SQLiteManager

VERSION 2.0

REVISION 2

Contents

Introduction

Overview of SQLiteManager

SQLiteManager is a “next generation” GUI database manager for sqlite databases, it combines an incredible easy to use interface with blazing speed and advanced features.

SQLiteManager allows you to open and work with sqlite 2.x, sqlite 3.x, in memory databases and REALSQL Server databases. It allows you to create and browse tables, views, triggers and indexes. It enables you to insert, delete and updates records in a very intuitive way and it supports you arbitrary SQL commands.

SQLiteManager also has a flexible report generator whereby you can create virtually any text report (including HTML and XML) using a powerful template language.

Each database that you open with SQLiteManager is presented in one main window with a tollbar with eight different panels: Design, Manage, SQL, Analyze, Verify, Optimize, Vacuum and Setting. This user’s manual will cover each panel in detail.

In addition to the main window, SQLiteManager provides a number of functions that you access through its menus and buttons. This user’s manual will cover every one of these functions in detail, but you might want to take a minute to browse all SQLiteManager’s menus to familiarize yourself with everything it can do.

Registering

The unregistered version of SQLiteManager runs with certain limitations. Any query will return no more than 20 rows and you will not be able to export and import any data into a database.

You will also not be able to convert, dump and generate reports on any database until you have registered. If you have a serial number for SQLiteManager, enter it into the preferences dialog, which you can reach through the Preferences menu item.

To became a registered user and receive technical support and updates you must register SQLiteManager via web at the address: http://www.sqlabs.net/store.php

SQLite2 and SQLite3

SQLiteManager can open and create both SQLite 2 and SQLite 3 databases. When you open a SQLite database, SQLiteManager will automatically determine whether the database is SQLite 2 or SQLite 3 and open it accordingly. The database version will be displayed in the window's title bar.

To create a new database, choose either SQLite 2 or SQLite 3 from the New menu (under File). SQLite 3 is the default if you are creating a database using the keyboard shortcut for new documents.

SQLiteManager also has the ability to convert SQLite 2 databases to SQLite 3. Please see the section on converting databases for more information.

Encodings

By default, SQLiteManager displays all query results in both the Manage and SQL tabs as UTF8. To change SQLiteManager's text encoding, choose a new encoding from the Database menu (under the Encoding sub-menu) The new encoding only affects the database in that window. Databases in other windows are unaffected.

When you change the text encoding with the encodings menu, the new text encoding takes effect immediately for data in the Manage and SQL panels. Also, if you add or edit records in the Manage tab, your data will be inserted into the database with the encoding in effect at that time.

InstantQuery Technology

InstantQuery is a new technique for retrieving rows from an SQLite 3 database query at “lightning speed”. Thanks to InstantQuery you can now browse records from a sql query containing millions of rows in a fraction of second.

The main idea behind the InstantQuery technology is that you can have a hidden thread that perform the hard work while you are free to play with the results of your query near instantaneously. This method not only enable you to gain blazing speed but also enable you to display millions of row with very little memory requirements.

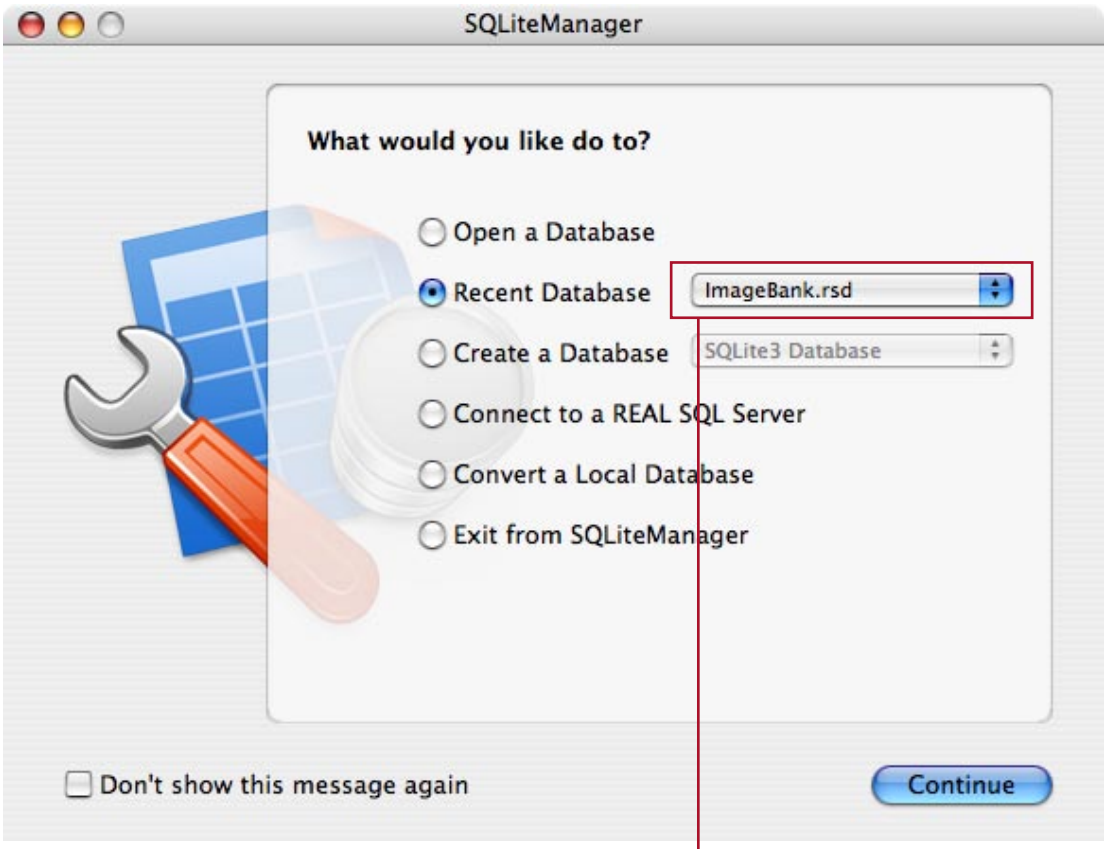
As a real world example, we created an SQLite 3 database with 10 columns and 200,000 rows (database size was about 200MB) and we tried to perform a “SELECT * FROM myTable” query.

We tried two of the top sqlite managers application availables from other companies and both fails with with this query (with an out of memory error, it seems that they both try to load the entire RecordSet in memory).

With InstantQuery (thanks to SQLite3ProfessionalPlugin) you can start receiving results in about 0.2 seconds and memory requirements for this operations will always be near zero.

	SQLiteManager	**Others**
Memory usage:	8 MB	200 MB
Query Time (sec.):	less than a second	some minutes

Startup Wizard



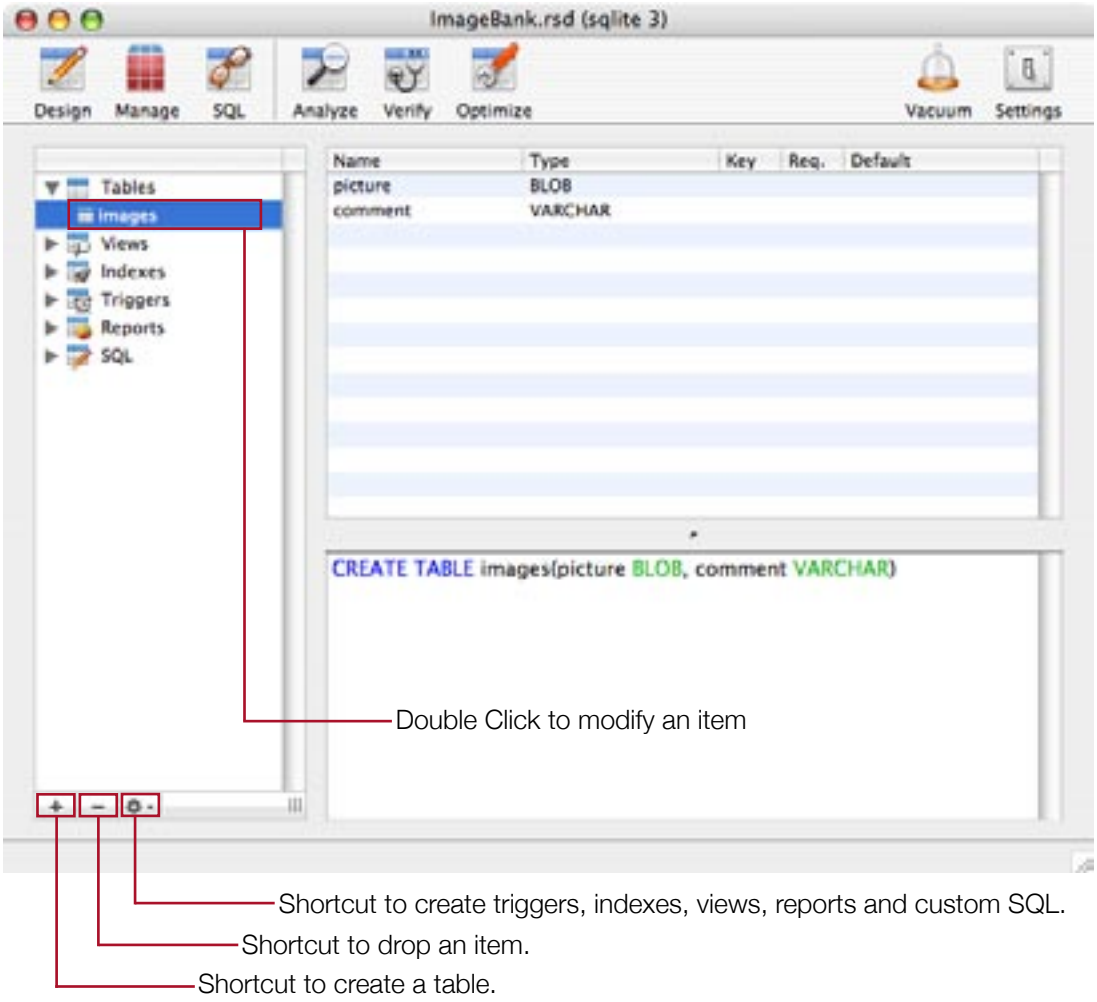
A separator divides file based databases (listed firts) from REALSQL Server databases.

At startup SQLiteManager allows you to peform frequently used operations in a very quick way. With the Startup Wizard you can:

- Open a existing database
- Open a recently used database (max items specified in the Preferences)
- Create a new database (sqlite2, sqlite3 or in-memory)
- Connect to a REALSQL Server
- Convert a local database
- Exit from SQLiteManager

On MacOS X you can hide the wizard at each startup.

Design Panel



The Design panel is where you create, inspect and update tables, views, indexes, triggers, report templates and saved SQL in a database. The tab is divided into two basic areas: an object browser on the left, and a space on the right where information about selected objects is displayed. The object browser is presented as a hierarchical list divided into Tables, Views, Indexes, Triggers, Reports, and SQL. Opening any one of those categories displayes the objects of that type that are in the database. If a category doesn't have any objects, then no objects will be displayed. Selecting an item in a category reveals details about that item in the space to the right of the object browser.

