

php-sqlite3geo

a PHP extension module supporting
SQLite with Spatial Data

State of the art: PHP support for SQLite is a little bit confusing.

- The old **SQLite-1.0.3** package was first in supporting **sqlite**, but support is limited to **sqlite v2**, so now it is completely obsolete; development ceased during 2004.
- **PHP5** supports natively **sqlite3** [*with no need for using extra extensions*] via the standard **PDO_SQLITE** driver, which comes bundled with any installation of PHP5. Regrettably the PDO_SQLITE has the following pitfalls:
 - There is no way at all to perform SQLite's dynamic extensions loading
 - Support for BLOB data is quite unsatisfactory
 - So it's not at all well suited to support SQLite's Spatial extensions.
- The **php-sqlite3-0.5** extension [http://sourceforge.net/project/showfiles.php?group_id=150569] developed by [Bruno Fleisch](#) on its own seems to offer a valid and sound start point; it's in *alpha state* [*quite buggy*], is quite simple but reasonably complete. So the **php-sqlite3geo** is merely a slightly revised derivative of Bruno Fleisch's original work.

Building and installing php-sqlite3geo: simply follow the attached README instructions.

Basically you have to do:

- ◆ **phpize**
- ◆ **./configure --with-sqlite3geo**
- ◆ **make**
- ◆ **make install**

Now you'll have the **php-sqlite3geo** PHP extension installed under **/usr/lib/php/modules**

It's not at all a bad idea to **strip** the extension module, so:

- ◆ **cd /usr/lib/php/modules**
- ◆ **ls -l php-sqlite3geo**
- ◆ **strip php-sqlite3geo**
- ◆ **ls -l php-sqlite3geo**

Final step is to load the **php-sqlite3geo** extension in a permanent way.

In order to accomplish this goal, you'll edit the **/etc/php.ini** file:

```
;;;;;;;;;;;;;
; Dynamic Extensions ;
;;;;;;;;;;;;;
;
; If you wish to have an extension loaded automatically, use the following
; syntax:
;
; extension=modulename.extension
;
; For example:
;
; extension=mysql.so
;
; Note that it should be the name of the module only; no directory information
; needs to go here. Specify the location of the extension with the
; extension_dir directive above.
extension=sqlite3geo.so
```

Now you simply need to restart the Apache web server, in order to actually load **php-sqlite3geo** :

◆ **service httpd restart**

May be you are planning to test SQLite Spatial as well; after all, this one is the main rationale to use **php-sqlite3geo** ... so, it's better to check if your system already has the **Spatialite.so** and **VirtualShape.so** extensions installed: they usually must be located under **/usr/lib**

If not, install them before calling **sqlite3_load_spatial()**, otherwise you'll get a failure at run time.

Documentation: the **examples** directory contains some useful code examples [original Bruno Fleisch's distro].

The **geo_examples** directory contains other code examples more specifically focused on using SQLite spatial, BLOBs and so on:

- **database metadata** handling: see *sqliteTables.php* and *sqliteColumns.php*
- full parametric [*blind*] **table's rows browsing**: see *sqliteBrowse.php*
- **BLOB hexadecimal dump**: see *sqliteHex.php*
- fetching **images from BLOBs**: see *sqliteImgFrame.php*
- using **GEOMETRY**: see *sqliteGeo.php* and *sqliteGeoSummary.php*
- **drawing GEOMETRY** via GD: see *sqliteGeoPreview.php*

Anyway, formal specification for new functions [the ones specific of **php-sqlite3geo** and not included into the original **php-sqlite3**], is as follows:

```
resource sqlite3_open_read_only ( string path_to_database )
```

Strictly analogous to the original **sqlite3_open()**

- the SQLite DB will be opened in READ-ONLY mode [no UPDATE, INSERT or DELETE will be subsequently allowed on this DB]
- if the the SQLite DB identified by *path to database* didn't exists [or read access is anyway denied by permission settings], no new DB will be created.
- on success will return a **valid handle** to access the SQLite DB
- on failure will return **NULL**

example:

```
$db = sqlite3_open_read_only('/home/user/db.sqlite');  
if ($db == NULL)  
    die('...');
```

```
constant sqlite3_guess_blob_content ( string blob_value )
```

Useful in order to check the content type for a column value of BLOB type; returns:

- **SQLITE3_BLOB_GIF** if *blob_value* contains a valid GIF image
- **SQLITE3_BLOB_PNG** if *blob_value* contains a valid PNG image
- **SQLITE3_BLOB_JPEG** if *blob_value* contains a valid JPEG image
- **SQLITE3_BLOB_GEOMETRY** if *blob_value* contains a valid GEOMETRY
- **SQLITE3_BLOB_PDF** if *blob_value* contains a valid PDF document
- **SQLITE3_BLOB_ZIP** if *blob_value* contains a valid Zipfile
- **SQLITE3_BLOB_HEX** in any other occurrence

example:

```
$rs = sqlite3_query ( $db, 'SELECT ....' );  
$row = sqlite3_fetch ( $rs );  
if ( sqlite3_column_type ( $rs, 0 ) == SQLITE3_BLOB )  
{  
    $blob_type = sqlite3_guess_blob_type ( $row[0] );  
}
```

```
bool sqlite3_load_spatials ( resource handle )
```

Loads the **SpatiaLite** and **VirtualShape** Spatial extensions for SQLite.

