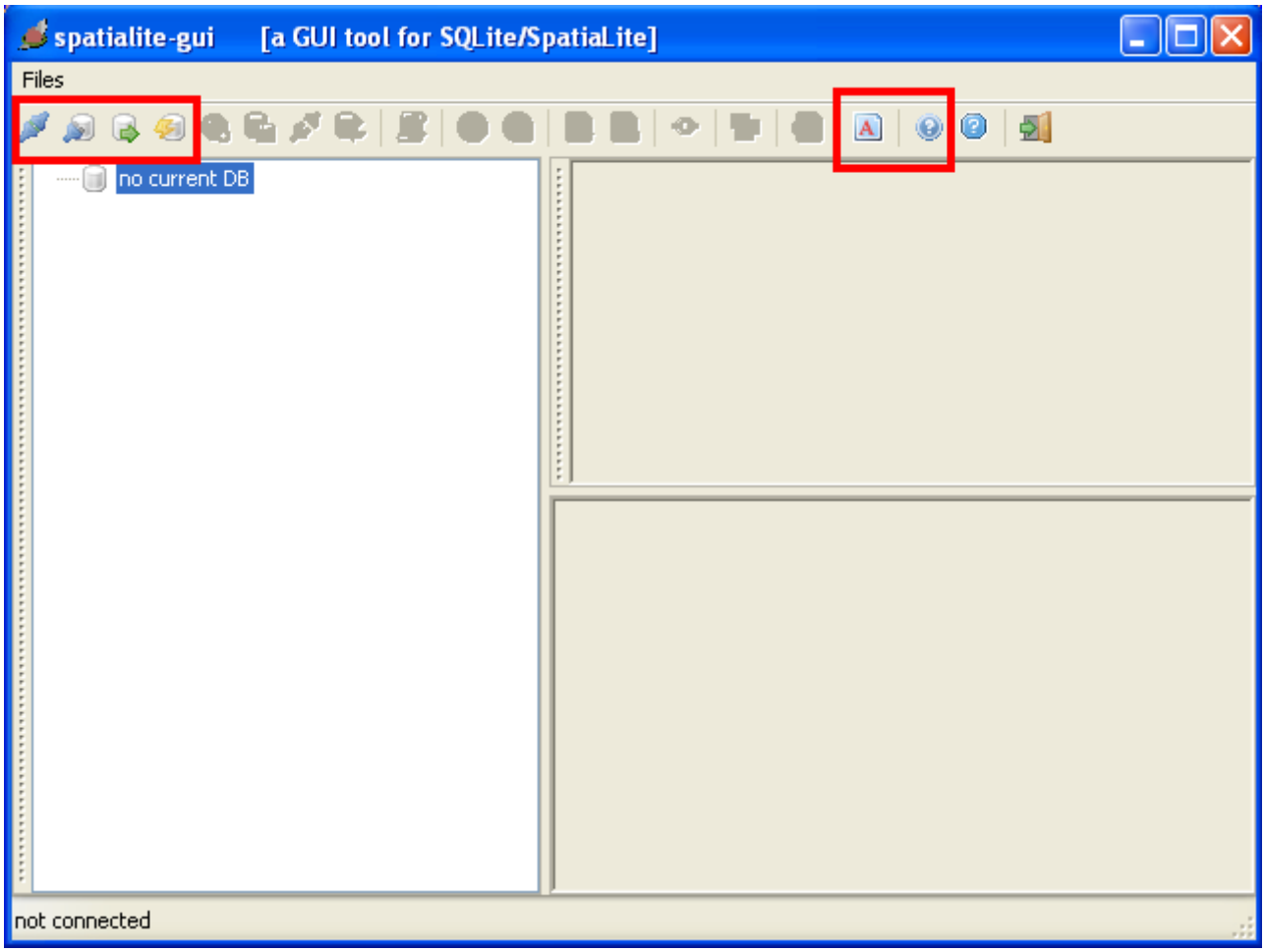
Spatialite-gui

a GUI tool to manage SQLite and SpatiaLite databases

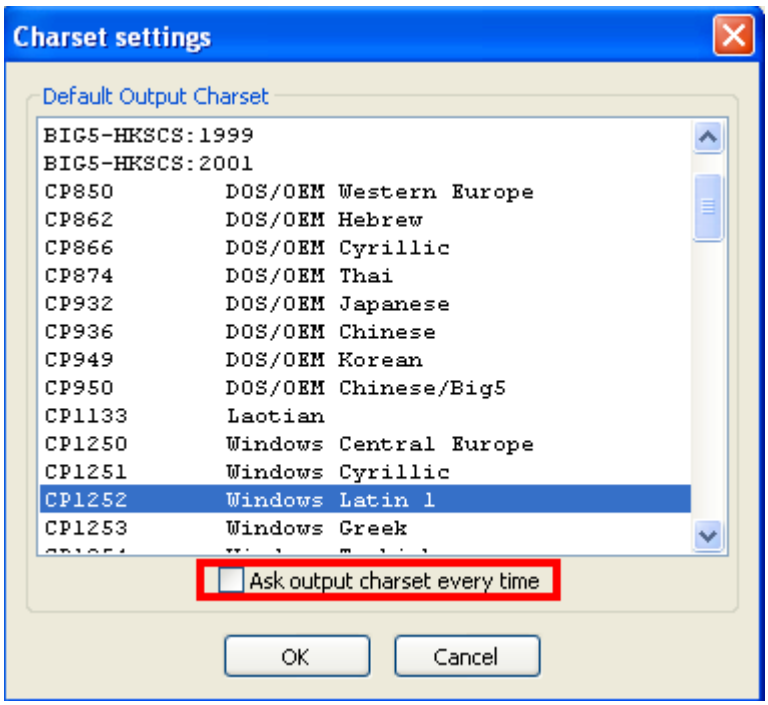
Just few very short notes showing

How to get started as quick as possible



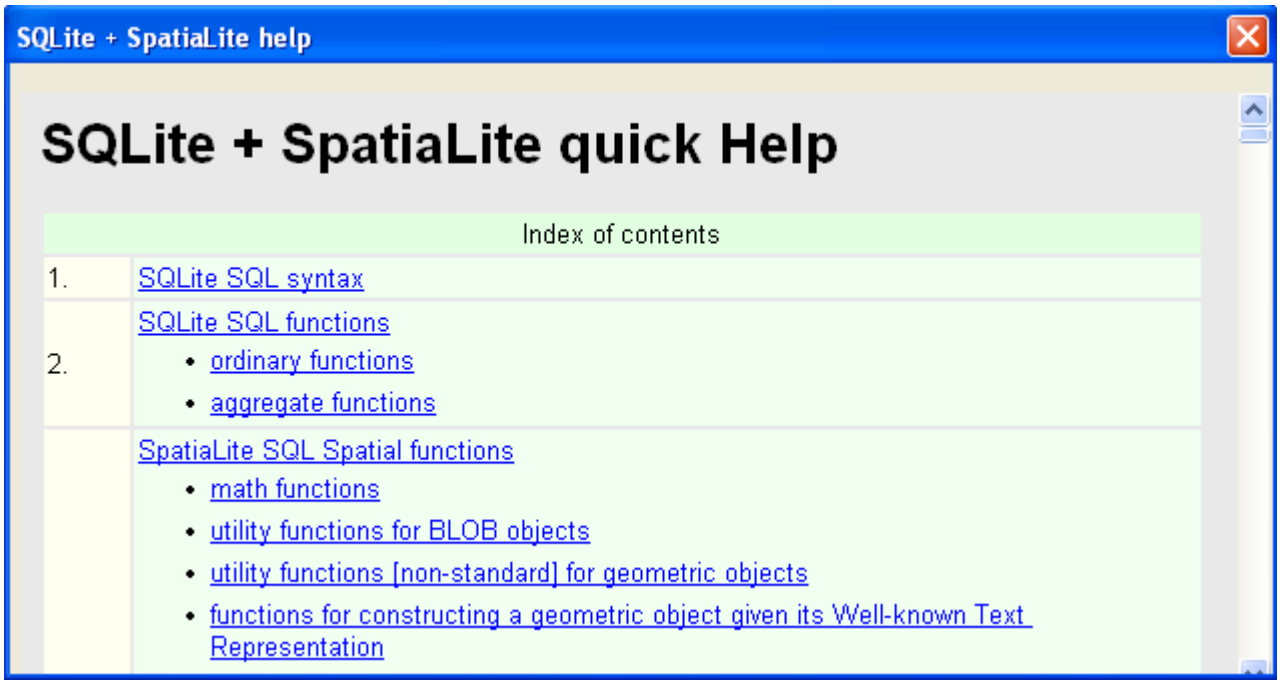
You've just launched **spatialite-gui**; so you are now facing a window like this one … as you can easily understand, currently there is no database connected.

We obviously have now to establish a connection to some database, but before beginning any actual work, let us examine the two buttons evidenced at the tool-bar rightmost side.



Default output charset:

spatialite-gui automatically detects the *locale charset* setting currently in use on your PC; but may well be you wish to use a different one, or you prefer choosing the output charset to be used each time one is needed. This small pane enables you to manage this aspect as you need.



On-line help:

You can easily check the SQL syntax expected by SQLite and SpatiaLite using this help pane.

Establishing a connection to some database:

SQLite adopts a peculiar architecture: a whole database is contained in an ordinary file. So you can choose one of the following actions:

- connect to an already existing ***database-file***
- create a new (initially empty) ***database-file***

