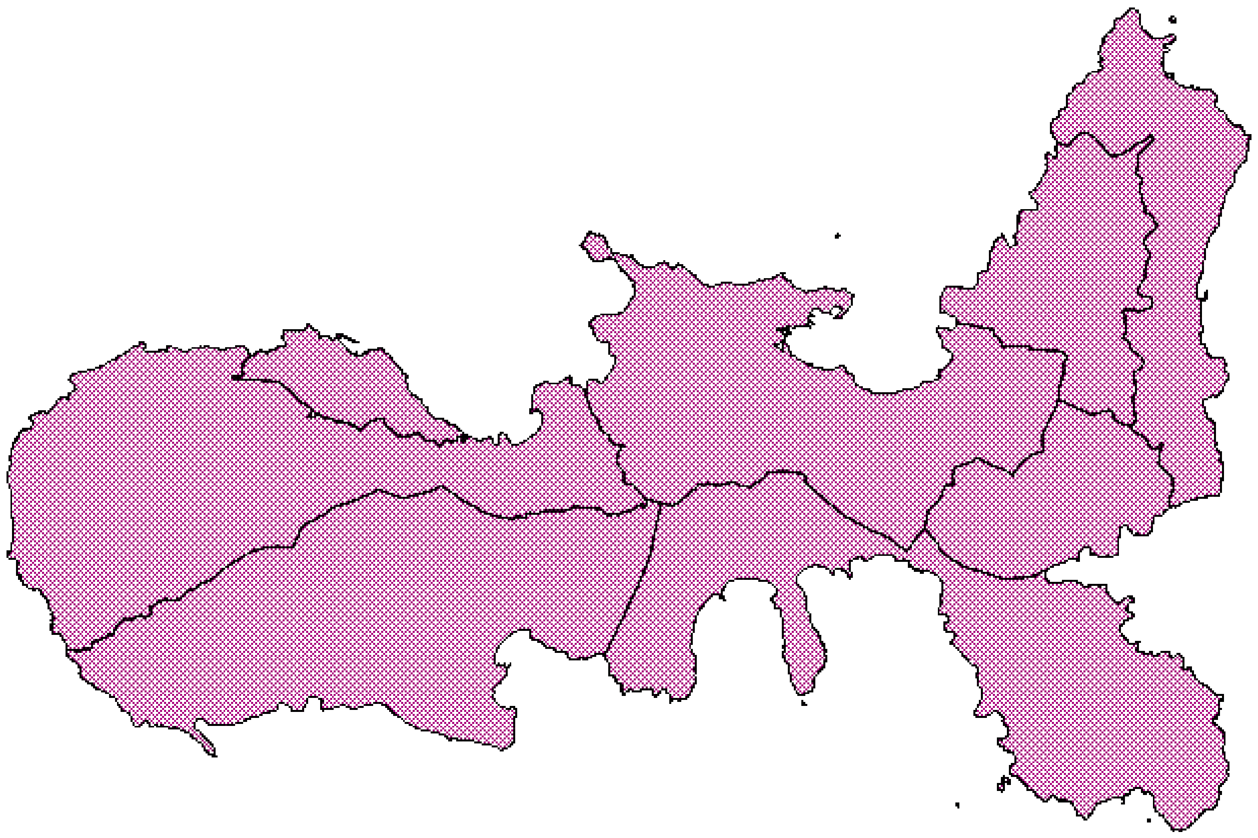
A quick introduction to SpatiaLite Topology

*** D R A F T ***

OGC-SFS Geometries

As you already know, the standard representation of Geometries supported by any existing Spatial DBMS is based on the following SFS classes (non-topological):

