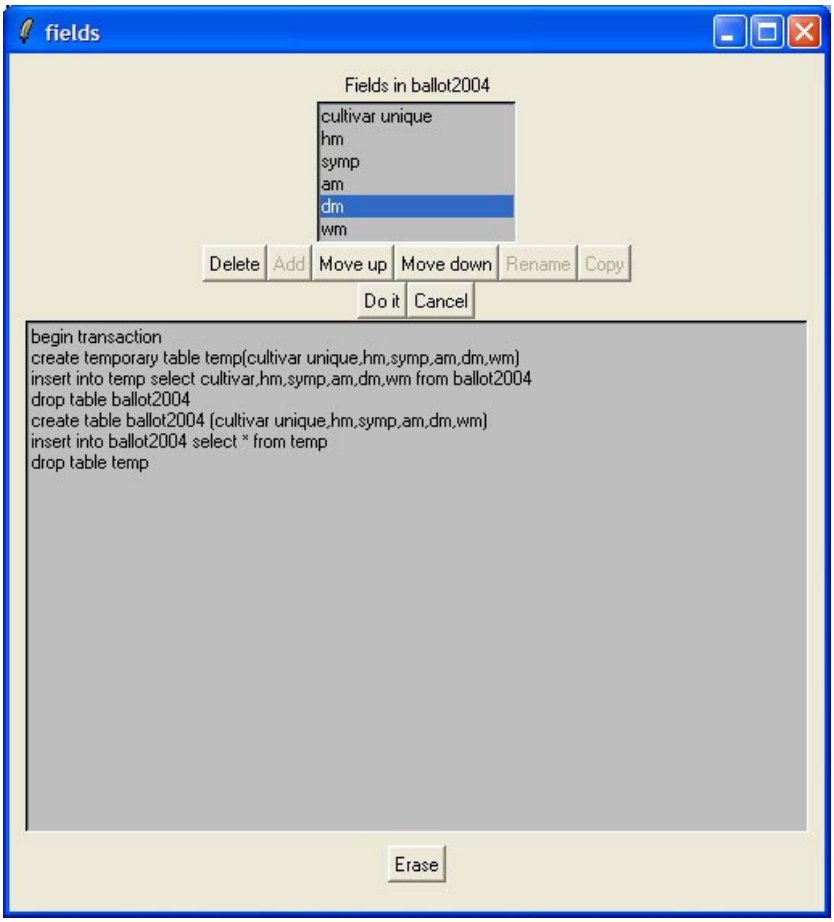

Figure 3. Alter Table Window

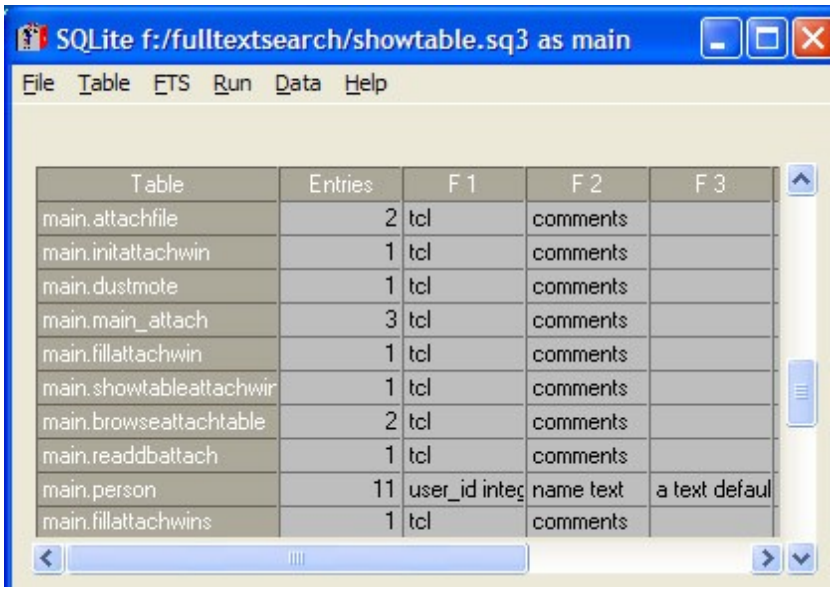

Figure 4. Main Window with tkTable

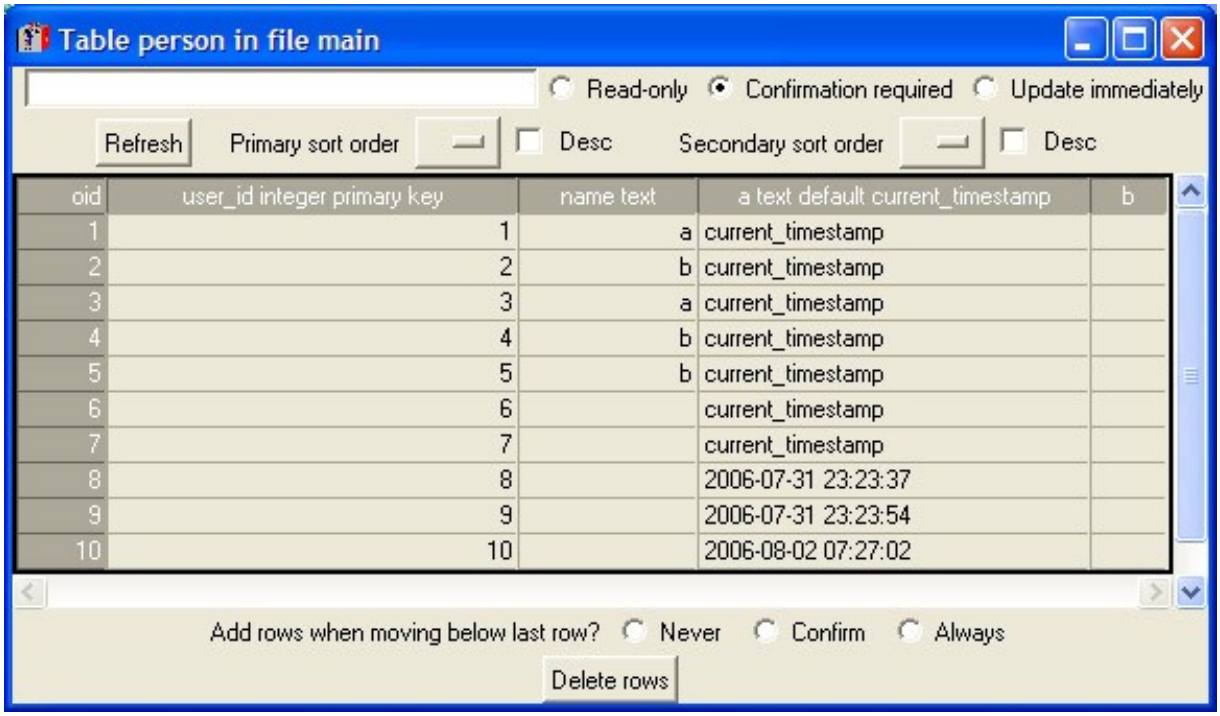

Figure 5. Table Display with tkTable