NOTE: the latest ***spatialite-gui*** version automatically initializes any newly created database.
Consequently now there is no need to manually execute the ***init_spatialite.sql*** script, because this task has already been performed during database creation.
As you can notice, the ***geometry_columns*** and ***spatial_ref_sys tables*** are already defined and populated as appropriate immediately after database creation.

And ***SQLite*** supports as well the capability to store databases directly in memory, with no need to use any file-system file. So the complete range of actions you can perform in order to establish a database connection is as follows:

- connecting to an already existing ***database-file***
- creating a new (initially empty) ***database-file***
- loading an already existing ***database-file*** as an ***in-memory-database***
- creating a new (initially empty) ***in-memory-database***

An ***in-memory-database*** is exactly identical to a ***database-file***; there is no difference at all, except for the specific features allowed by each one of the storage media used:

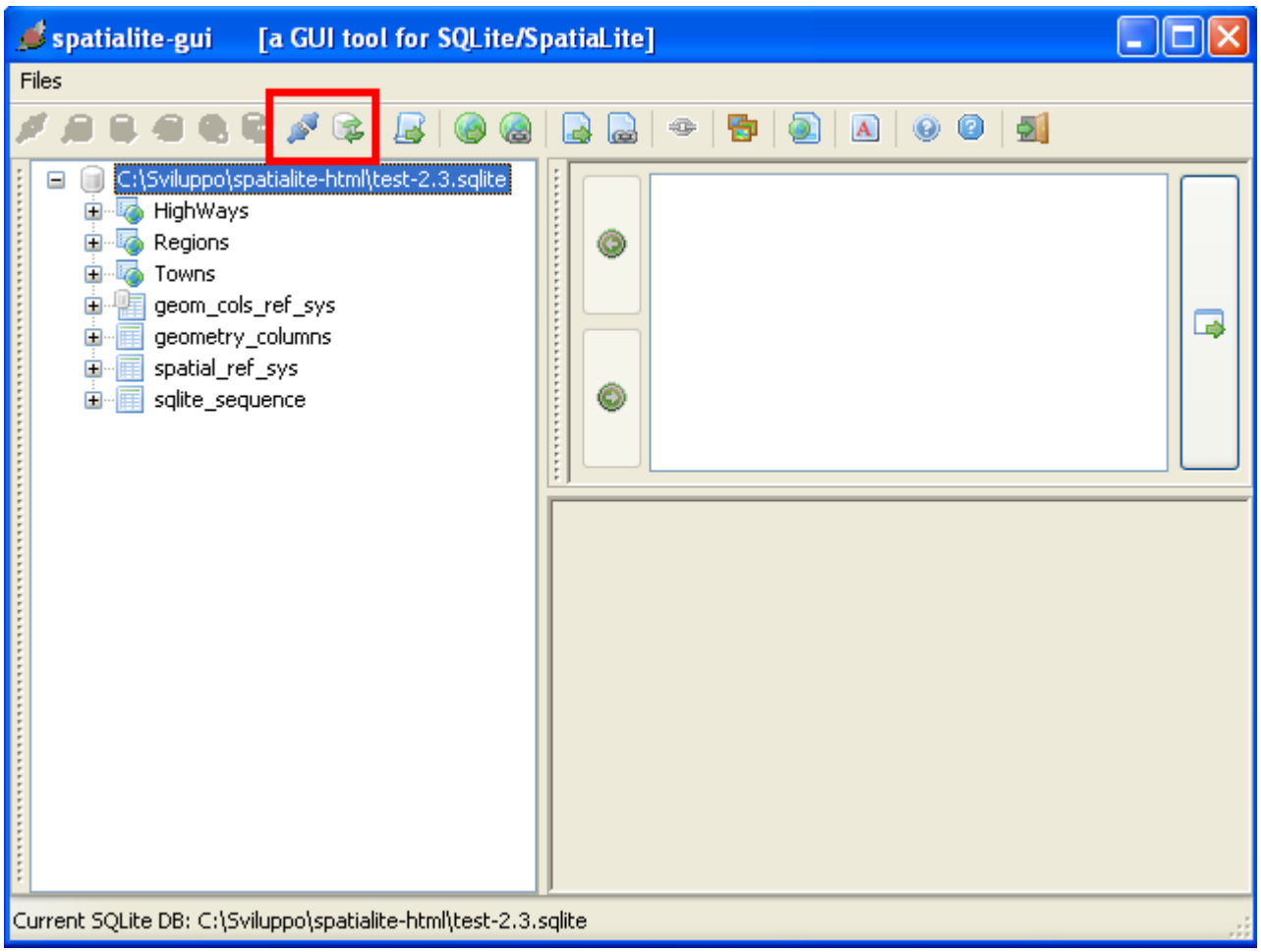
- **disk storage** is slow, but persistent and usually offers an huge space to be allocated.
- **memory storage** is fast, but volatile and usually offers a much more limited space.