For example, selecting a table displays the schema for that table in a read-only edit field, and a table with more details about that table's fields and the properties of those fields. Field properties include the field's name and type, whether the field is a primary key, whether the field is required (can't be NULL), and the default value for the field. Selecting a view displays information about that view that is very much like the information that is displayed for a table. Note that the schema field for tables, views, and indexes is read-only, but you can copy the text in the field and paste it elsewhere.

We will have more to say about report templates and saved SQL later, but it is important to realize that in order to implement those two features, it is necessary for SQLiteManager to save information in a special table in your database file. The table is called "sqlitemanager_extras" and it contains all of the extra information that SQLiteManager may need to keep about a particular SQLite database.

SQLiteManager endeavors to filter that table name out of the places where table names are displayed. For example, you will not find a table by that name in the list of tables in the Design tab. But it is still possible to interact with that table directly by typing SQL queries and commands into the SQL tab. We recommend that you do not delete the "sqlitemanager_extras" table, as you may lose functionality by doing so. On the other hand, deleting that table should not interfere with the rest of your SQLite database objects in any way. While in the Design tab, you may wish to drop tables, indexes, or views. To do that, select the object you wish to drop from the object browser and then choose "Drop" from the Edit menu or just push the "Delete" button at the bottom of the Object browser table.

Creating Tables

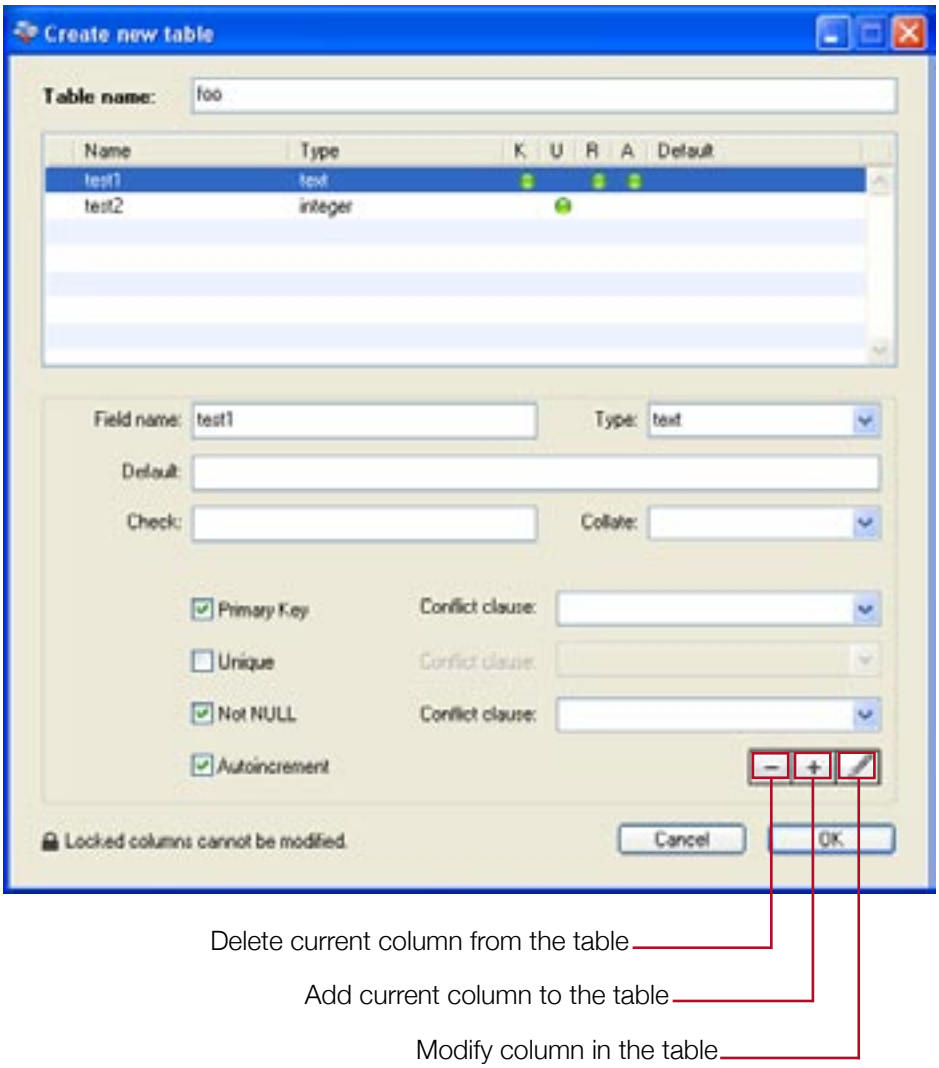
To create a new table in SQLiteManager, push the "Add Table" shortcut button at the bottom of the object browser.

You will be presented with a new table dialog, pictured below. Within the new table dialog, you can name your table, and then assign fields to it. You may also set the properties of each field, such as whether a field is a Primary Key, Required (values in the field can't be NULL), Unique (every row in the table must havea different value for the field) or Autoincrement.

You may also specify a default value for the field. If a field has a default value and you don't specify a value for that field when you insert a new record into the table, the field will be set to the default value automatically. Field types may either be typed in or set with the popup. The popup has the field types that SQLite and REALbasic understand. But because SQLite is fundamentally type-less, you are free to type in just about any type you'd like.

The optional conflict-clause following each constraint allows the specification of an alternative default constraint conflict resolution algorithm for that constraint. The default is ABORT. Different constraints within the same table may have different default conflict resolution algorithms.

Design Panel



The same dialog in used when you need to alter a table (not available for SQLite 2 databases).

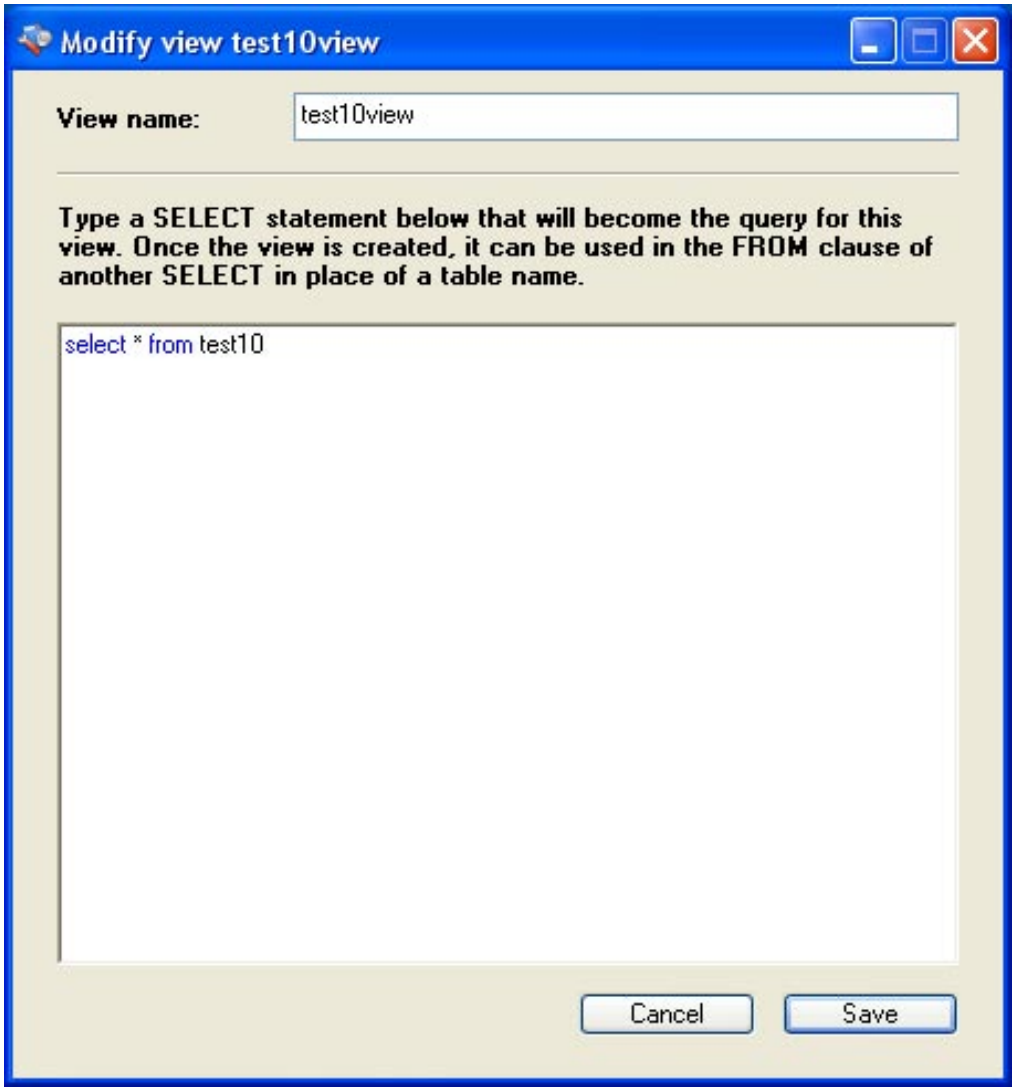
Current SQLite's version of the ALTER TABLE command allows the user to rename a table or add a new column to an existing table. It is not possible to remove a column from a table. If the table being renamed has triggers or indices, then these remain attached to the table after it has been renamed. However, if there are any view definitions, or statements executed by triggers that refer to the table being renamed, these are not automatically modified to use the new table name. If this is required, the triggers or view definitions must be dropped and recreated to use the new table name by hand.

Creating Views

To create a new view in SQLiteManager, choose Create View from the Action button (at the bottom of the object browser). You will be presented with a new view dialog, pictured above. A view is basically a saved query (SELECT SQL statement, in other words).

SQLite treats views very much like read-only tables. You can query them as you would a table, and you can include them in other queries as well. You are free to name your view anything you would like.

You may then type any arbitrary SELECT statement into the query field and that SELECT statement will become the query for the view.