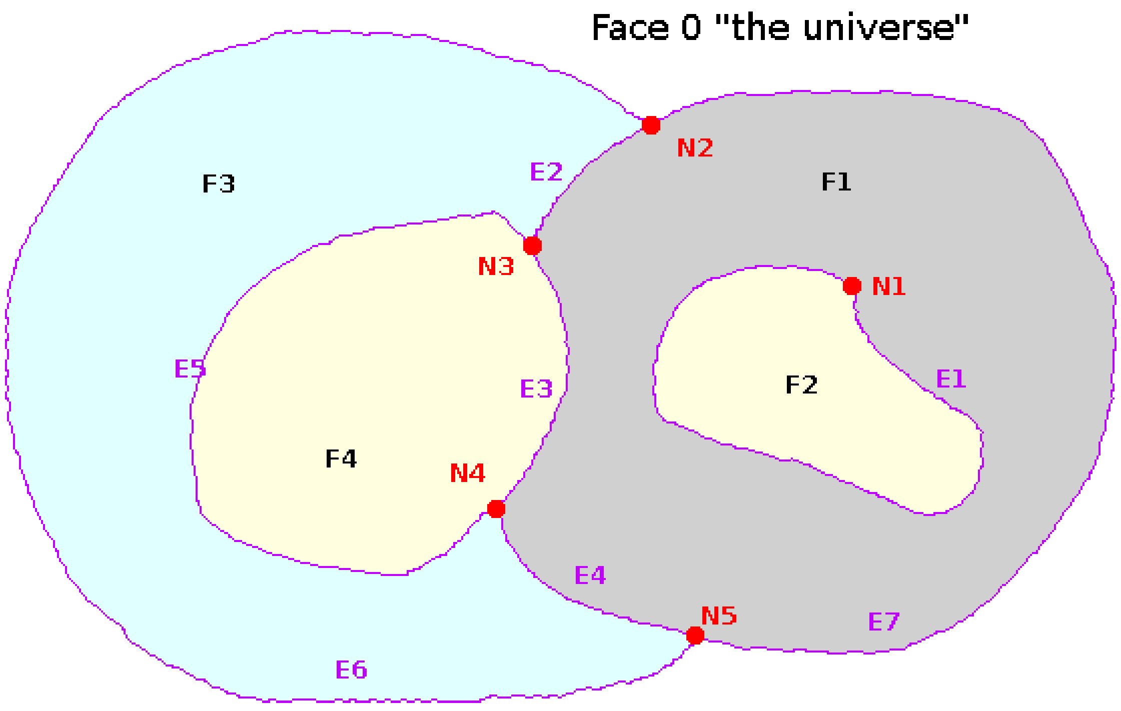
- (MULTI)POINT
- (MULTI)LINESTRING
- (MULTI)POLYGON
- GEOMETRYCOLLECTION



The above figure (Elba Island) represents a typical vector layer of the MultiPolygon type; each Local Council is represented by a corresponding MultiPolygon Geometry.
There is nothing wrong in this representation: standard SFS Geometries support many useful algorithms, and can be processed in a fast, highly optimized way.
Anyway a potential design flaw exists: each single Geometry is a self-standing entity, and there is no way to represent special relationships shared by two (or more) different Geometries. Quite obviously the **borderline** dividing two confining Local Councils is one and the same for both them; but SFS Geometries doesn't allow at all to explicitly represent such borderline.
As a practical consequence, any change (editing) affecting one or both Geometries will very probably introduce some spatial inconsistency (overlapping areas, unfilled gaps, and so on).

Topology

An alternative representation (planar graph) is based on the following elements:



- a **NODE** simply is a notable point, and can be assumed to be equivalent to SFS POINT. *Examples*: ***N1, N2, N3, N4, N5***
- an **EDGE** is an oriented path joining two nodes, and can be assumed to be equivalent to SFS LINESTRING. *Examples*: ***E1, E2, E3, E4, E5, E6, E7***
- a **FACE** is a portion of the plane delimited by edges, and can be assumed to be equivalent to SFS POLYGON. *Examples*: ***F1, F2, F3, F4***
- a **TopoCurve** is a collection of one (or more) Edges, and can be assumed to be equivalent to SFS MULTILINESTRING.
- a **TopoSurface** is a collection of one (or more) Faces, and can be assumed to be equivalent to SFS MULTIPOLYGON. *Example*: Faces ***F2*** and ***F4*** belongs to the same TopoSurface.

Constraints on Nodes:

- two (or more) Nodes can never overlap.

Constraints on Edges:

- each Edge always has a *node-from* and a *node-to*: this implies that any Edge is *oriented* (has a direction).
- *node-from* and a *node-to* may be the same: and in this case we have a *self-closed* Edge (aka Ring). *Example*: ***E1***
- an Edge cannot contain loops (or any other kind of self-intersection).
- two (or more) Edges can intersect only where a Node is defined. *Example*: Edges ***E3***, ***E4*** and ***E5*** intersect exactly at Node ***N4***
- no Node can overlap an Edge except than at its extreme points.