This practically means that very easily you can get ***supersonic performances*** using ***in-memory-databases***, but this may be a risky operation because:

- if some trouble arises (*power failure, system crash* …) any update you have performed on the database will be irreparably lost.
- if the memory space required to store the database is too big, you can then get the paradox effect to slow down your system in a severe way.

Anyway, using an ***in-memory-database*** really is a very smart idea if:

- your database requires a reasonable amount of space: e.g. supposing you are using some PC mounting 2GB RAM, you can safely load a 512GB database. But trying to load an 1GB database on 512GB RAM is a very bad idea for sure.
- you are using the database essentially in a ***read-only*** mode, i.e. you are plannig to perform lots and lots of SELECTs, and very few INSERTs, UPDATEs or DELETEs
- you are initially feeding a new database importing several shapefiles. Creating and feeding lots of tables, and creating many R*Tree spatial indices and ordinary indices may take a long time; working ***in-memory*** will help a lot. And after all, if something goes the wrong way you can restart anything from scratch with few pain.



Supposing you have opted to connect some ***database-file*** [the conventional way], you are now enabled to perform the following actions:

Close the database connection:

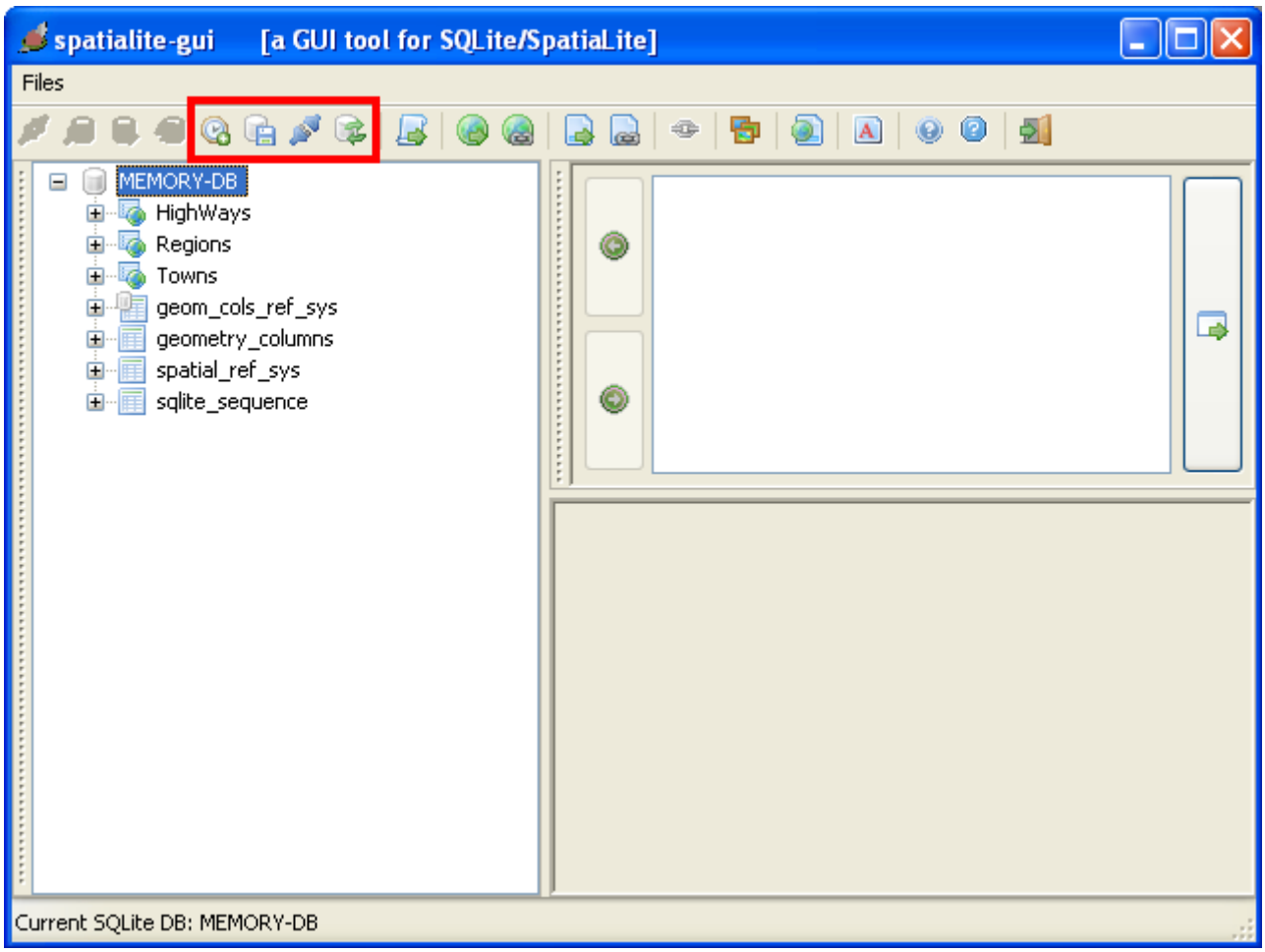
spatialite-gui is no longer connected to any database.
And you can now connect to a different database.