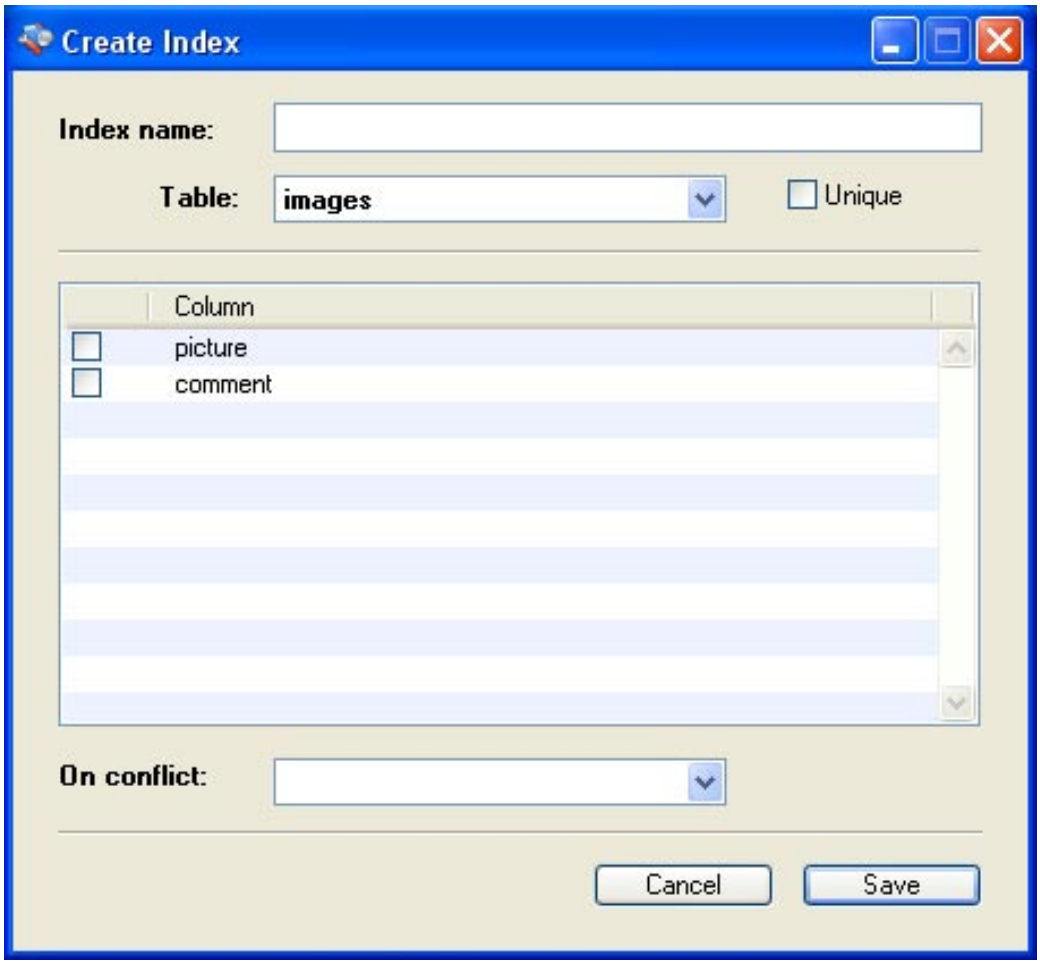


Creating Indexes

SQLite supports multiple indexes on a table and multiple fields in an index. To create a new index, choose Create Index from the Action button (at the bottom of the object browser). You will be presented with a new index dialog, pictured above.

The dialog allows you to name the index and choose the table for which the index will be created. Once you have chosen a table from the table popup menu, the ListBox will display the fields for that table. Check all fields that you wish to include in the index.

You may also indicate whether or not the index is unique by checking the Unique checkbox [Indexes cannot be modified].



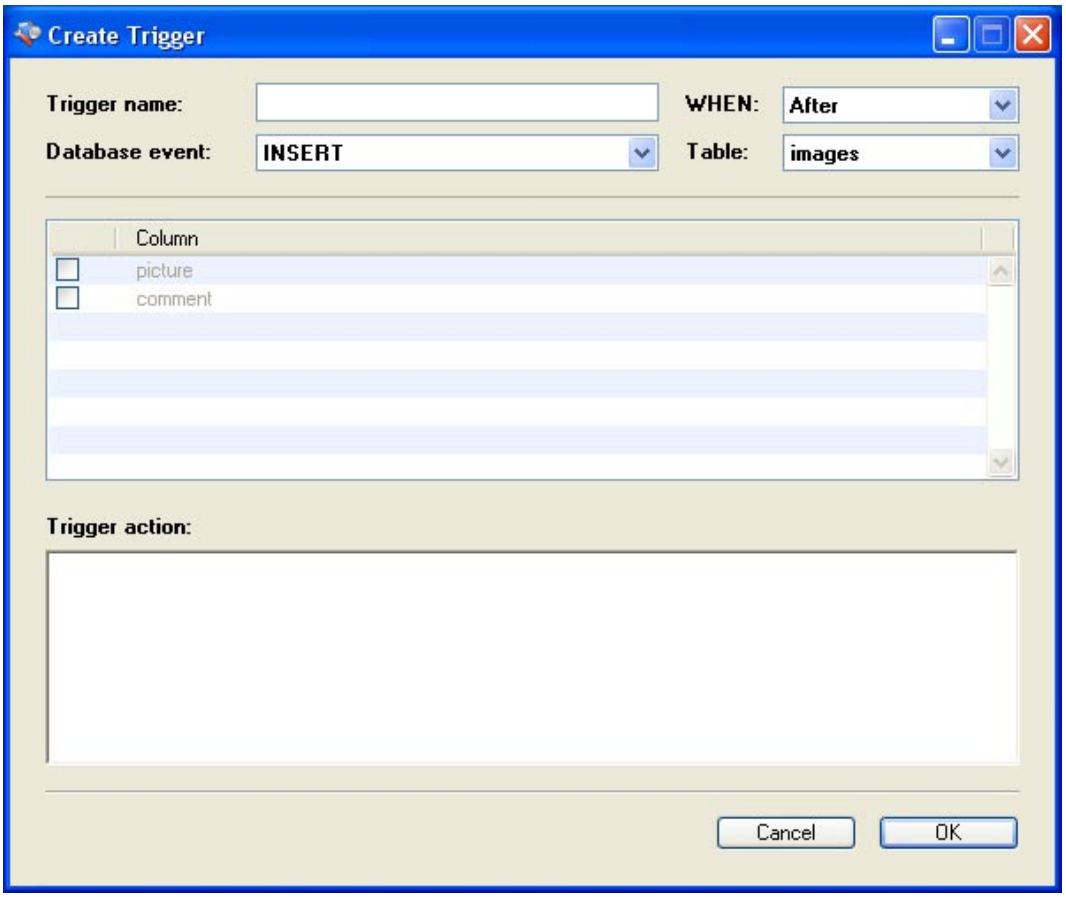
Creating Triggers

SQLite fully supports triggers. To create a new trigger, choose Create Trigger from the Action button (at the bottom of the object browser) and you will be presented with a new trigger dialog, pictured above.

Triggers are database operations (the trigger-action) that are automatically performed when a specified database event (the database-event) occurs.

A trigger may be specified to fire whenever a DELETE, INSERT or UPDATE of a particular database table occurs, or whenever an UPDATE of one or more specified columns of a table are updated.

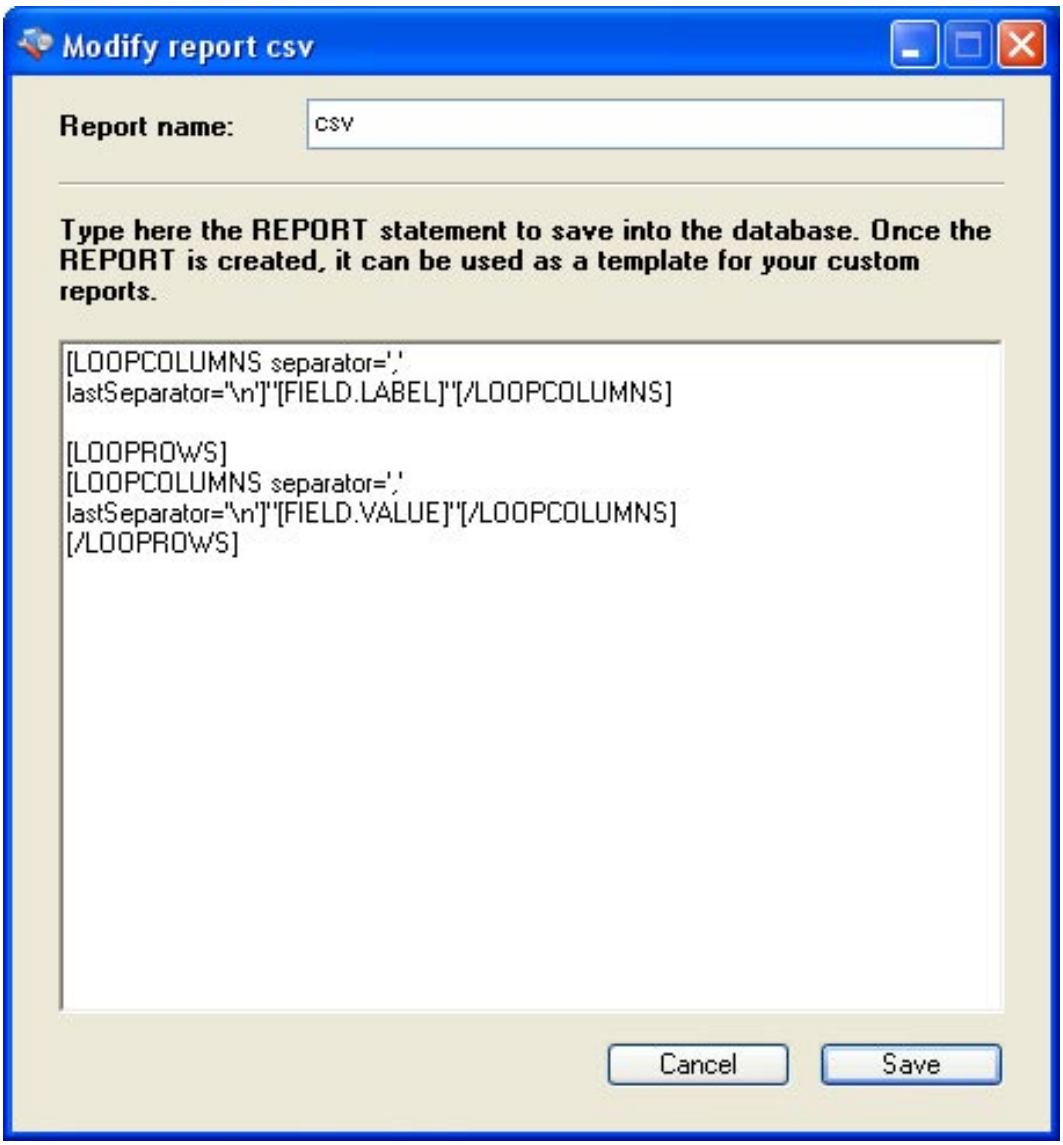
At this time SQLite supports only FOR EACH ROW triggers, not FOR EACH STATEMENT triggers. Hence explicitly specifying FOR EACH ROW is optional. FOR EACH ROW implies that the SQL statements specified as trigger-steps may be executed (depending on the WHEN clause) for each database row being inserted, updated or deleted by the statement causing the trigger to fire.



Creating Reports Templates

We will talk more about generating reports in a later section, but before you can generate any reports, you have to create a report template. To create a report template, choose Create Report from the Action button (at the bottom of the object browser). You will be presented with a new report dialog, pictured above.