Constraints on Faces:

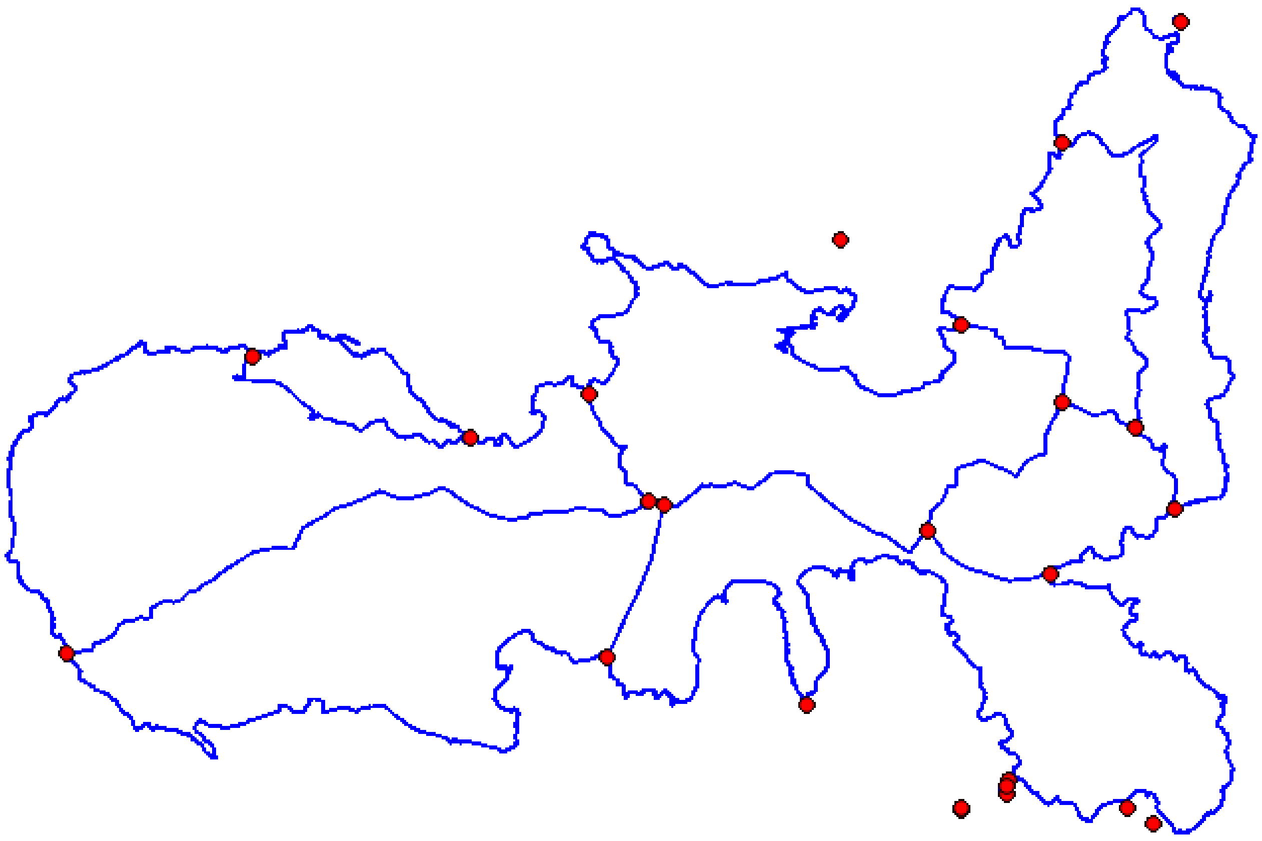
- each Face is delimited by one (or more) Edges.
- each Edge always separates two different Faces.

The Universe: **Face 0**

- a fictitious **Face 0** (aka "*The Universe*") is silently assumed to exist; Face 0 covers the whole plane where no other Face is explicitly defined.
- as a consequence, any Edge surely always divides two different Faces:
  - e.g. the Edge ***E3*** separates the Faces ***F1*** and ***F4***
  - whilst Edge ***E6*** separates Face ***F3*** and the universal Face 0.

Representation of Faces:

- as already stated, each Face is delimited by a set of Edges: *Example*: Face ***F3*** is delimited by Edges ***E2, E5, E4*** and ***E6***
- a Face can be delimited by a single Edge: *Example*: Face ***F2*** is delimited by Edge ***E1***
- a Face must always have an *exterior boundary*; but can legitimately have one (or more) *interior boundaries* (aka holes) at the same time. *Example*: Face ***F1*** is delimited by Edges ***E1, E2, E3*** and ***E7***
  - Edges ***E2, E3*** and ***E7*** represent the *exterior boundary*
  - Edge ***E1*** represents an *interior boundary* (i.e. a hole).