Perform a VACUUM:

This one is a maintenance operation. The complete database will undergone a full rebuild, and any unused space will be actually freed:

- VACUUM*ing* an huge database may require a long time
- a properly VACUUM*ed* database surely performs better

It's your choice why and when VACUUM*ize* your databases; anyway performing a VACUUM is absolutely suggested as a ***good practice*** every time you've performed an huge number of INSERTs and/or DELETEs.



Supposing you have opted to connect some ***in-memory-database*** [the unconventional way], you are now enabled to perform the following actions:

Close the database connection:

Exactly the same as above. But this time your database was using a ***volatile*** storage media, so any data is now irremediably lost.
This one may be a good new is you where performing some stupid test, but may be a real catastrophe if you where performing any serious work.

Perform a VACUUM:

Exactly the same as above, but this time anything runs faster.
Please, carefully consider that VACUUM*ing* an ***in-memory-database*** more or less requires ***twice*** the memory amount currently required.

Save the *in-memory-database* as a *database-file*:

Any ***in-memory-database*** is volatile, but any ***database-file*** is persistent.
So, exporting the complete database precariously stored ***in-memory*** as a permanent ***database-file*** hosted on the local file system may be a very good idea, if you are anyway interested in preserving your data for the eternity [*or so on* …].

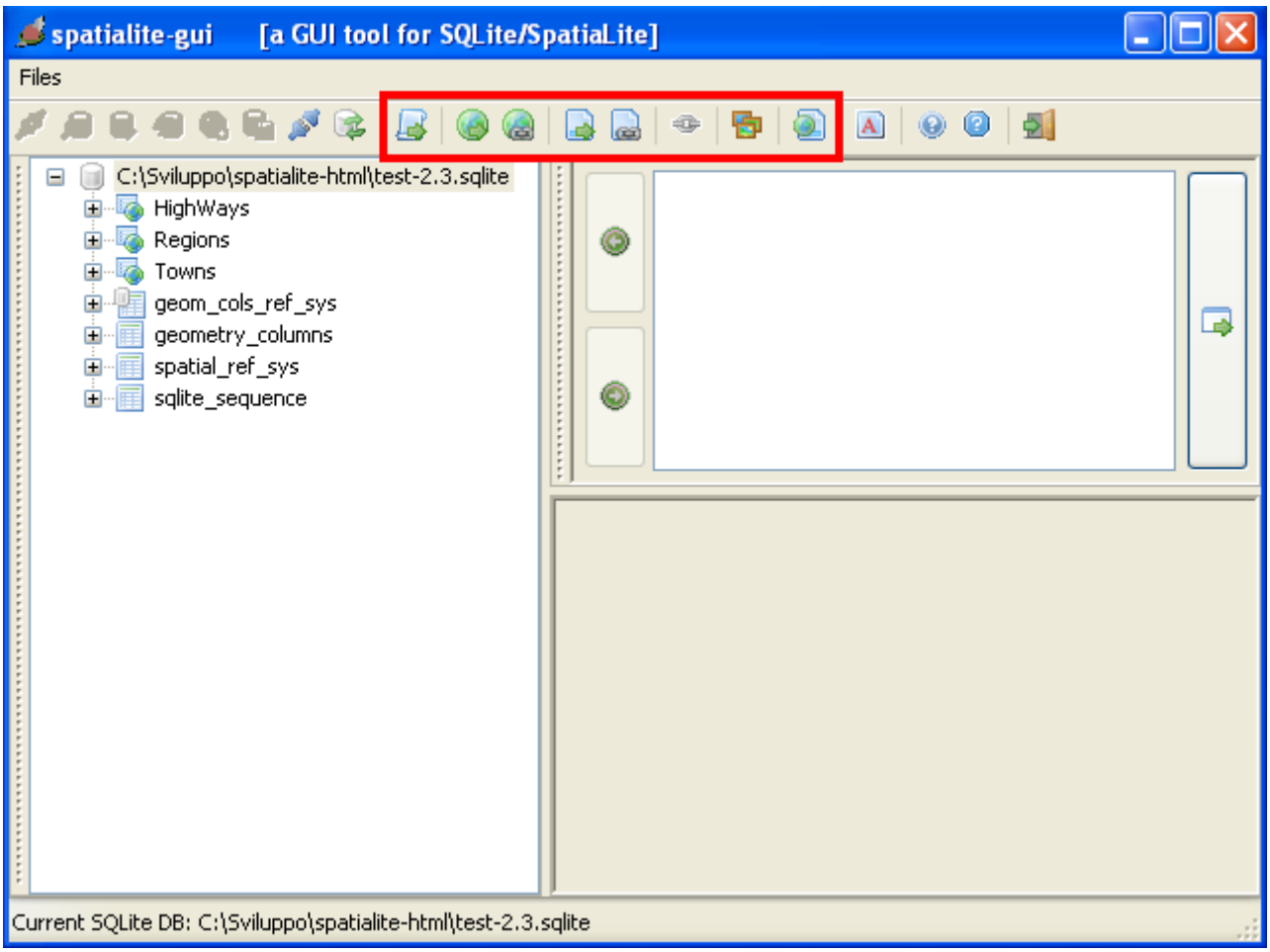
Exporting a database requires a several seconds / few minutes; obviously this depends on available hardware and/or database size; anyway this option makes the use of memory storage a sane solution, because you are allowed to:

- take any speed benefit deriving from using memory storage.
- and then save permanently any update you've performed.

