spatialite_gui-1.4.1 libspatialite v.2.4.0-RC5 Experimental

#1 The new spatialite_history metadata table

🧉 spatialite-gui [a GUI tool for SQLite/SpatiaLi	ite]							- 0 X
Files								
□····································	 INS SEL FRO WHE 	ERT IN ECT NU M citi RE pop	TO cities_dupl (id LL, name, populati es ulation > 500000	, name, populat on, geom	cion, geom)			
idx_cities_dupl_geom_node				1		1	1	
idx_cities_dupl_geom_parent		ROWID	event_id table_name	geometry_column	event	timestamp	ver_sqlite	ver_splite
idx_cities_dupl_geom_rowid		-	1 matial safe and	NEE 1	table successfully searched	2011-02-01-12-20-12	275	2.4.0.005
idx_cities_geom	1	1	1 spauai_rei_sys	NULL	table successfully created	2011-03-01 12:39:12	3.7.5	2.4.0 RC5
idx_cities_geom_node	2	2	2 geometry_table	NULL	table successfully created	2011-03-01 12:39:12	3.7.5	2.4.0-RC5
idx_cities_geom_parent	3	3	3 spatial_ref_sys	NULL	table succesfully populated	2011-03-01 12:39:14	3.7.5	2.4.0-RC5
idx_cities_geom_rowid	4	4	4 cities	geom	Geometry [POINT,XY,SRID=4326] succesfully created	2011-03-01 12:42:11	3.7.5	2.4.0-RC5
🗄 📠 idx_topoEdges_geom	5	5	5 cities	geom	Geometry [POINT,XY,SRID=4326] succesfully created	2011-03-01 12:44:34	3.7.5	2.4.0-RC5
idx_topoEdges_geom_node	6	6	6 cities	geom	Geometry [POINT,XY,SRID=4326] succesfully created	2011-03-01 12:45:22	3.7.5	2.4.0-RC5
idx_topoEdges_geom_parent	7	7	7 cities_dupl	geom	Geometry [POINT,XY,SRID=4326] succesfully created	2011-03-01 12:47:41	3.7.5	2.4.0-RC5
idx_topoEdges_geom_rowid	8	8	8 cities_dupl	geom	R*Tree Spatial Index succesfully created	2011-03-01 12:54:12	3.7.5	2.4.0-RC5
idx_topoNodes_geom	9	9	9 cities	geom	R*Tree Spatial Index succesfully created	2011-03-01 12:54:22	3.7.5	2.4.0-RC5
idx_topoNodes_geom_node	10	10	10 us_states	geometry	Geometry [LINESTRING,XY,SRID=4269] succesfully recovered	2011-03-01 13:03:21	3.7.5	2.4.0-RC5
	11	11	11 us_states	geometry	R*Tree Spatial Index succesfully created	2011-03-01 13:03:33	3.7.5	2.4.0-RC5
idx_topoNodes_geom_rowid	12	12	12 topoNodes	geom	Geometry [POINT.XY.SRID=4259] succesfully created	2011-03-01 13:09:57	3.7.5	2.4.0-RC5
i spatial_ref_sys	12	13	13 topoNodes	Geom	R*Tree Spatial Index successfully created	2011-03-01 13:10:26	375	2.4.0.005
	14	14	14 topoNodes	geom	Geometry [POINT XX SPID=4269] successfully created	2011-03-01 13:11:56	375	2.4.0.005
sqlite sequence	14	17	14 toponodes	geom	Difference Control Teday average de mante de	2011-03-01 13:11:30	0.7.5	2.4.0 PC5
topoEdges	15	15	15 topolyodes	geom	R*Tree Spatial Index succesfully created	2011-03-01 13:12:05	3.7.5	2.4.0-RC5
topoFaces	16	16	16 topoEdges	geom	Geometry [LINESTRING, XY, SRID=4269] succestully created	2011-03-01 13:16:43	3.7.5	2.4.0-RC5
topoNodes	17	17	17 topoEdges	geom	R*Tree Spatial Index succesfully created	2011-03-01 13:18:50	3.7.5	2.4.0-RC5
In views geometry columns	isert row							
B wirs_geometry_columns			Current block: 1/	17 [17 rows] [fetched]	n 00:00:00.014]			
Lurrent SOLite DB: F\vanuatu\istat\duol.solite								

Internal event logging:

- metadata table creation and population: InitSpatialMetadata()
- geometry column creation: AddGeometryColumn(), RecoverGeometryColumn()
- Spatial Index: CreateSpatialIndex()
- evet description / timestamp
- versioning infos

#2 Checking / removing Duplicated Rows

Please see the **cities_dupl** table.