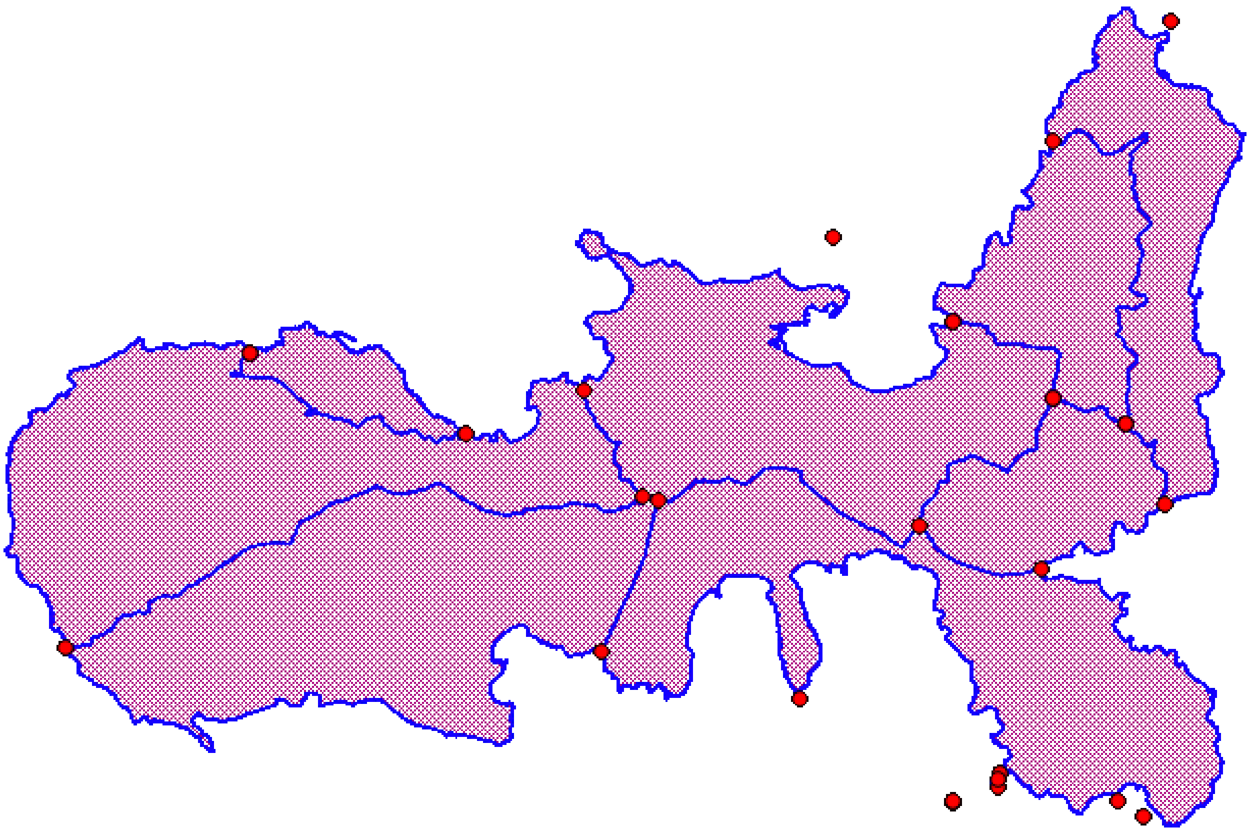


The Elba Island, again: this time represented as a network (aka graph) of Nodes and Edges.
<u>Please note well</u>: now there are no Polygons or MultiPolygons at all; we have simply identified Topology elementary items:

- an Edge has been defined for each borderline between two adjacent Local Councils (or between a Local Council and the sea, in this case corresponding to Face 0 "the universe").
- Nodes have been defined where two (or more) Edges intersect.

Real and Virtual Geometries:

- Each Node is represented by a *real* Geometry, of the POINT type.
- Each Edge is represented by a *real* Geometry, of the LINESTRING type.
- Faces, TopoCurves and TopoSurfaces has no *real* Geometry at all; all them simply are *virtual* Geometries.
- Faces and TopoCurves are simply defined as sets of Edges: accordingly to this, building a *real* Geometry corresponding to some Face (or TopoCurve) requires a heavy computational load (and obviously, is a slow process).
  - *real* Geometry corresponding to a single Face always is of the POLYGON type.
  - *real* Geometry corresponding to a single TopoCurve always is of the MULTILINSTRING type.
- TopoSurfaces are defined as sets of Faces: accordingly to this, building a *real* Geometry corresponding to some TopoSurface requires to build in turn any required Face: and obviously, this is an even heavier and slower process.



The Elba Island, yet another time: now Nodes and Edges are still represented; but this time any Face / TopoSurface is represented as well.
i.e. *virtual* items have now been properly <u>*materialized*</u> so to become visible.