Configure the *AutoSave* feature:

spatialite-gui supports an even smarter feature, i.e. the one of ***AutoSaving*** the ***in-memory-database*** to a corresponding ***database-file*** from time to time, on a regular and periodic base:

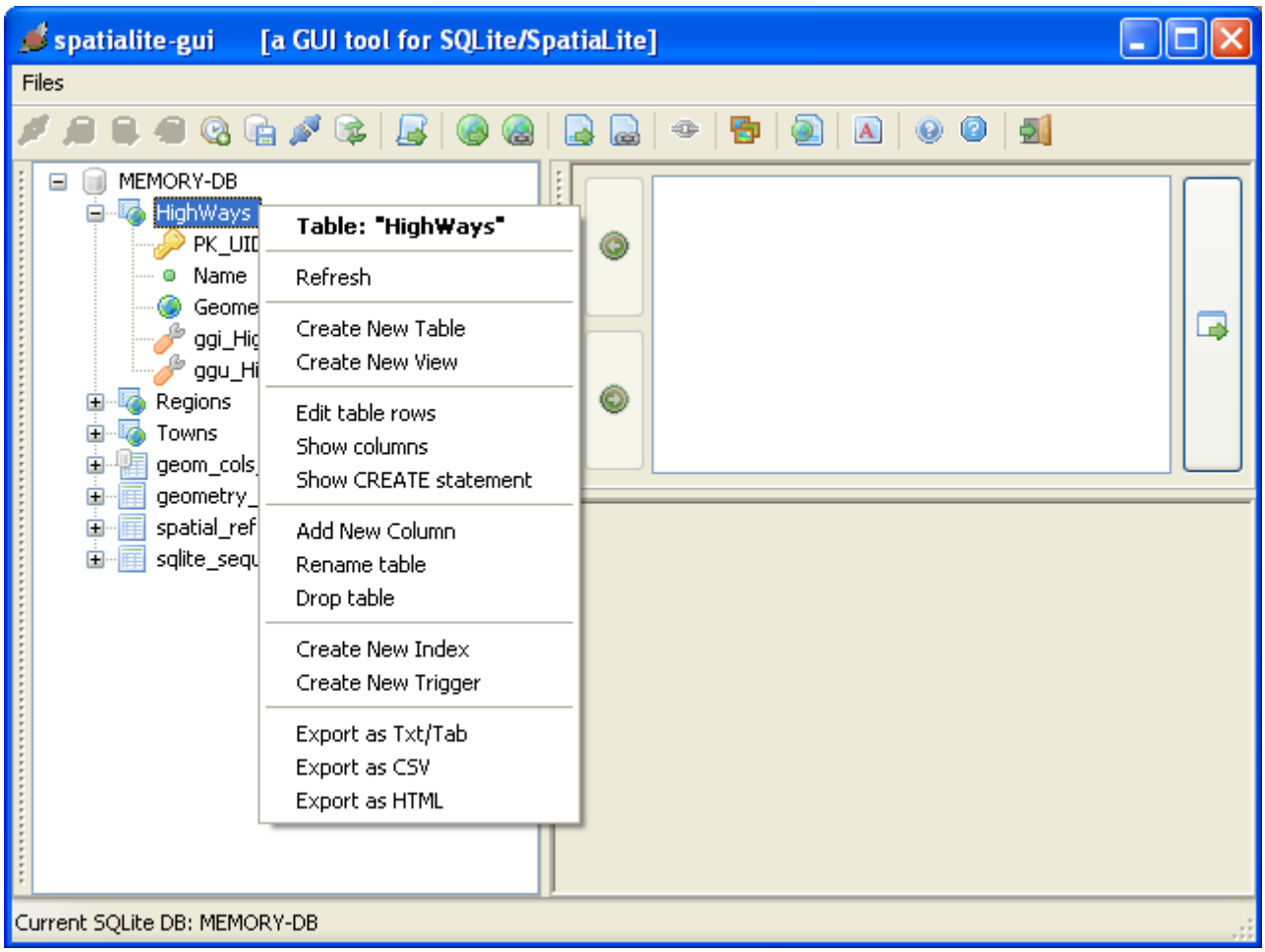
- an external ***database-file path*** is required
  - if the ***in-memory-database*** was loaded from a ***database-file***, then the original ***database-file*** will be periodically overwritten.
  - if the ***in-memory-database*** was created from scratch, in order to actually start the ***AutoSave*** feature you are required to perform a ***manual Save*** a first time; then the same path will be periodically overwritten.
- you can select the time interval, choosing the one most well suited to your needs.
- The AutoSave feature is smart enough to skip unneeded exports [i.e. when the ***in-memory-database*** has no pending changes to be saved]
- To prevent any obnoxious consequence deriving from system crashes, power failures and so on, the following security schema is implemented by ***AutoSave***:
  - the already existing ***database-file*** is renamed as ***database-file.bak***
  - then a new copy of database-file is generated by export.
  - and finally ***database-file.bak*** is deleted
  - so, if something goes the wrong way, ***database-file.bak*** still contains a (*may be obsolete*) valid copy of your database.



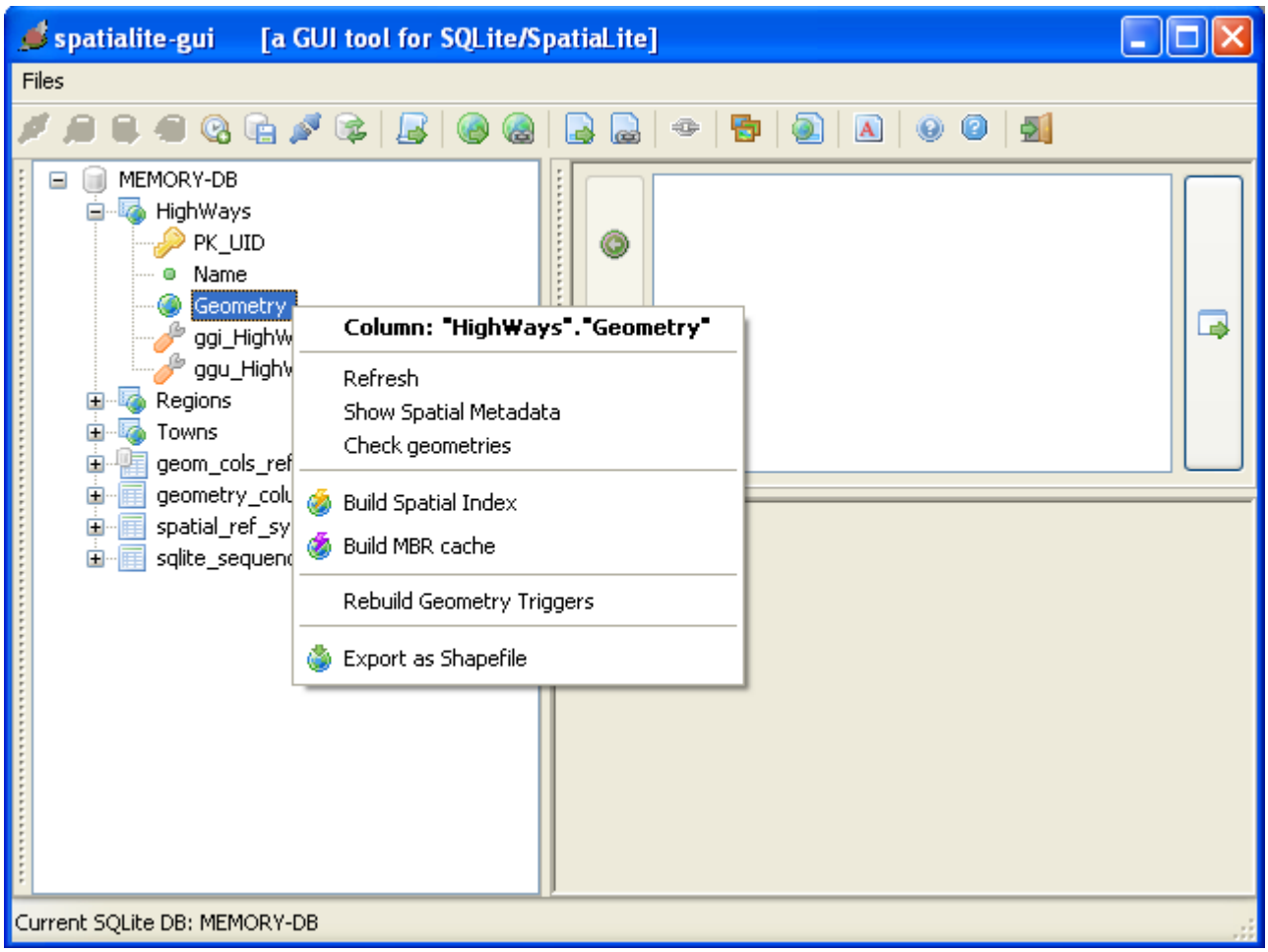
All right, now you know all we need to know about database connections.
It's time to start using ***spatialite-gui*** for some useful task.