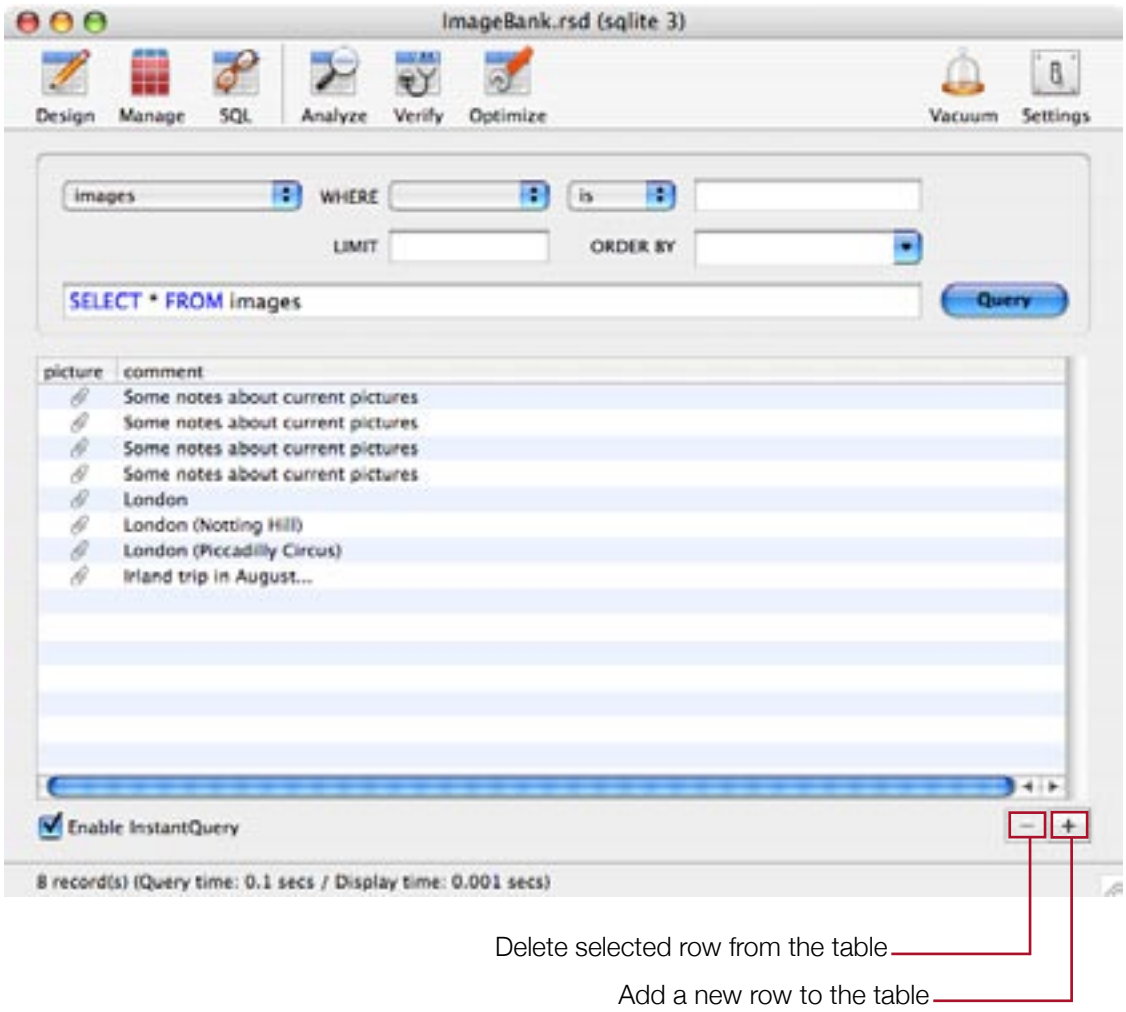
The new report dialog allows you to enter a name for the report template, and then to enter the text of the template itself. You can edit the report template after you have created it by double cliking the template in the object browser (in the Design tab).



Manage Panel

You will visit the Manage panel when you want to insert, remove, or edit records in a table. The tab is divided into three main areas. At the top of the tab are the popup menus and edit fields for building a query. Below that is a read-only edit field where the SQL for the query is displayed. You are free to copy this SQL and paste it elsewhere, but you cannot type arbitrary SQL into that field.

Below the edit field is a list box containing the results of a query. To build a query, simply construct a SELECT statement out of the popups and edit fields and click the Query button. The query results will be displayed in the list box. To remove a record from the queried table, select it and click the Remove button. To edit a record, double-click it in the list box and the Edit Record dialog will appear.

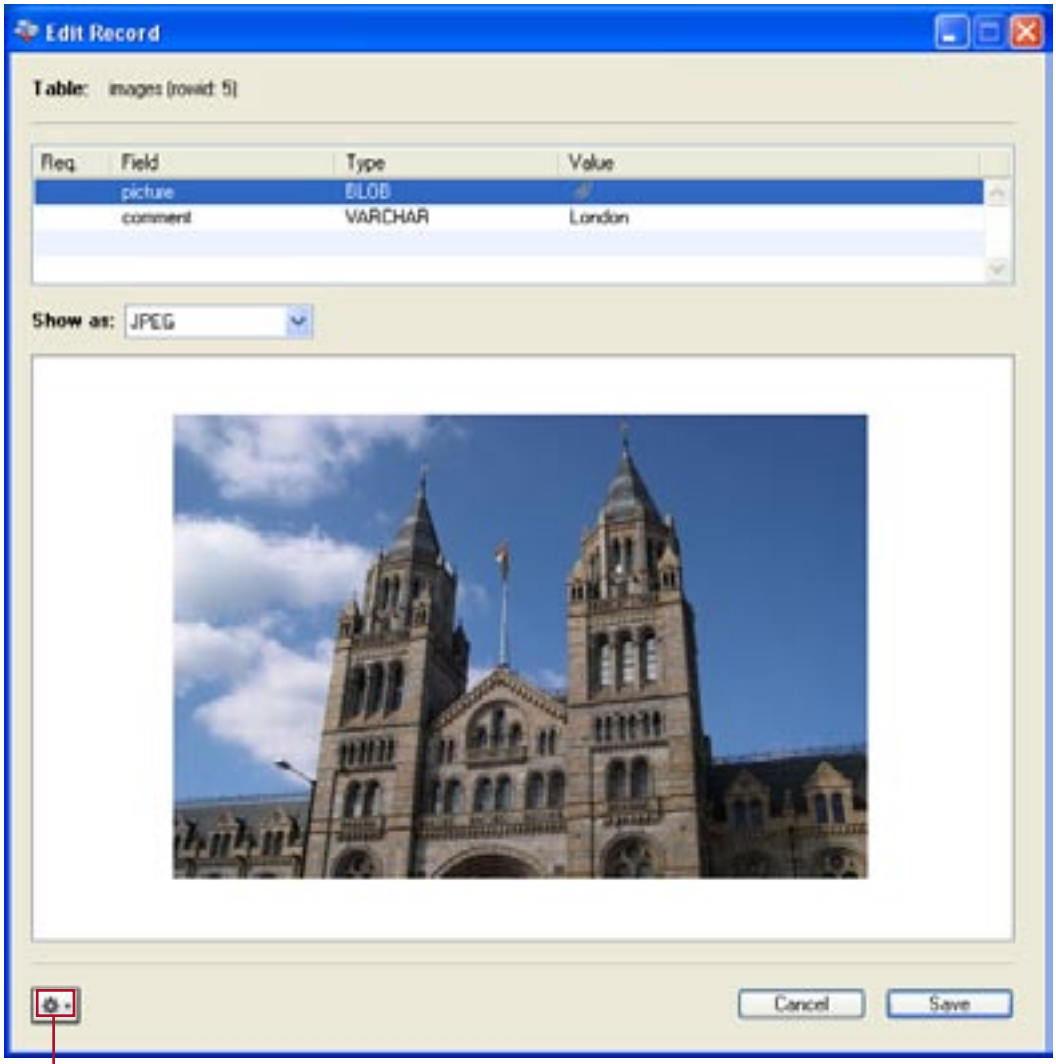


When you open a record to edit it (just double click on it), the fields of the record will be presented in a list box, with a text box below to edit the values of those fields. Make any changes you would like and click the Save button. The changes will be saved in the database immediately. Or, you can click Cancel and the changes will be discarded.

Manage Panel

This powerful dialog gives you the option to display current fields as TEXT, various graphical formats like JPEG, TIFF, BMP and so on (it depends on the platform) and to show it as a raw BinHex image.

To insert a new record into a table in the database, click the Insert button in the Manage tab of SQLiteManager's main window. The dialog used for inserting records is exactly the same as that used for editing records.

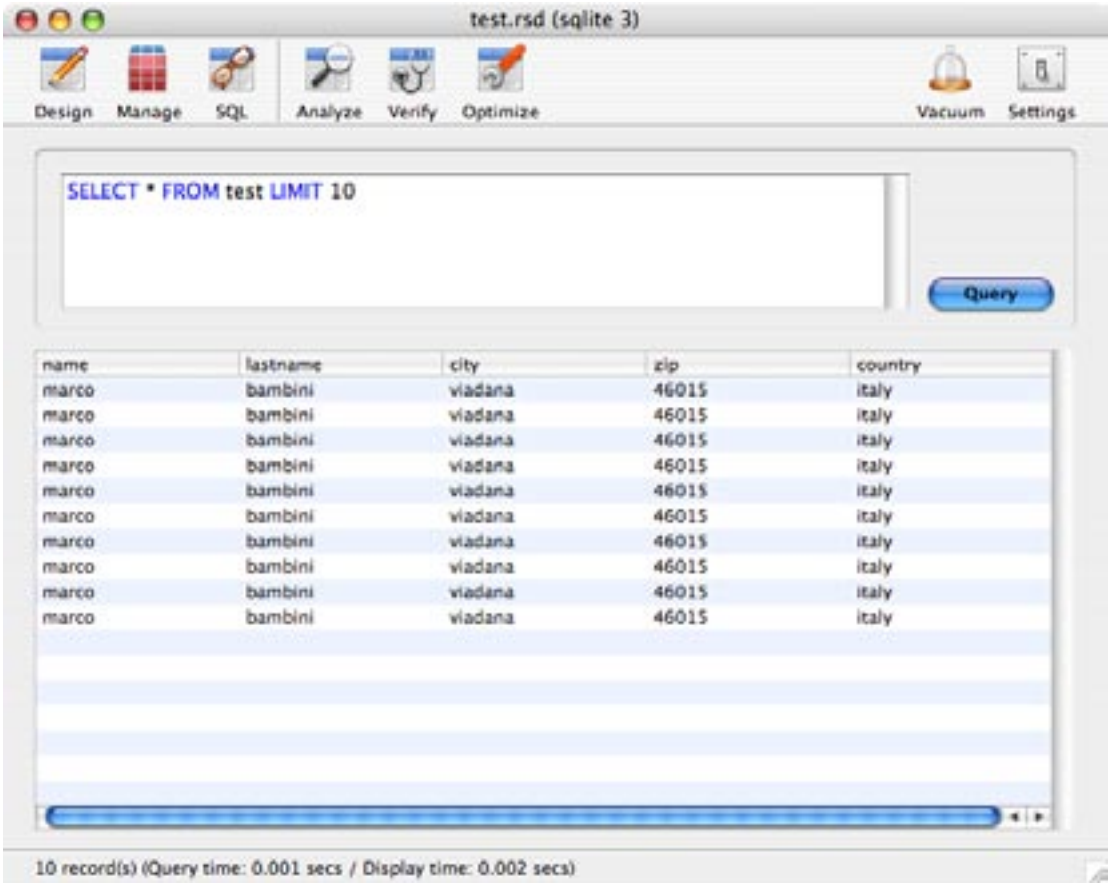


The action shortcut gives you a quick way to set current field to NULL, to current DATE ot to current TIMESTAMP. It gives you also the option to load/save current field from/to a file.