Pros and Cons in using Topology:

- **Pros**:
  - any special spatial relationship is clearly and unambiguously defined.
  - Editing/modifying the borderline between two different Local Councils now requires editing the corresponding Edge (and, may be, some Node).
  - such change will surely affect both Local Councils at the same time, thus preserving a strong spatial consistency.
- **Cons:**
  - high-level objects (Faces, TopoCurves and TopoSurfaces) simply are *virtual*.
  - materializing the *real* representation for *virtual* objects is a heavy (and slow) task.
  - very few GIS tools support Topological editing; notably, not the most widely used ones.

Topology support implemented in SpatiaLite 3.0.x

SpatiaLite is an unsophisticated, easy to be used and light-weight Spatial DBMS mainly intended for personal (Desktop) usage.
So don't expect too much, a complete and comprehensive implementation of Topology is not yet available (at least, not for the current release … may well be in the future, who knows).

The initial goals of this first Topology support are not at all too much ambitious, and are simply limited to support the following very basic tasks:

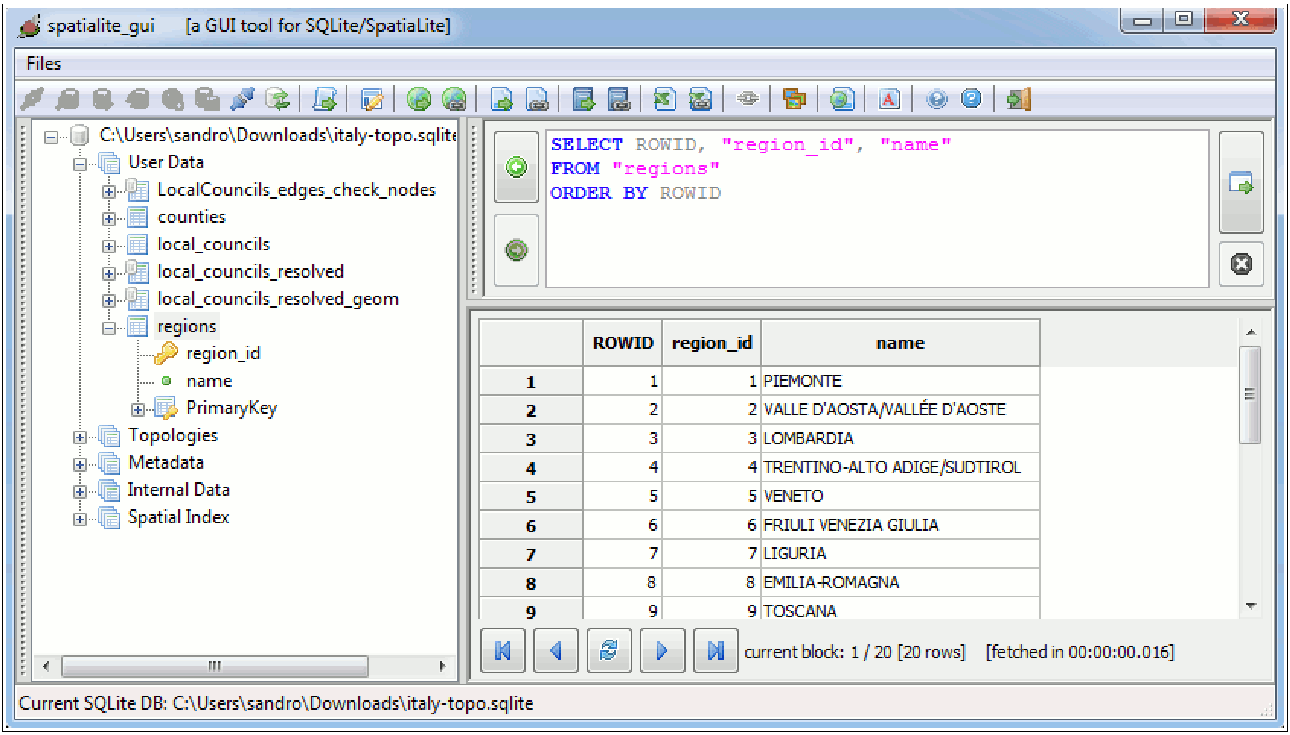
- storing a full Topology (based on Nodes, Edges, Faces, TopoCurves and TopoSurfaces) within a DB-file
- supporting a minimal set of validation tools useful to check for sanity and consistency.
- reconstructing *classic* SFS Geometries (MULTILINESTRINGs and MULTIPOLYGONs) starting from their corresponding topological representations.