The tool-bar enables you to start the following activities:

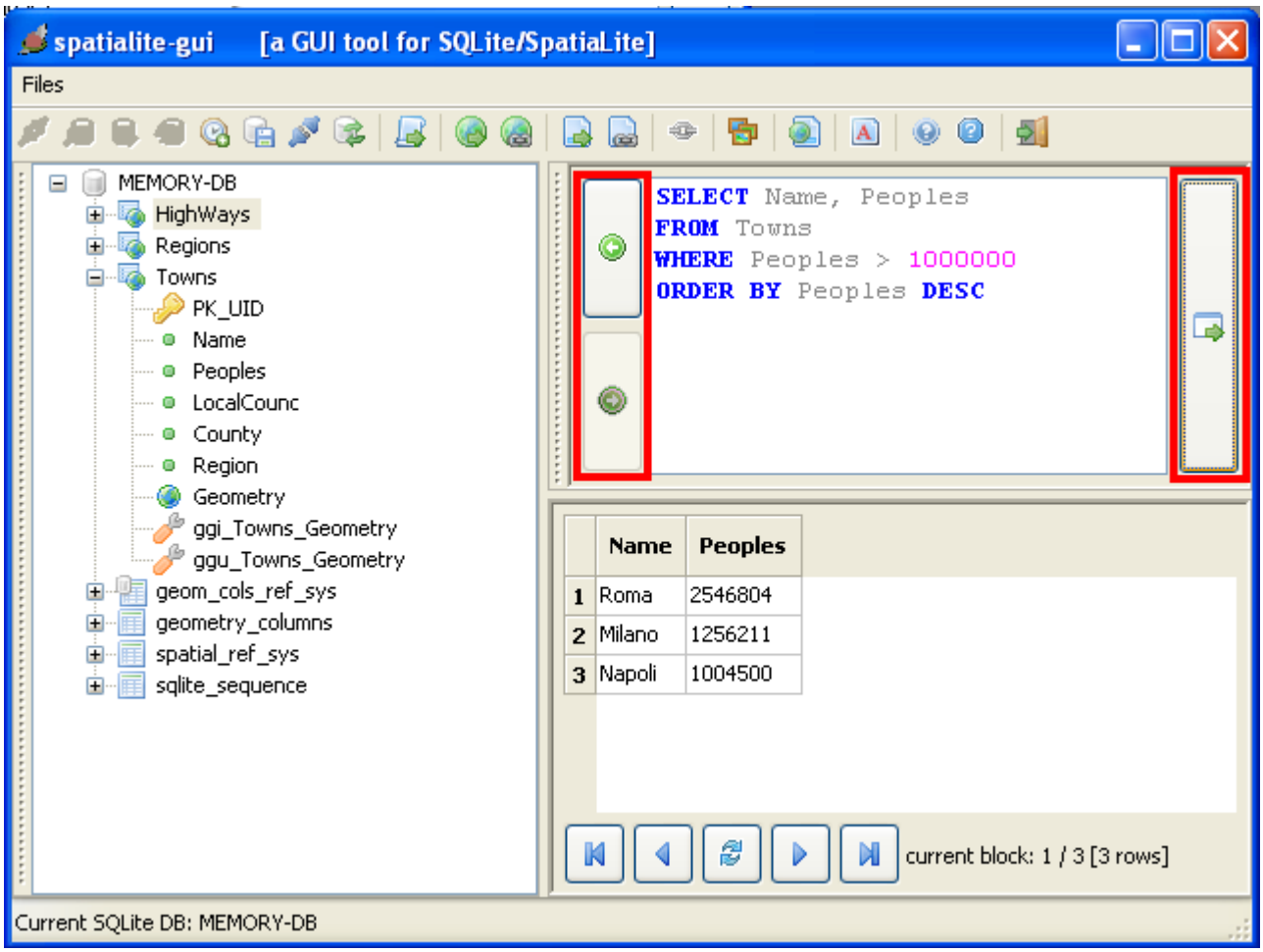
- execute some SQL script
- load some Shapefile into the database
- access some external Shapefile using the VirtualShape module
- load some TXT/CSV file into the database
- access some external TXT/CSV file using the VirtualText module
- build a Network to be used for Routing
- import one (or more) EXIF GPS pictures into the database
- search the EPSG reference system dataset by name