SQL Panel

You will visit the SQL panel when you need to type arbitrary SQL to perform complex queries and commands on an SQLite database. The tab is divided into two main areas: a large edit field for typing SQL at the top of the tab and a list box for displaying results at the bottom of the tab. To use the SQL tab, just type any SQL into the edit field and click either the Execute/Qeury button. If you want to display the results from such a command you must click the Select button.

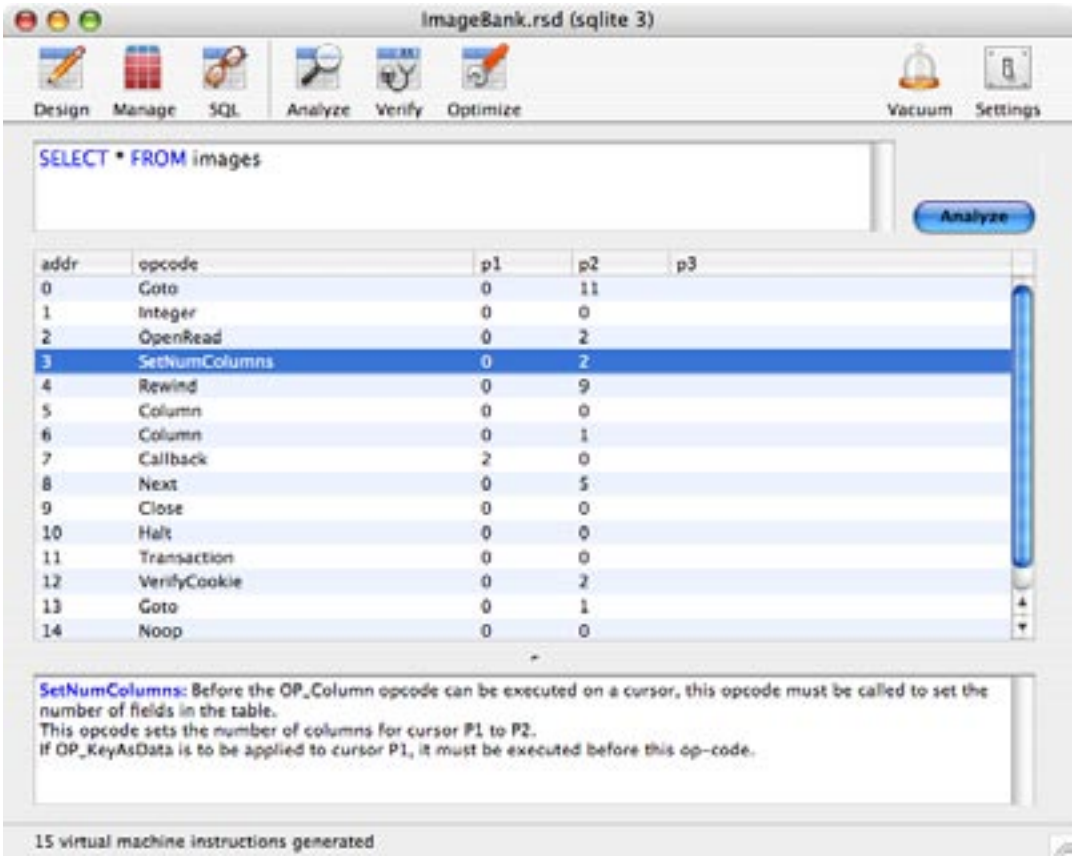
If you type an SQL statement that you'd like to save to use another time, just click the left mouse button and a popup menu will appear allowing you to save the SQL in the database or retrieving a previously saved one .



Analyze Panel

You will visit the Analyze panel when you want to examine and disassemble SQL commands into low level virtual machine instructions.

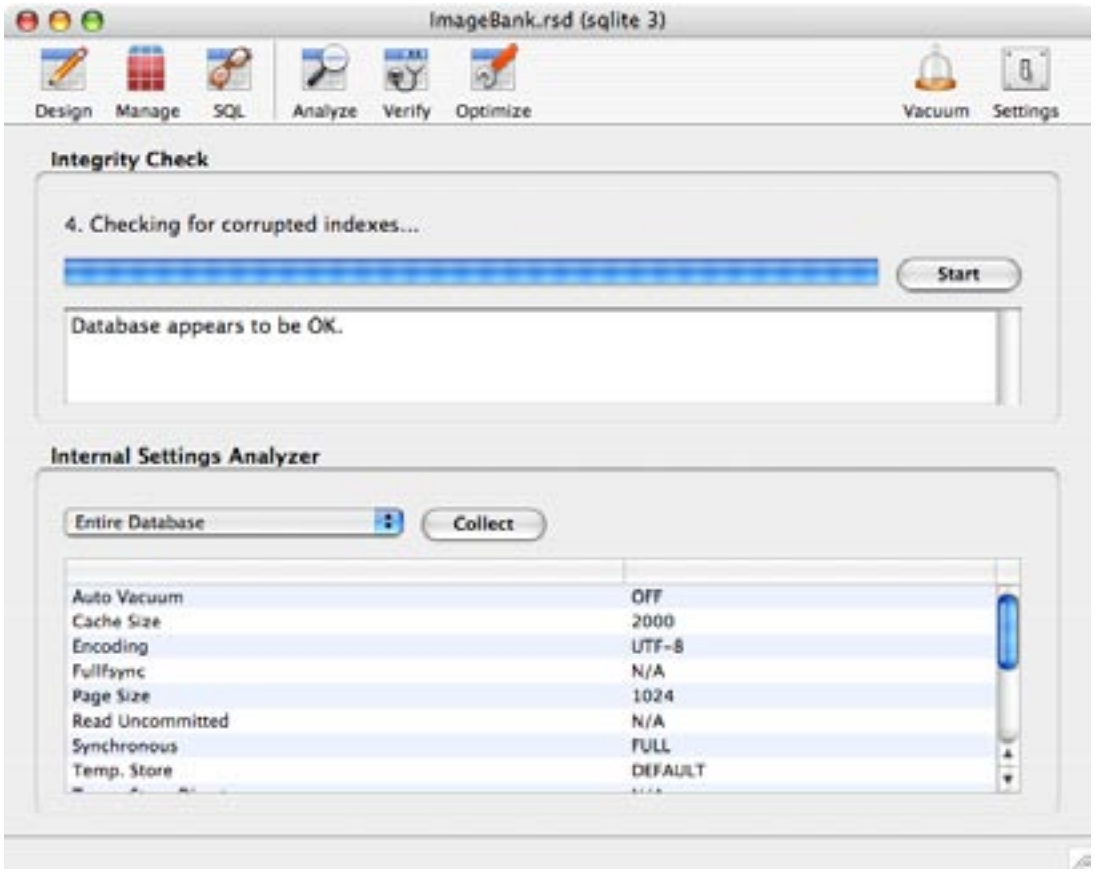
This is the most powerful way to check if a query or a SQL statement is efficient or if it can be rewitten in a better way. Each virtual machine opcode has a very detailed description at the bottom of the window.



Verify Panel

You will visit the Verify panel when you want to perform a sanity check over your entire database or when you want to examine all the most important internal database settings in a very easy way.

The Integrity Checker checks for out of orders records, for missing pages, for malformed records and for corrupted indexes. A detailed reports is shown if some problems are found inside the database.

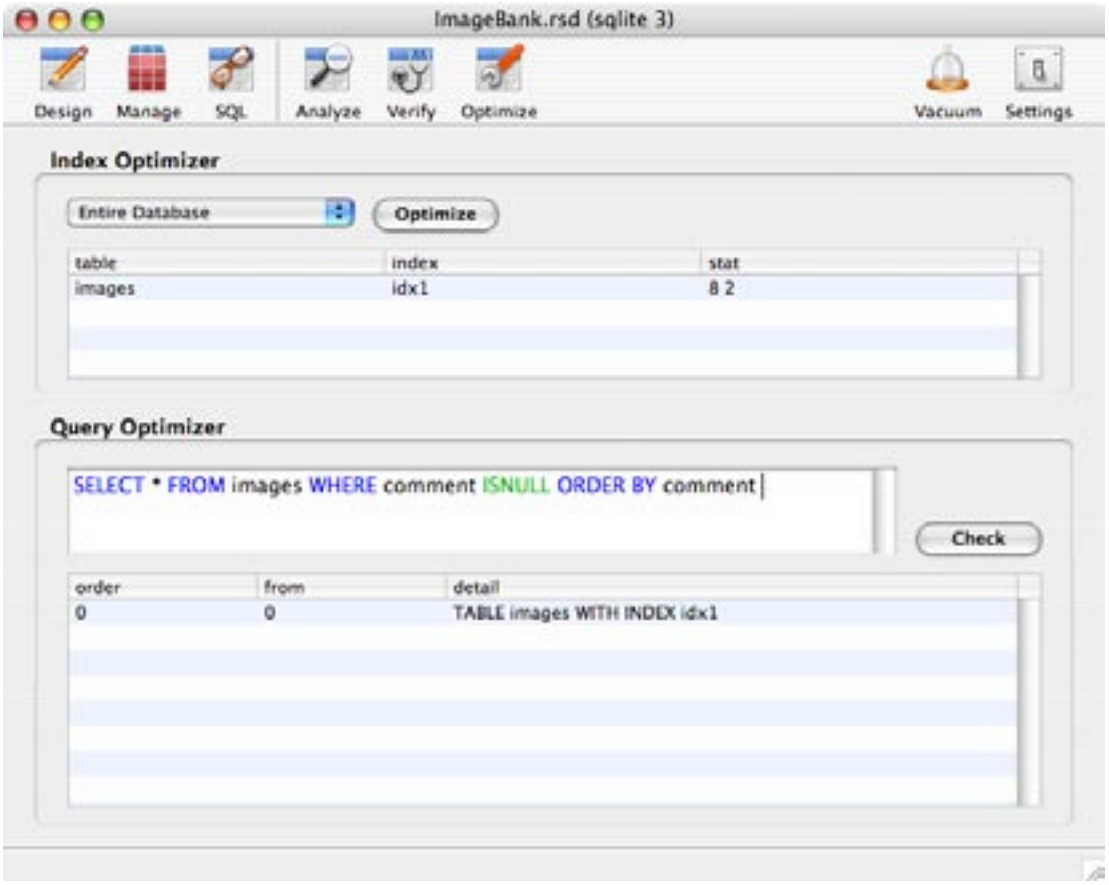


Optimize Panel

You will visit the Optimize panel when you want to find out a way to optimize your database or your queries. The panel is divided in two main areas:

The Index Optimizer that gathers statistics about indices and stores them in a special tables in the database where the query optimizer can use them to help make better index choices.