A practical tutorial

You can download a SpatiaLite DB containing a full Topology sample (Local Councils of Italy) from this URL: http://www.gaia-gis.it/gaia-sins/italy-topo.7z
This DB is compressed (7-Zip) and has a size of about 15.5 MB: once you've downloaded (and uncompressed) the DB-file you can directly open it using the latest **spatialite_gui-1.5.0**

As you can immediately notice, this DB doesn't contains any Geometry Table at all: don't be fooled by this quick first glance; this one actually is a Spatial DBMS, after all.

Let us go to explore in further depth the general DB layout, table by table and view by view ...


spatialite_gui [a GUI tool for SQLite/SpatiaLite]
Files
C:\Users\sandro\Downloads\italy-topo.sqlite
User Data
LocalCouncils_edges_check_nodes
counties
local_councils
local_councils_resolved
local_councils_resolved_geom
regions
region_id
name
PrimaryKey
Topologies
Metadata
Internal Data
Spatial Index

```
SELECT ROWID, "region_id", "name"
FROM "regions"
ORDER BY ROWID
```

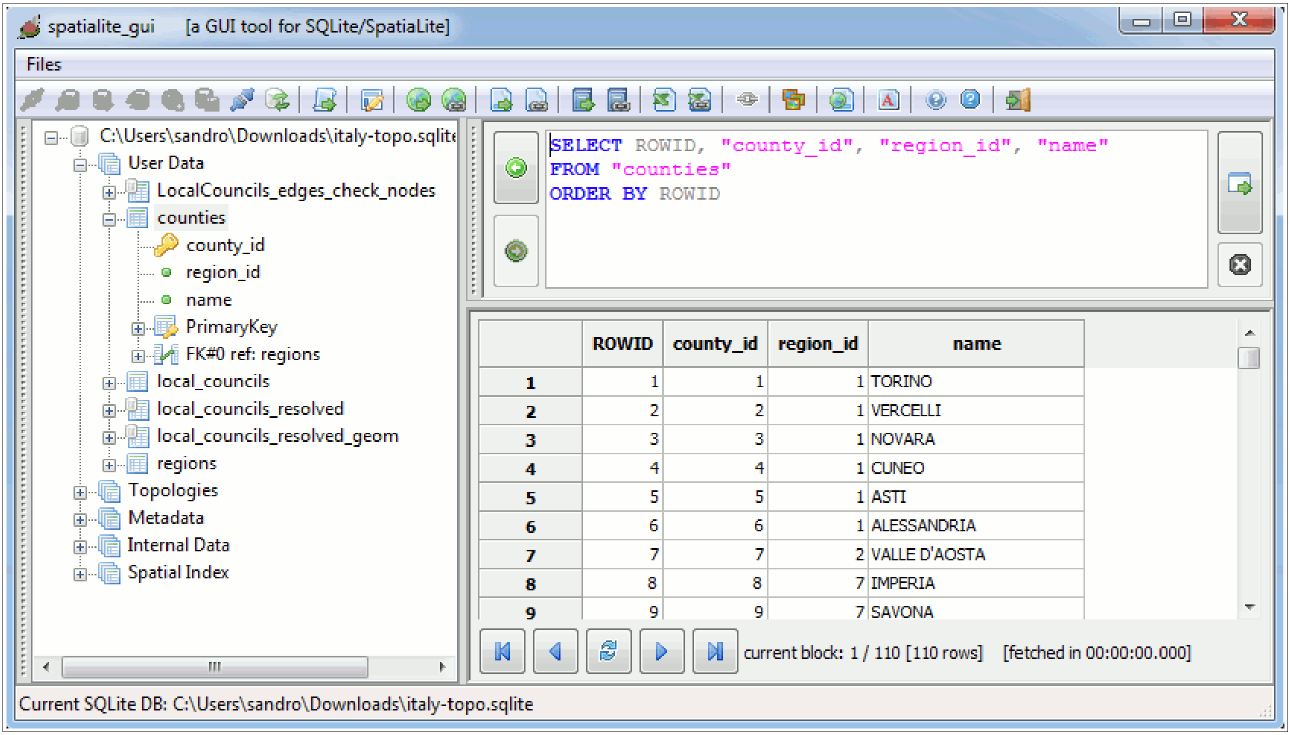
	ROWID	region_id	name
1	1	1	PIEMONTE
2	2	2	VALLE D'AOSTA/VALLÉE D'AOSTE
3	3	3	LOMBARDIA
4	4	4	TRENTINO-ALTO ADIGE/SUDTIROL
5	5	5	VENETO
6	6	6	FRIULI VENEZIA GIULIA
7	7	7	LIGURIA
8	8	8	EMILIA-ROMAGNA
9	9	9	TOSCANA

current block: 1 / 20 [20 rows]   [fetched in 00:00:00.016]