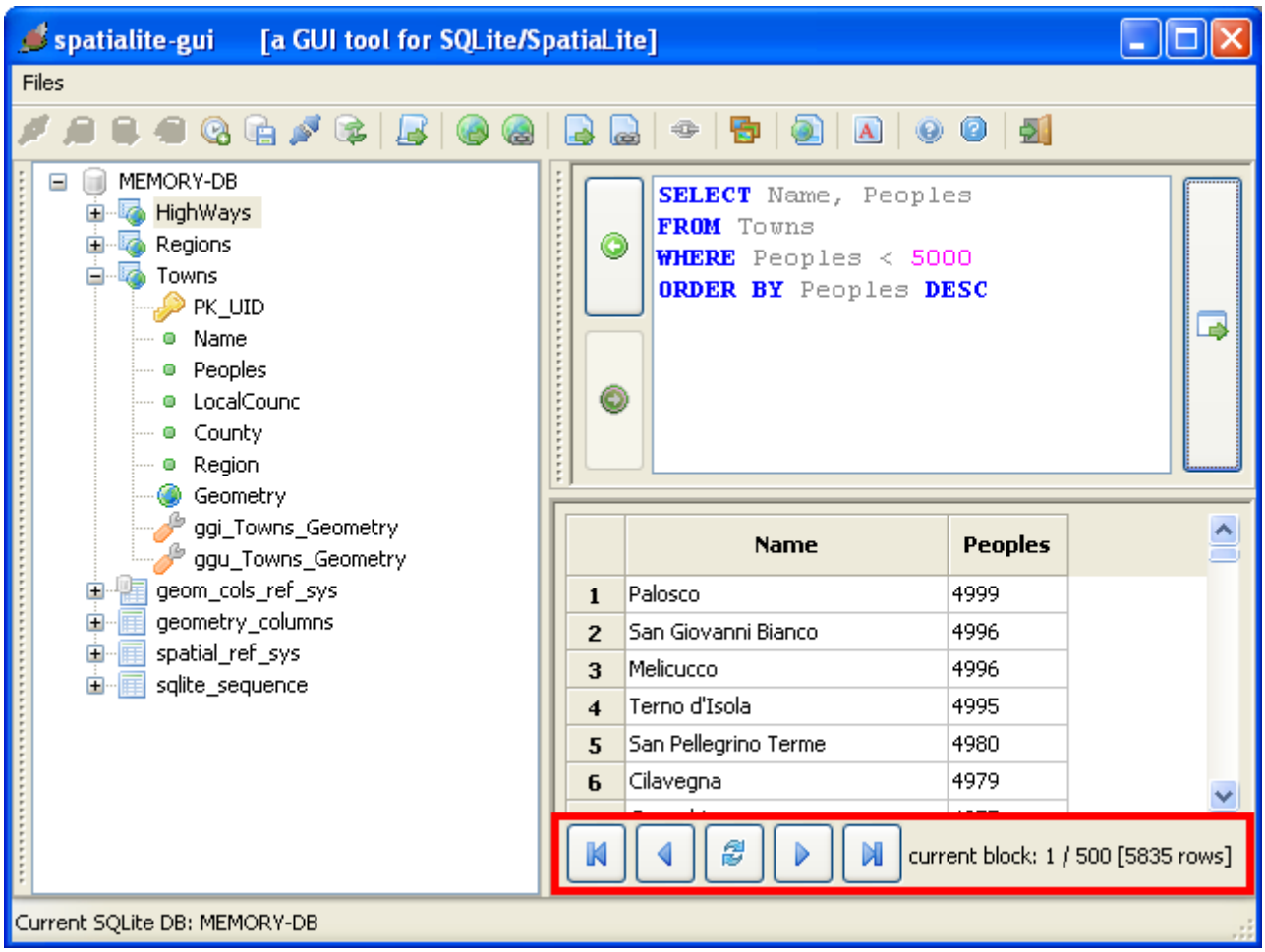
You can perform more specific actions involving a single table simply left-clicking on the table name, and then selecting the required action from the context menu …



… and you can perform specific actions involving a single Geometry column following the same way as above …