The Query Optimizer that enables you to analyze which index is used when you perform a query and its order if more than one index is used. Its output tells you how sqlite is scanning the tables and indexes to implement your query.



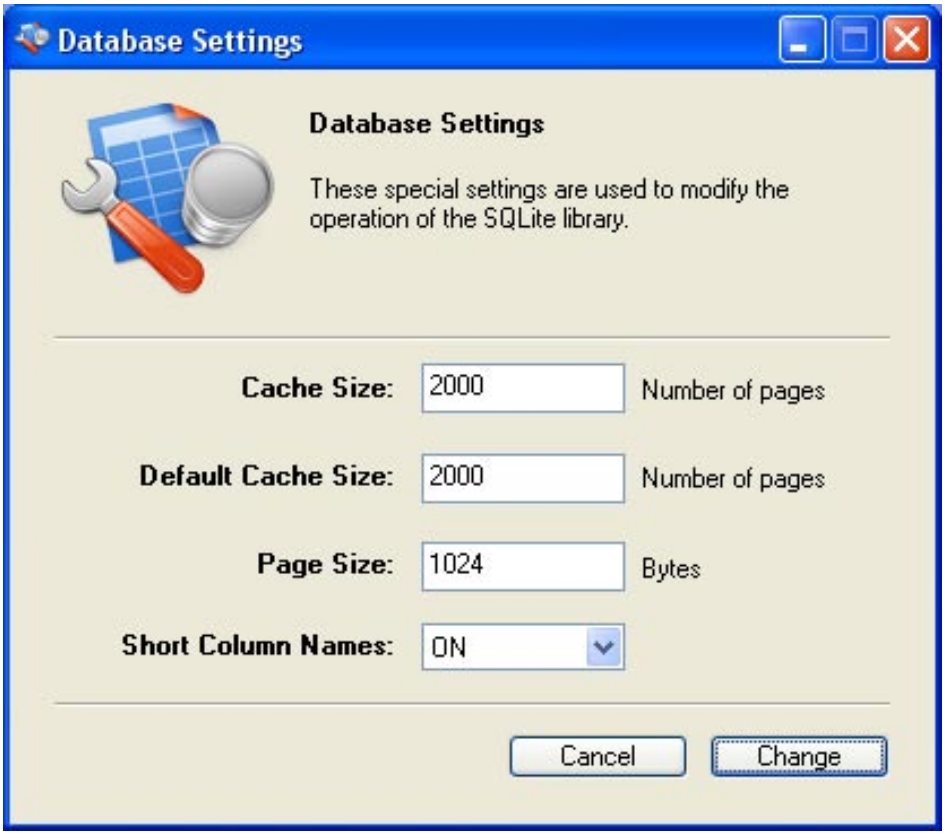
Vacuum and Settings

Vacuum

When an object (table, index, or trigger) is dropped from the database, it leaves behind empty space. This makes the database file larger than it needs to be, but can speed up inserts. In time inserts and deletes can leave the database file structure fragmented, which slows down disk access to the database contents. The VACUUM command cleans the main database by copying its contents to a temporary database file and re-loading the original database file from the copy. This eliminates free pages, aligns table data to be contiguous, and otherwise cleans up the database file structure.

Settings

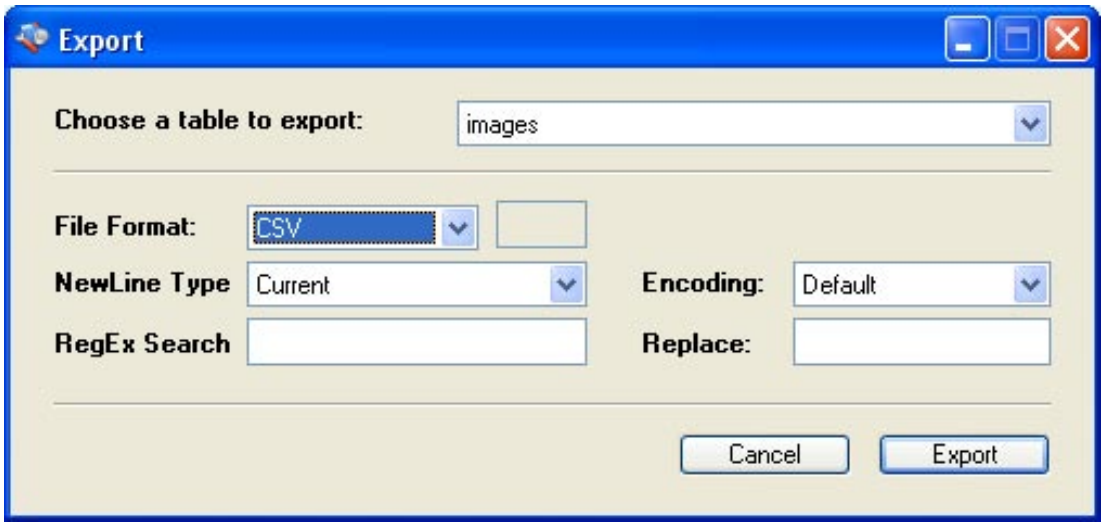
You will visit the Setting panel when you want to inspect or change the most important internal database settings. Please make sure to completly understand what are you trying to change because these settings can really affect your database performance.



Exporting Data

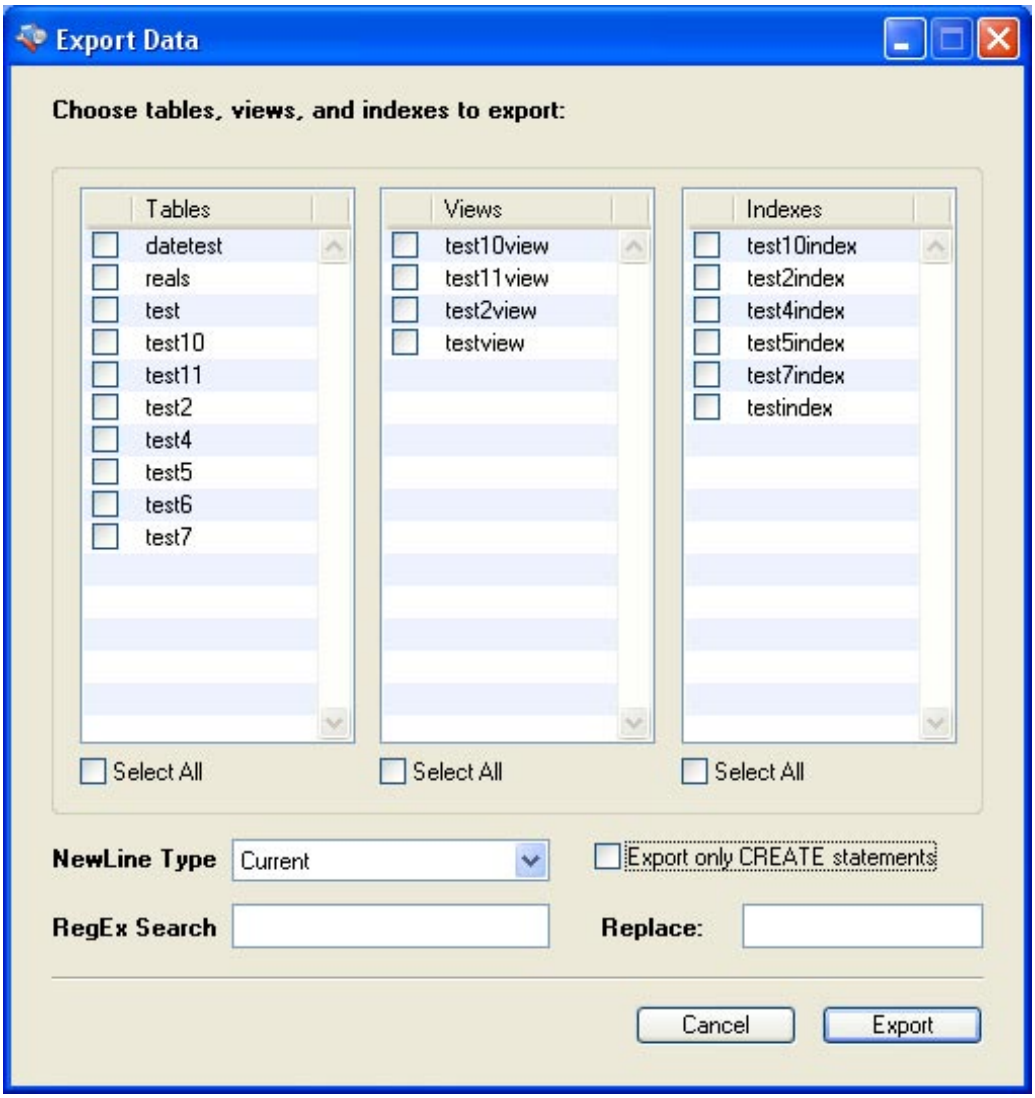
SQLiteManager can export data in several common formats. The currently supported formats are CSV, SQL, tab-delimited and custom character delimited. To export data from an open database, choose an export format from the Export hierarchical menu in the File menu.

Choosing “Others” exports displays a dialog asking which table in the database to export, which format to use, which encodings and so on. Every types of exports include the table headers as the first row of the exported data.



Choosing a SQL export will display the dialog pictured above. Select which tables, views, and indexes you wish to export. You can choose to export only the CREATE statements by unchecking the “Export data” checkbox.

You can also select the new line type (MacOS, Windows, Linux or Current) and apply a special RegEx search/replace pattern to “transform” your data before exporting.