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

regions

a plain Table, simply containing ids and names for Italy's Regions; not very interesting.



```
SELECT ROWID, "county_id", "region_id", "name"
FROM "counties"
ORDER BY ROWID
```

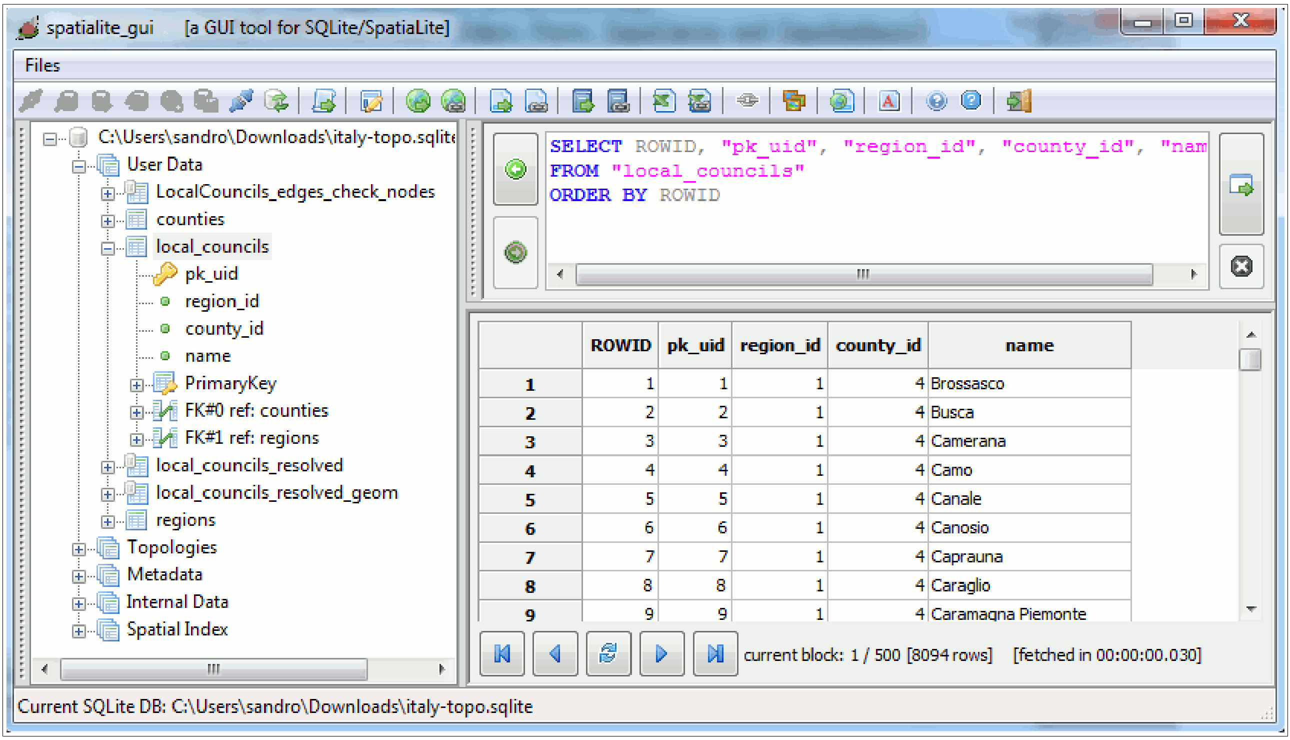
	ROWID	county_id	region_id	name
1	1	1	1	TORINO
2	2	2	1	VERCELLI
3	3	3	1	NOVARA
4	4	4	1	CUNEO
5	5	5	1	ASTI
6	6	6	1	ALESSANDRIA
7	7	7	2	VALLE D'AOSTA
8	8	8	7	IMPERIA
9	9	9	7	SAVONA

current block: 1 / 110 [110 rows]   [fetched in 00:00:00.000]

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

counties

yet another plain Table representing Italy's Counties; not at all exciting.
This time we have a Foreign Key referencing **regions**, but there isn't the slightest trace of Geometry.


spatialite_gui [a GUI tool for SQLite/SpatiaLite]

```
C:\Users\sandro\Downloads\italy-topo.sqlite
  User Data
    LocalCouncils_edges_check_nodes
    counties
    local_councils
      pk_uid
      region_id
      county_id
      name
      PrimaryKey
      FK#0 ref: counties
      FK#1 ref: regions
    local_councils_resolved
    local_councils_resolved_geom
    regions
  Topologies
  Metadata
  Internal Data
  Spatial Index
```

```sql
SELECT ROWID, "pk_uid", "region_id", "county_id", "nam
FROM "local_councils"
ORDER BY ROWID
```