You'll need to call this function immediately after opening the SQLite DB in order to enable subsequent call to any SQL Spatial function, such as **AsText()**, **Glength()**, **Area()** and so on.

Return values are:

- TRUE if Spatial extensions have been successfully loaded
- FALSE otherwise

You must have the required loadable extension installed on your system:
/usr/lib/SpatiaLite.so
/usr/lib/VirtualShape.so

```
array sqlite3_geometry_from_blob ( string blob_value )
```

Parses *blob_value* and returns a complex array representing corresponding GEOMETRY, if possible. Return values are:

- NULL if *blob_value* doesn't contains a valid GEOMETRY
- an **array** if successful

example:

```
$rs = sqlite3_query ( $db, 'SELECT ....' );
$row = sqlite3_fetch ( $rs );
if ( sqlite3_column_type ( $rs, 0 ) == SQLITE3_BLOB )
{
    $geom = sqlite3_geometry_from_blob ( $row[0] );
    if ( $geom != NULL )
    {
        ... process geometry ...
    }
}
```

PHP supports *associative arrays* aka *dictionaries*; the one returned by `sqlite3_geometry_from_blob()` is exactly a quite complex dictionary, reflecting *OpenGis* GEOMETRY logical structure.

- a valid GEOMETRY can contain any arbitrary collection of POINTs, LINESTRINGs and POLYGONs
- a POINT [the simplest geometry] has X and Y coordinates
- a LINESTRING is an array of vertices, each of them being a POINT; a valid linestring must have at least two vertices.
- a RING is strictly analogous, but first and last vertex must coincide, in order to ensure this one is a closed figure; a valid ring must have at least four vertices.
- a POLYGON is defined by an array of rings
 - first ring is always the EXTERIOR RING [any valid polygon must have at least this one]
 - subsequent rings are INTERIOR RINGS, i.e. they identify holes inside the polygon

examples:

```
$geom = sqlite3_geometry_from_blob($blob_value);
if ($geom != NULL)
{
    $points = $geom['POINTS']; // referencing the points array
    if ($points != NULL)
    {
        $cnt = count($points); // how many points in array
        for ($i = 0; $i < $cnt; $i++)
        {
            $pt = $points[$i]; // referencing each POINT
            $x = $pt['X']; // now fetching [X,Y] coords
            $y = $pt['Y'];
        }
    }
}
```

```

$geom = sqlite3_geometry_from_blob($blob_value);
if ($geom != NULL)
{
    $lines = $geom['LINESTRINGS']; // the lines array
    if ($lines != NULL)
    {
        $cnt = count($lines); // how many lines in array
        for ($i = 0; $i < $cnt; $i++)
        {
            $ln = $lines[$i]; // referencing each line
            $cnt2 = count($ln); // how many vertices in this line
            for ($v = 0; $v < $cnt2; $v++)
            {
                $pt = $ln[$v]; // referencing each vertex
                $x = $pt['X']; // now fetching [X,Y] coords
                $y = $pt['Y'];
            }
        }
    }
}

$geom = sqlite3_geometry_from_blob($blob_value);
if ($geom != NULL)
{
    $polygons = $geom['POLYGONS']; // the polygons array
    if ($polygons != NULL)
    {
        $cnt = count($polygons); // how many polygons in array
        for ($i = 0; $i < $cnt; $i++)
        {
            $pg = $polygons[$i]; // referencing each polygon
            $cnt2 = count($pg); // how many rings in this polygon
            for ($b = 0; $b < $cnt2; $b++)
            {
                if ($b == 0)
                    $exterior = TRUE; // first ring is the EXTERION one
                else
                    $exterior = FALSE; // otherwise is an INTERIOR RING
                $ring = $pg[$b]; // referencing each ring
                $cnt3 = count($ring); // how many vertices in this ring
                for ($v = 0; $v < $cnt3; $v++)
                {
                    $pt = $ring[$v]; // referencing each vertex
                    $x = $pt['X']; // now fetching [X,Y] coords
                    $y = $pt['Y'];
                }
            }
        }
    }
}
}

```