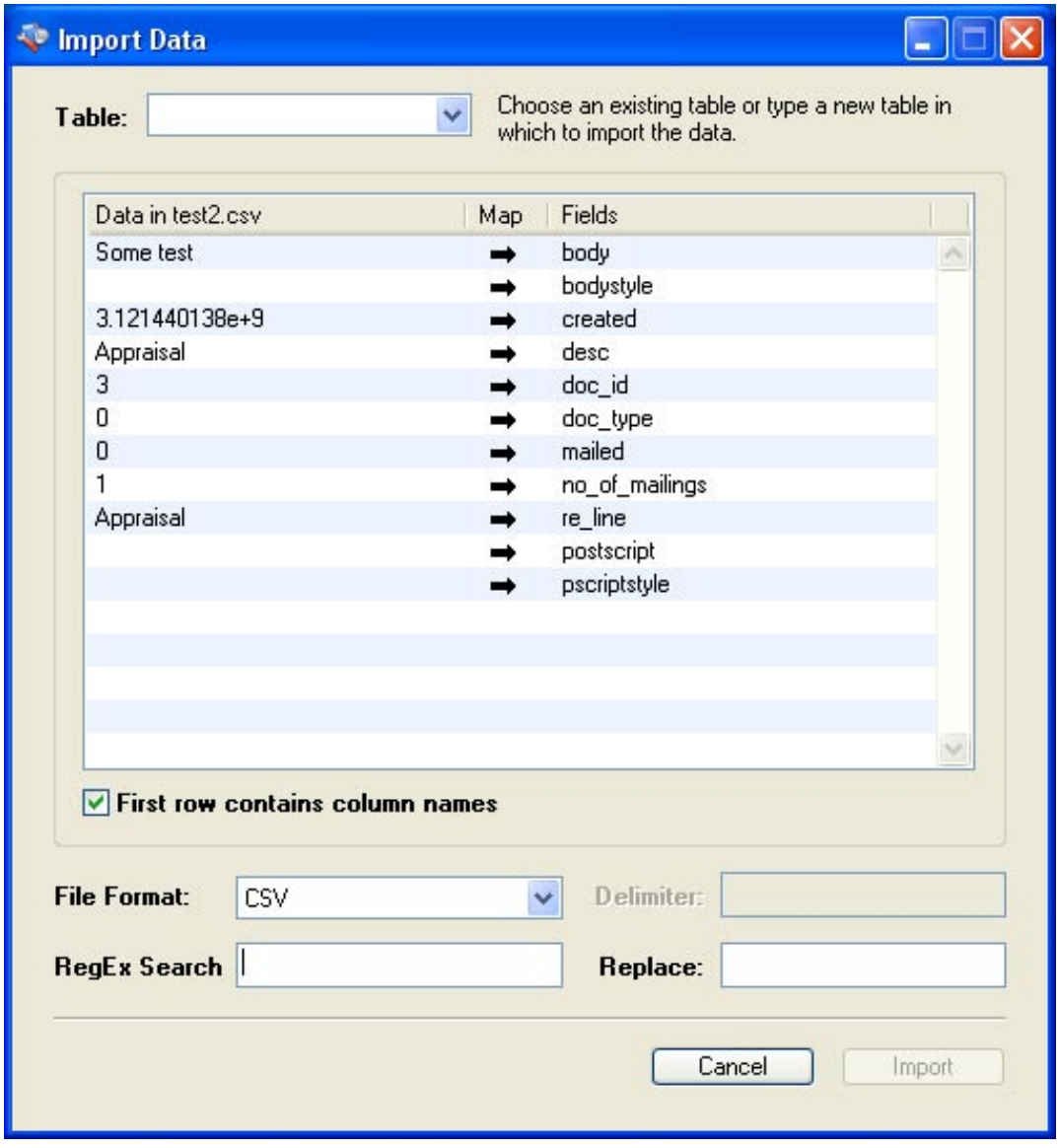


Although SQLiteManager can export in CSV, SQL, and tab-delimited formats automatically for you, you have virtually unlimited exporting options when you use a custom report template. See the section on report templates for more information.

Importing Data

To import data to an open database, choose an import format from the Import hierarchical menu in the File menu. If you choose “Others” you will be prompted both for a file of data of the appropriate format and a table name into which to import the data.

If you choose to import SQL, then you will not be prompted for a table in which to import the data, as SQL statements that insert data into tables refer to those tables directly. Any SQL statements can be in the file you import, including statements to create tables and indexes and such. If you wanted to recreate an entire database, you could create a new, empty database in SQLiteManager, and then import the SQL that would create and populate all of the tables in that database.



Converting Databases

SQLiteManager can convert all three types of REALbasic databases (old pre-5.5 databases, new single-file databases, and new folder-based databases) to a SQLite database.

Use the Convert menu under the File menu to choose which kind of database you would like to convert. You will be prompted for the REALbasic database to convert. If the conversion is successful, the newly converted SQLite database will be opened in SQLiteManager. All of the column constraints and field types should be preserved in the SQLite database that is created during the conversion.

SQLiteManager can also convert SQLite 2 databases to SQLite 3 format. Choose "SQLite 2 to SQLite 3" from the Convert menu and you will be prompted for a SQLite 2 database to convert. In addition to tables, SQLiteManager will also convert the views, indexes, and triggers in the SQLite 2 database.

REALSQL Server

REALSQL Server is a powerful and fully featured SQL server build upon SQLite 3 databases technology. SQLiteManager can easely connects to a remote server and uses its databases as normal local database files.

For a better experience, an admin account is recommended when you try to manage a remote database.

Creating Reports

You need two things to generate reports in SQLiteManager: an SQL query and a report template. To create a report template, first open an existing database or create a new one. Choose Create Report from the Action button (at the bottom of the object browser). A dialog will appear into which you need to type a template name and the template itself. You may call the template anything you'd like. In fact, you can have more than one template with the same name, though that isn't recommended. Next, type the text of the template itself and then click the OK button. We'll go over the specifics of creating a report template in a minute, but for now, you can use this template, which creates an HTML report. Copy and paste the following into the text area of the new report dialog:

```
<html><body>
<table>
<tr>
[LOOPCOLUMNS]<td><b>[FIELD.LABEL]</b></td>[/LOOPCOLUMNS]
</tr>

[LOOPROWS]
<tr>
[LOOPCOLUMNS]
<td>[FIELD.VALUE]</td>
[/LOOPCOLUMNS]
</tr>
[/LOOPROWS]

</table>
</body>
</html>
```

You should now see your report template under reports in the objects browser pane of the Design tab. Assuming you have a table for which you'd like to generate a report, choose "Generate Report..." from the File menu. A dialog will appear that prompts you for the name of a report template and an SQL query to use to create the report.

All of your report templates should be available in a popup. Choose the HTML report template we just created. Next you need to enter some SQL to use as a query for the report that you will generate. If you had saved some SQL, you could choose it from a popup as well, but we will assume you don't have any saved SQL. Type in a simple query, such as the following, where "<table-name>" is replaced with the name of the table for which you want to create a report:

```
SELECT * FROM <table-name>
```

Click the OK button and you will then be prompted for a file name and location to save

the report. Type something like “report.html” and click OK. An HTML report containing all the data in your table should be generated.

Note that your query SQL can be as complicated as you would like it to be and can include multiple tables and named columns with constraints.

Now let’s consider the specifics of writing a report template. Report templates are text embedded with directives to the report generator for how to insert query data into the report. As an example, let’s consider the above HTML template. The template begins with the following text:

<html><body>
<table>
<tr>

This looks like the beginning of an HTML document. But the next line of the template has some funny stuff in it that doesn’t look like HTML. [LOOPCOLUMNS][/LOOPCOLUMNS] is a template directive that tells the report generator to loop over all the columns of a row of query data. It is assumed that the query begins with the first row of the query data. Between the beginning and ending LOOPCOLUMNS tags is the following text:

<td>[FIELD.LABEL]</td>

This text will be generated in our report for every column of the row, almost like there was a for...next loop that looped over all the columns and spit out that text each time through the loop. Embedded within the text is what looks like another template directive. [FIELD.LABEL] is a directive that tells the report generator to insert the label of the current field or column. Putting everything together, what we have is a loop over all of the columns of the first row of data, and for each column, we will spit out the label (or header) for that column surrounded by a bit of HTML that will put the label in a table cell and bold it.

The next section of the template looks familiar, in that we have a [LOOPCOLUMNS] directive, but we also have something new: [LOOPROWS]. [LOOPROWS][/LOOPROWS] is a template directive that tells the report generator to loop over all of the rows of the query. Contained within the LOOPROWS directive is another

LOOPCOLUMNS directive. In other words, what we have is a loop within a loop. We are going to loop over all the rows of the query, and for each row we are going to loop over every column. Finally, embedded inside the LOOPCOLUMNS directive is a [FIELD.VALUE] directive. The FIELD.VALUE directive is much like the FIELD.LABEL directive except that it will be replaced by the value of the current column instead of the label.

There are a couple more things you can do with reports that we haven’t seen in the above template. Let’s consider another template.

This template produces a CSV report:

[LOOPCOLUMNS separator=',' lastSeparator='\n']"[FIELD.LABEL]"[/LOOPCOLUMNS]

[LOOPROWS]
[LOOPCOLUMNS separator=',' lastSeparator='\n']"[FIELD.VALUE]"[/LOOPCOLUMNS]
[/LOOPROWS]

As you can see, the LOOPCOLUMNS directive can take two arguments: 'separator' and 'lastSeparator'. The 'separator' argument specifies what text to insert between each column generated by the LOOPCOLUMNS. The 'lastSeparator' argument specifies what text to end the LOOPCOLUMNS with. You can use any arbitrary text as a separator or a lastSeparator. You can also use the following special characters: '\t' for tab, '\n' for a UNIX newline, '\r' for a Mac newline, and '\r\n' for a DOS newline. Note that currently if you use one of these special characters, then that must be the only value for the separator. In other words, you can't construct a separator or lastSeparator that contains an arbitary mix of characters and special characters.

In adition to the [FIELD.VALUE] directive, you can also refer to column values directly by name by enclosing a column's name in brackets. For example, if you had a table of information about people and you wanted to generate an HTML report containing the names and ages of everybody in the table, you could use the following template:

```
<html><body>
<table>
<tr>
<td><b>Name</b></td><td><b>Age</b></td>
</tr>

[LOOPROWS]
<tr>
<td>[name]</td><td>[age]</td>
</tr>
[/LOOPROWS]

</table>
</body>
</html>
```

Instead of looping over all the columns for every row of the query, we just refer directly to the "name" and "age" columns. The report generator will replace [name] and [age] with the values for the "name" and "age" columns for each row.

The only template directive left to talk about is the [INDEX] directive. When the report generator sees the [INDEX] directive it replaces it with either the row number or column number of the current loop. Whether it is a row number or a column number is determined by context. If [INDEX] occurs inside of a [LOOPROWS], then it will be replaced by a row number. If, on the other hand, [INDEX] is inside a [LOOOPCOLUMNS], then it will be replaced by a column number.