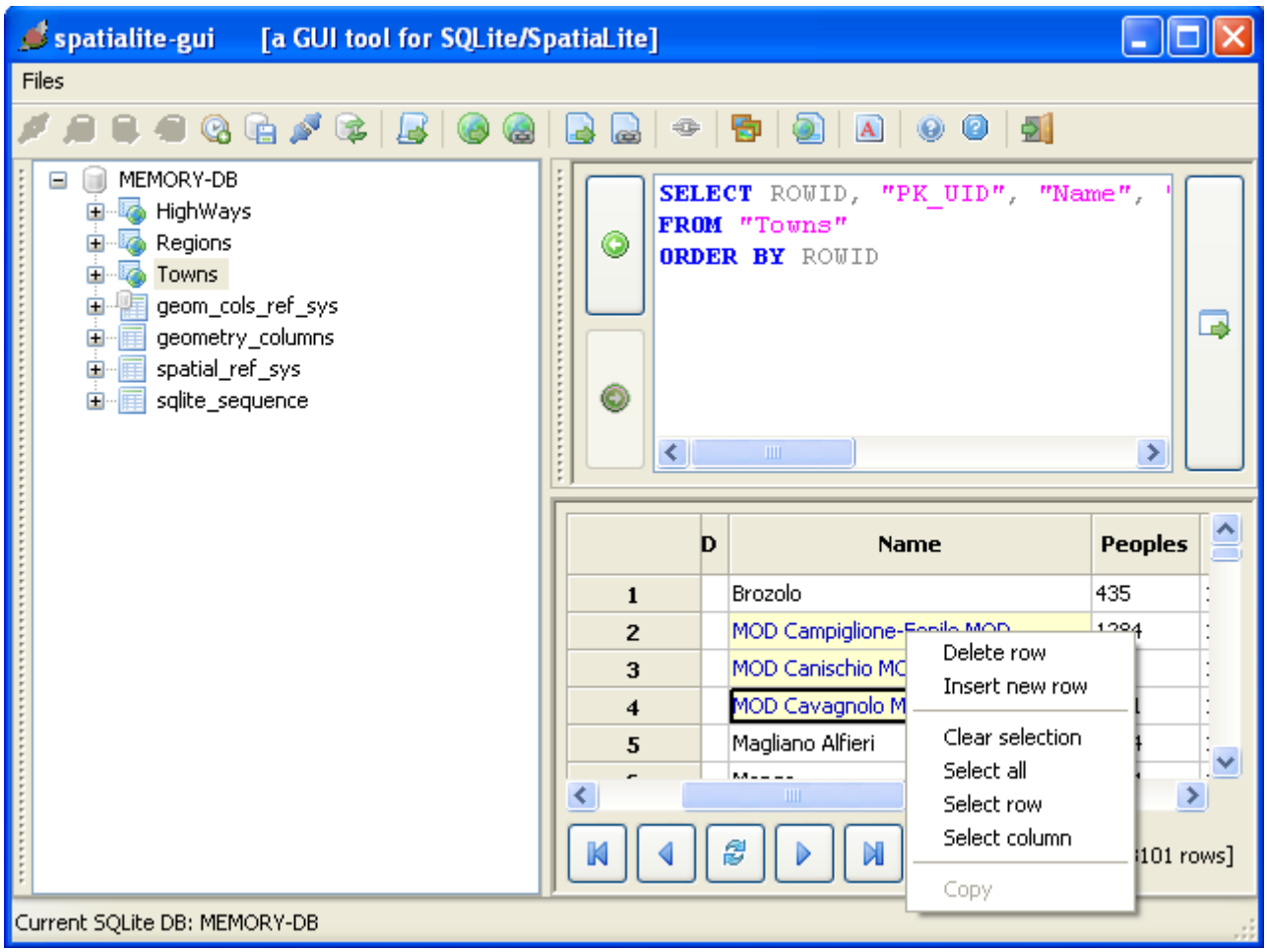


... you can compose any SQL statement at your will and then execute it.
If the SQL statement you've just executed returned some ***result set***, then this latter will be shown as a scrollable grid.
And ***Spatialite-gui*** manages an history containing any SQL statement performed since now; you can navigate the history back and forward, and eventually re-execute some statement again.

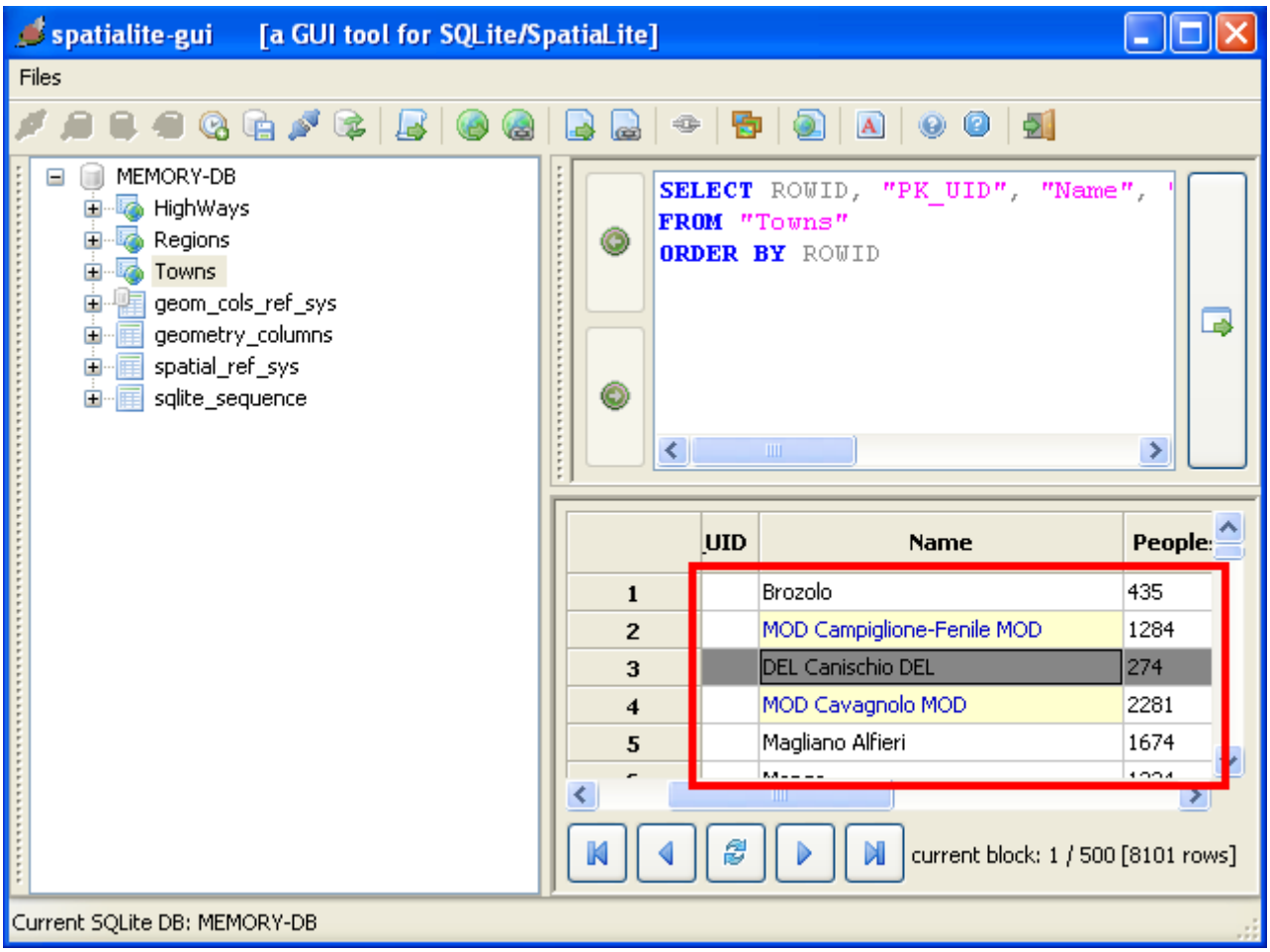


If the ***result set*** returned by some SQL statement is a very long one [i.e. if it contains a large number of rows], then ***spatialite-gui*** avoids to show all the lines contained in the result set, because such an action may potentially require an huge memory amount, and may take a very long time.
So, when an huge result set is encountered, ***spatialite-gui*** will show only 500 rows at each time [i.e. a single block of rows is shown at each time]. You then can:

- goto the first block in the result set
- goto the last block in the result set
- goto the next block, i.e. the one immediately following the current one
- goto the previous block, i.e. the one immediately preceding the current one
- refresh the result set, i.e. performing the query again another time.



If the result set currently shown into the grid was obtained by activating the **Edit table rows** menu item, then you are actually enabled to edit cell values, more or less in the same way as if you where using some spreadsheet software.



Any modified cell value will then be evidenced; and deleted rows will be evidenced as well.

That's all folks ... enjoy and have fun