Simply the same as **cities**, but several duplicated rows were purposely inserted using the following SQL statements:

INSERT INTO cities_dupl (id, name, population, geom)
SELECT NULL, name, population, geom
FROM cities;
INSERT INTO cities_dupl (id, name, population, geom)
SELECT NULL, name, population, geom
FROM cities
WHERE population > 100000;
INSERT INTO cities_dupl (id, name, population, geom)
SELECT NULL, name, population, geom
FROM cities
WHERE population > 200000;
INSERT INTO cities_dupl (id, name, population, geom)
SELECT NULL, name, population, geom
FROM cities
WHERE population > 200000;
INSERT INTO cities_dupl (id, name, population, geom)
SELECT NULL, name, population, geom
FROM cities
WHERE population > 500000;

🧉 spatialite-qui 🛛 [a GU	I tool for SQLite/Spat	tiaLite]					
Files							
F:\vanuatu\istat	\dupl.sqlite		URON Compt (*) AC U				
🗄 🔄 🐻 cities			CLECT Count(*) AS "	aup1-cour	ntj", "name", "po	opulation", "geom"	
ities_dupl		GR	OUP BY "name", "pop	ulation",	"geom"		
geom_cols	Table: cities_dupl		VING "[dupl-count]	> 1			
i geometry_	Refresh		DER BY "[dupl-count] " DESC			
geometry_	recircular and a second						
	😿 Query/View Comp	ooser					
idx_cities_c -	Create New Table						
idx_cities_c	Create New View		countj name	population	geom		
- , idx cities c			4 Genova	601951	BLOB sz=60 GEOMETRY		
	Edit table rows		4 Milano	1306661	BLOB sz=60 GEOMETRY		
	Show CREATE stat		4 Napoli	988972	BLOB sz=60 GEOMETRY		
idx_cities_c _	Show CREATE stat	ement	4 Palermo	672175	BLOB sz=60 GEOMETRY		
🗄 - 🌉 idx_topoEd	Maintenance	×.	Add New Column	+1	BLOB SZ=60 GEOMETRY		
idx_topoEd	Export as Txt/Tab		Rename table	33	DLUD SZ=60 GEOMETRY		
	Export as CSV		Drop table	52	BLOB SZ=60 GEOMETRY		
idx_topoEd	Export as HTML			10	BLOB 52=60 GEOMETRY		
idx_topoNod	les geom node	10	Create New Index	17	BLOB SZ=60 GEOMETRY		
idx topoNod	des geom parent	11	Create New Trigger		BLOB SZ=60 GEOMETRY		
idx_topoNod	les_geom_rowid	12	Check Duplicate rows	70	BLOB SZ=60 GEOMETRY		
ig spatial_ref_sy	ys	13	Remove Duplicate row	. 33	BLOB sz=60 GEOMETRY		
🕴 🐵 📶 spatialite_his	tory	14		211184	BLOB sz=60 GEOMETRY		
🗄 🗄 🛄 sqlite_sequer	nce	15	3 Venice	270816	BLOB sz=60 GEOMETRY		
Image:		16	3 Verona	253208	BLOB sz=60 GEOMETRY		
topoFaces		17	2 Ancona	100.507	BLOB sz=60 GEOMETRY		
i views geom	eta: columns	18	2 Bergamo	113143	BLOB sz=60 GEOMETRY		
wirts geomet	try columns	19	2 Brescia	187567	BLOB sz=60 GEOMETRY		
i i i i i i i i i i i i i i i i i i i		20	2 Cagliari	164249	BLOB sz=60 GEOMETRY		
		21	2 Ferrara	130992	BLOB sz=60 GEOMETRY		
		22	2 Foggia	155203	BLOB sz=60 GEOMETRY		
		23	2 Forli	108335	BLOB sz=60 GEOMETRY		
		24	2 Latina	107898	BLOB SZ=60 GEOMETRY		
		25	2 Livorno	156274	BLOB SZ=60 GEOMETRY		
		26	2 Mestre	176000	PLOB SZ=50 GEOMETRY		
		2/	2 Monza	170502	BLOB SZ=60 GEOMETRY		
		28	2 Novara	100910	BLOB SZ=60 GEOMETRY	-	
		20	2 Novelle 2 Parma	163457	BLOB 52-50 GEOMETRY		
		31	2 Perugia	149125	BLOB SZ=60 GEOMETRY		
		32	2 Pescara	116286	BLOB SZ=60 GEOMETRY		
		33	2 Pinocchio di Ancona	102117	BLOB sz=60 GEOMETRY		
		34	2 Prato	172499	BLOB sz=60 GEOMETRY		
		3.5	2 Ravenna	134631	BLOB sz=60 GEOMETRY		
		36	2 Rennin di Calabria	180353	BLOB \$7=60 GEOMETRY		
			🖉 🕨 🕅 current	block: 1 / 44 [4	H rows] [fetched in 00:0	00:00.085]	
Connect COLite DB: E \							
Current SQLITE DB: F:\Vanu	uacu (istat \cupi.sqlite						