For example, if we wanted to number all of the rows of the above name-age template, then we could do it as follows:

```
<html><body>
<table>
<tr>
<td></td><td><b>Name</b></td><td><b>Age</b></td>
</tr>

[LOOPROWS]
<tr>
<td>[INDEX]</td><td>[name]</td><td>[age]</td>
</tr>
[/LOOPROWS]

</table>
</body>
</html>
```

The report generation system is unable to maintain newlines, so to provide exact control of where newlines are inserted, there is a [NEWLINE] directive that you can add to your report templates. For example, the above HTML template would not preserve newlines. If you wanted to preserve newlines, you could rewrite the template as follows:

```
<html>[NEWLINE]
<body>[NEWLINE]
<table>[NEWLINE]
<tr>[NEWLINE]
<td></td><td><b>Name</b></td><td><b>Age</b></td>[NEWLINE]
</tr>[NEWLINE]

[LOOPROWS]
<tr>[NEWLINE]
<td>[INDEX]</td><td>[name]</td><td>[age]</td>[NEWLINE]
</tr>[NEWLINE]
[/LOOPROWS]

</table>[NEWLINE]
</body>[NEWLINE]
```

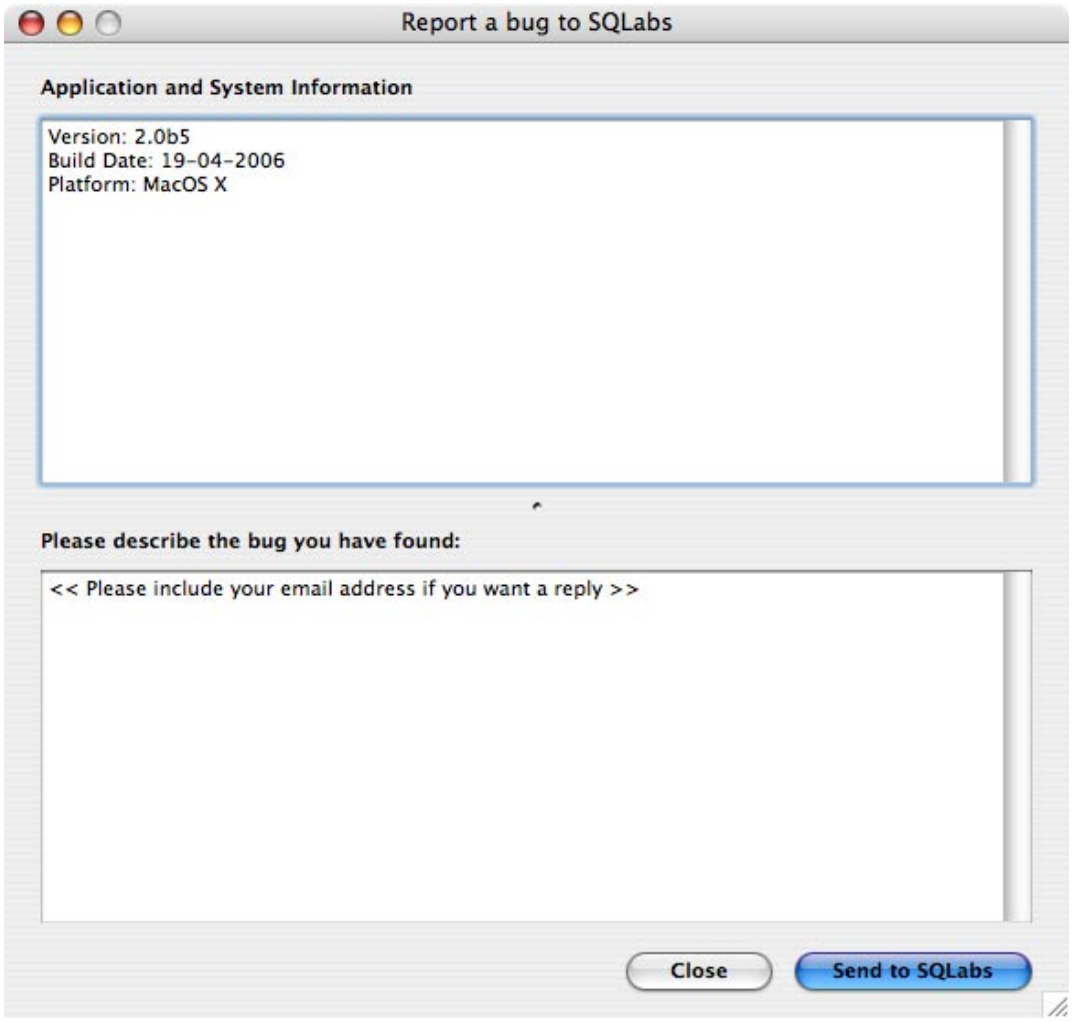
</html>[NEWLINE]

The kind of newline that is inserted into the report will depend on the platform on which the report is generated. For example, if the report is generated on Windows, then a Windows-style newline will be used (carriage return followed by linefeed).

Bug/Crash Reporter

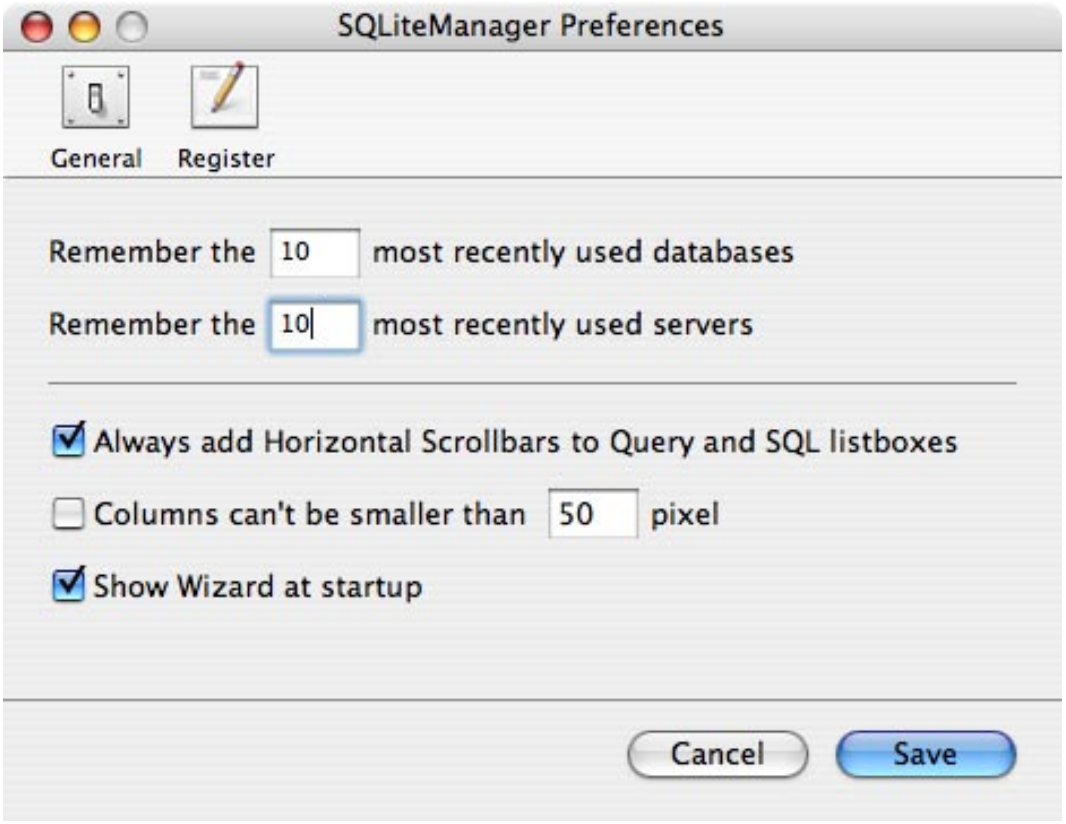
SQLiteManager has a built-in bug reporter, just select “Report a bug to SQLabs” from the Help menu in order to have the Bug Reporter dialog.

Note that this dialog will automatically appears each time a crash bug will occurs in the application. The top field will be automatically filled with important information necessary to SQLabs to better track down the bug. No other sensible information like your name or you serial number will be sent with the bug/crash report.



Preferences

Use SQLiteManager's preferences dialog, shown above, to adjust the SQLiteManager's behavior.



You can set the maximum number of databases and server to remember or you can modify the appearance of the Query and SQL listboxes.

Please note that the option "Show Wizard at startup" is available only in the MacOS X version.

Preferences

Select the “Register” icon, shown above, to register your copy of SQLiteManager. Until you register, SQLiteManager runs with a lot of limitations. To register, enter your serial number in the “Serial Number” field. The Company and Name fields are optional.

Press the button “Check SN” if you want to check if your SN is valid and if it can unlocks all the SQLiteManager’s features.

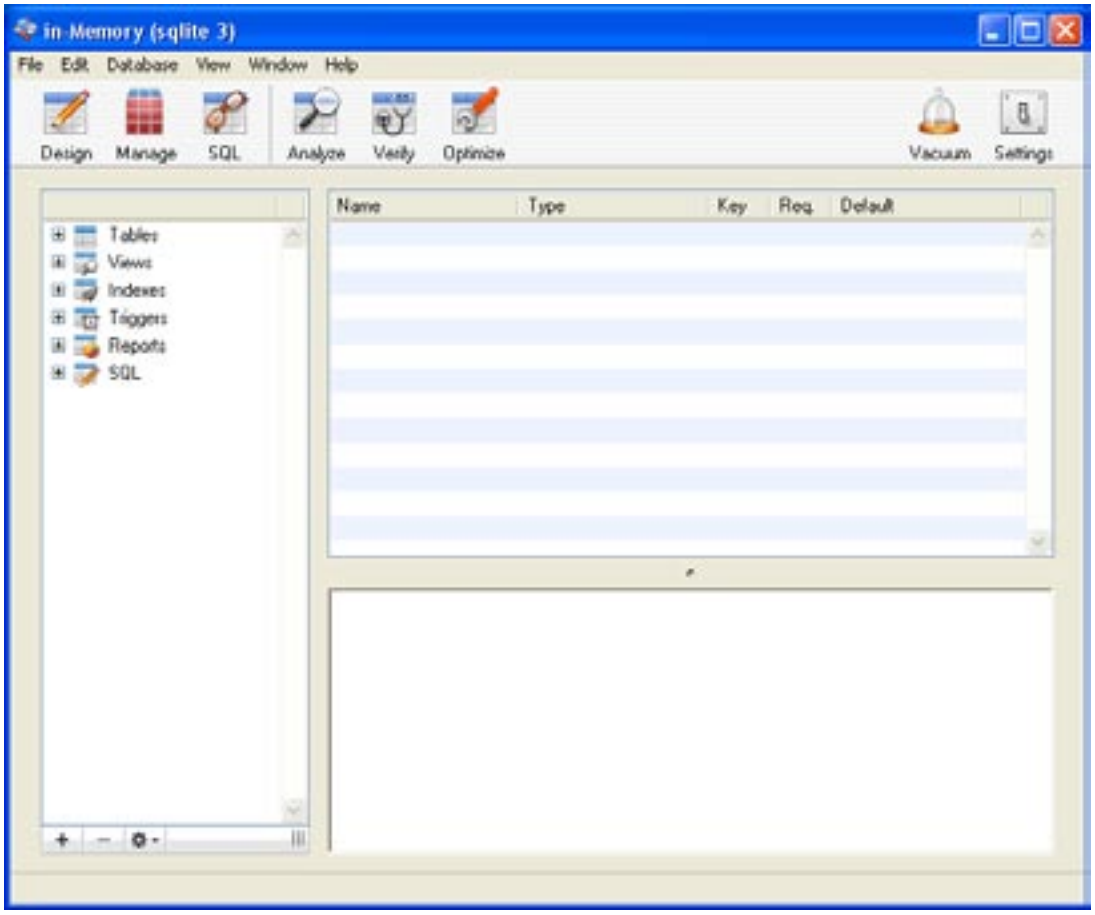
Appendix A

InMemory Databases

SQLiteManager supports creation of InMemory database (from the startup Wizard or from the New submenu in the File menu).

InMemory databases can be very useful if you need to test your ideas, your sql or just to have the speed of RAM based databases.

Once you have finished working with InMemory databases you can choose to discard it or you can choose to dump the entire database (with its structure and data) to file. To do this just select “Dump database” from the File menu.



Appendix B

Contact Information

Marco Bambini
marco@sqlabs.net

Web: http://www.sqlabs.net
Email: info@sqlabs.net

Copyright

Legal Stuff