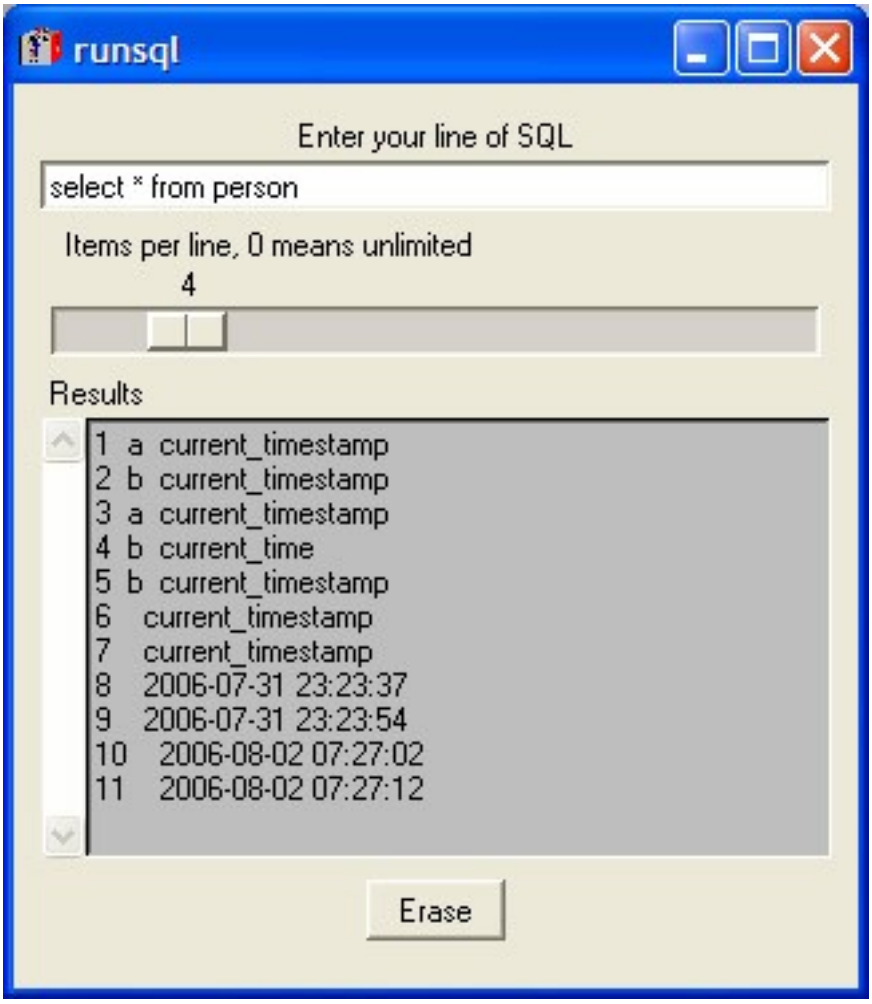

Figure 6. Window for runsql

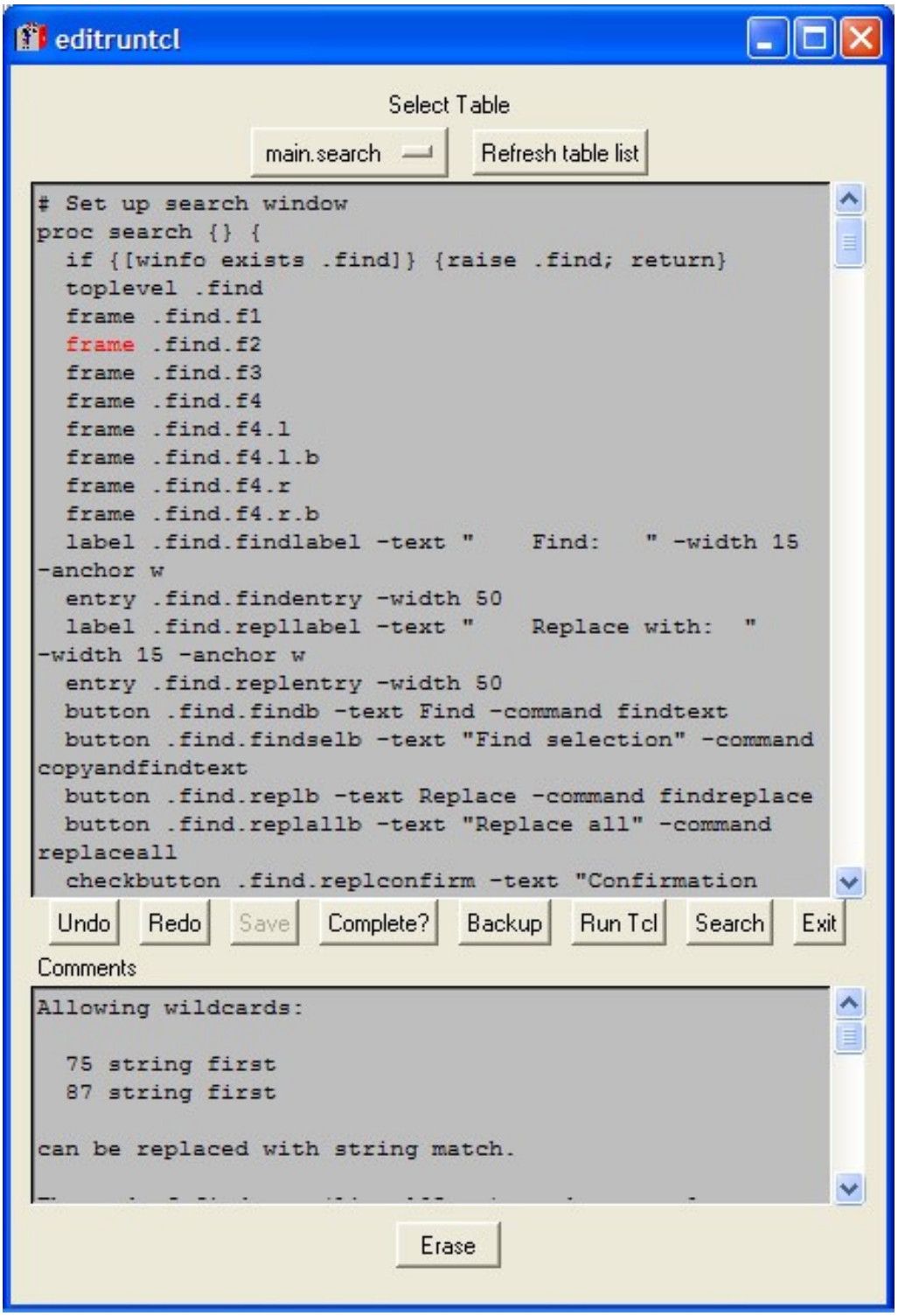

Figure 7. Window for editruntcl

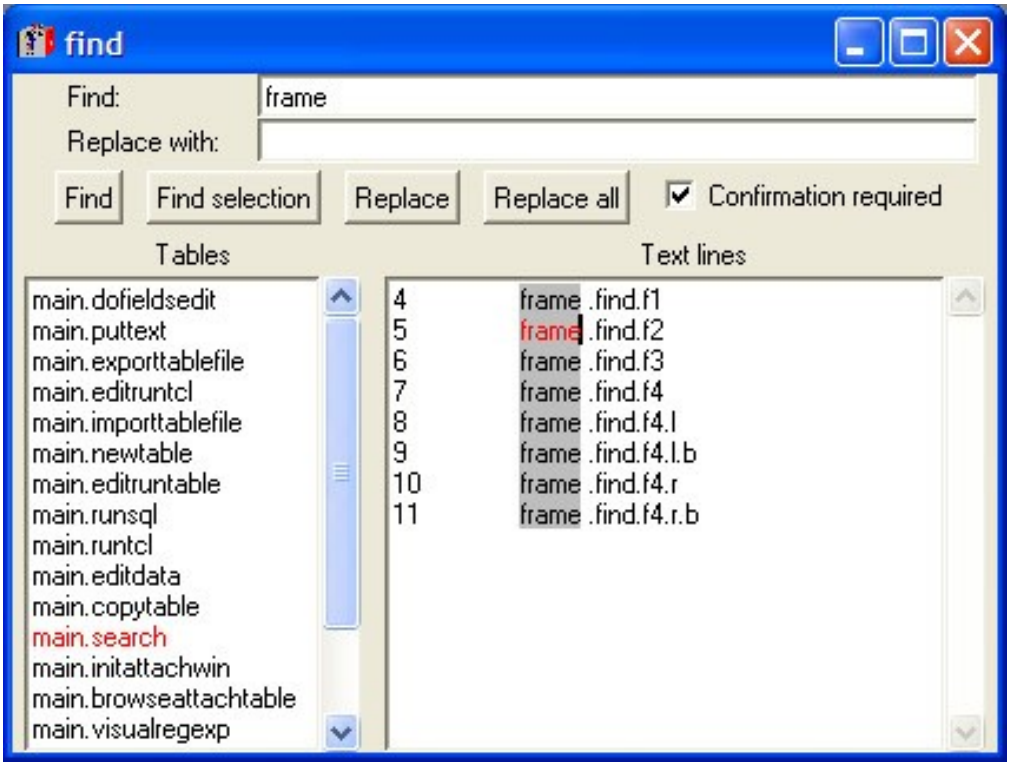

Figure 8. Search Window

```
package require Tk; package require Tktable; console show
# Create namespace for all "globals" and procs
namespace eval ::GEB {}
 load /sqlite/tclsqlite3.dll
 load /sqlite/tclsqlite.dll
set dbfile showtable.sq3
 sqlite3 sq $dbfile
 set ::GEB::attachfilename(main) $dbfile
proc evalsqlitetcl table {
  uplevel #0 [join [sq eval "select tcl from $table limit 1"]]
}
evalsqlitetcl main_attach
editruntcl
# attachit s2008 /irisdata/2008symp/2008symp.sq3
```

Figure 9. GEB Bootstrap