The **Check** tool will identify any duplicate row (please note: any **Primary Key** column will be ignored). And the **Remove** tool will eventually delete any duplicate row except the first one.

The same tool is supported by the spatialite CLI front end as .chkdupl and .remdupl

#3 Rudimentary Topology support



Please see the topoNodes, topoEdges and topoFaces tables.

They simply contain (alltogether) a topological representation of the United States (derived from the U.S. Census Bureau TIGER dataset).

As you can easily notice, there is no direct representation of *States as polygons*.

- an **Edge** is a LINESTRING representing a common boundary shared by to two adjacent States
- a Node is a POINT where two (or more) Edges intersects
- so each single State is represented by one (or more) **Faces**; but a **Face** simply is represented as a collection of **Edges** (i.e. the ones delimiting the face's own boundary). No explicit POLYGON is represented at the topolological level. And there is no GEOMETRY directly corresponding to a **Face**.



SELECT f.face_id, Polygonize(Collect(e.geom)) FROM topoFaces AS f JOIN topoEdges AS e ON (e.edge_id = f.edge_id) GROUP BY f.face id;

Collect() (aggregate function) will create a MULTILINESTRING corresponding to the complete boundary delimiting a **Face**.

And then Polygonize() will reconstruct a POLYGON representing the same Face.

Nobody forbids us to create a further table representing State-Faces as POLYGONs:

```
CREATE TABLE state_polygs (
face_id INTEGER NOT NULL PRIMARY KEY);
SELECT AddGeometryColumn('state_polygs', 'geom', 4269, 'POLYGON',
'XY');
INSERT INTO state_polygs (face_id, geom)
SELECT f.face_id, Polygonize(Collect(e.geom))
FROM topoFaces AS f
JOIN topoEdges AS e ON (e.edge_id = f.edge_id)
GROUP BY f.face_id;
```

This is useful in order to test another two SQL functions recently introduced:

```
SELECT e.edge_id, e.geom
FROM state_polygs AS s
JOIN topoEdges AS e ON (CoveredBy(e.geom, s.geom))
WHERE s.face id = 108;
```

This first query will identify any Edge covered by the Face corresponding to the Texas State.

```
SELECT s.face_id, s.geom
FROM topoEdges AS e
JOIN state_polygs AS s ON (Covers(s.geom, e.geom))
WHERE e.edge id = 156;
```

And this second query will identify the two States sharing a common Edge (in this example: Texas and Louisiana).

#4 reconstruncting a GPS track from WayPoints

This time we'll use the **gps_track** table. As you can notice this table actually contains several GPS WayPoints:

```
SELECT MakeLine(MakePoint(longitude, latitude, 4326))
FROM gps_track
GROUP BY track no;
```

Using the MakeLine() aggregate function you can easily get the whole GPS Track as a LINESTRING.

```
SELECT MakeLine(MakePoint(longitude, latitude, 4326))
FROM gps_track
WHERE gps_timestamp BETWEEN
   '2011-02-14T14:45' AND
   '2011-02-14T14:55'
GROUP BY track_no;
```

And you can obviously extract a specific portion of this GPS track setting an appropriate time interval: in this example the track walked on 2011-02-14 starting at 14:45 and ending at 14:55

#5 exporting KML files

We'll start again from the cities table.



Just to make things a little bit difficulties, we'll create first a **main_cities** VIEW (filtering towns > 250,000 peoples). Then we'll properly register this VIEW into **view_geometry_columns**, so to get a real Spatial VIEW.



1	Dump KML	×
	Table name:	cities
	GeomColumn name:	geom
		Precision 8
	Name from Column name	Constant value
	Description from Column population	Constant value
		OK Cancel

You must specify two columns: one corresponding to the **<name>** tag, the other corresponding to the **<descricption>** tag: anyway, you can specify a constant string if no such column exist.



Once you've exported the KML file, you can perform a direct check using **Google Earth** (or any other appropriate sw supporting KML).