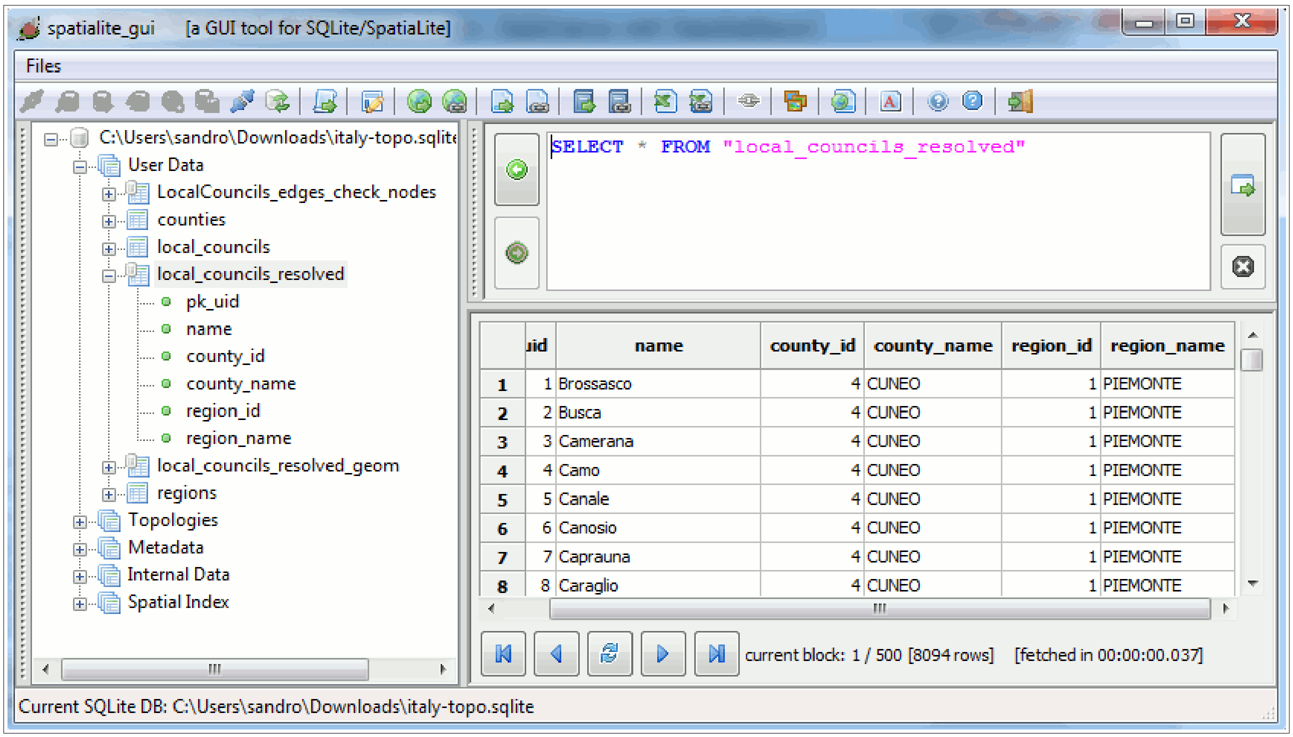
	ROWID	pk_uid	region_id	county_id	name
1	1	1	1	4	Brossasco
2	2	2	1	4	Busca
3	3	3	1	4	Camerana
4	4	4	1	4	Camo
5	5	5	1	4	Canale
6	6	6	1	4	Canosio
7	7	7	1	4	Caprauna
8	8	8	1	4	Caraglio
9	9	9	1	4	Caramagna Piemonte

current block: 1 / 500 [8094 rows] [fetched in 00:00:00.030]

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

local_councils

again, a plain Table representing Italy's Local Councils: there are two Foreign Keys, one referencing **regions**, the other referencing **counties**. And again, there is no Geometry.


spatialite_gui [a GUI tool for SQLite/SpatiaLite]

```
C:\Users\sandro\Downloads\italy-topo.sqlite
  User Data
    LocalCouncils_edges_check_nodes
    counties
    local_councils
    local_councils_resolved
      pk_uid
      name
      county_id
      county_name
      region_id
      region_name
    local_councils_resolved_geom
    regions
  Topologies
  Metadata
  Internal Data
  Spatial Index
```

```sql
SELECT * FROM "local_councils_resolved"
```

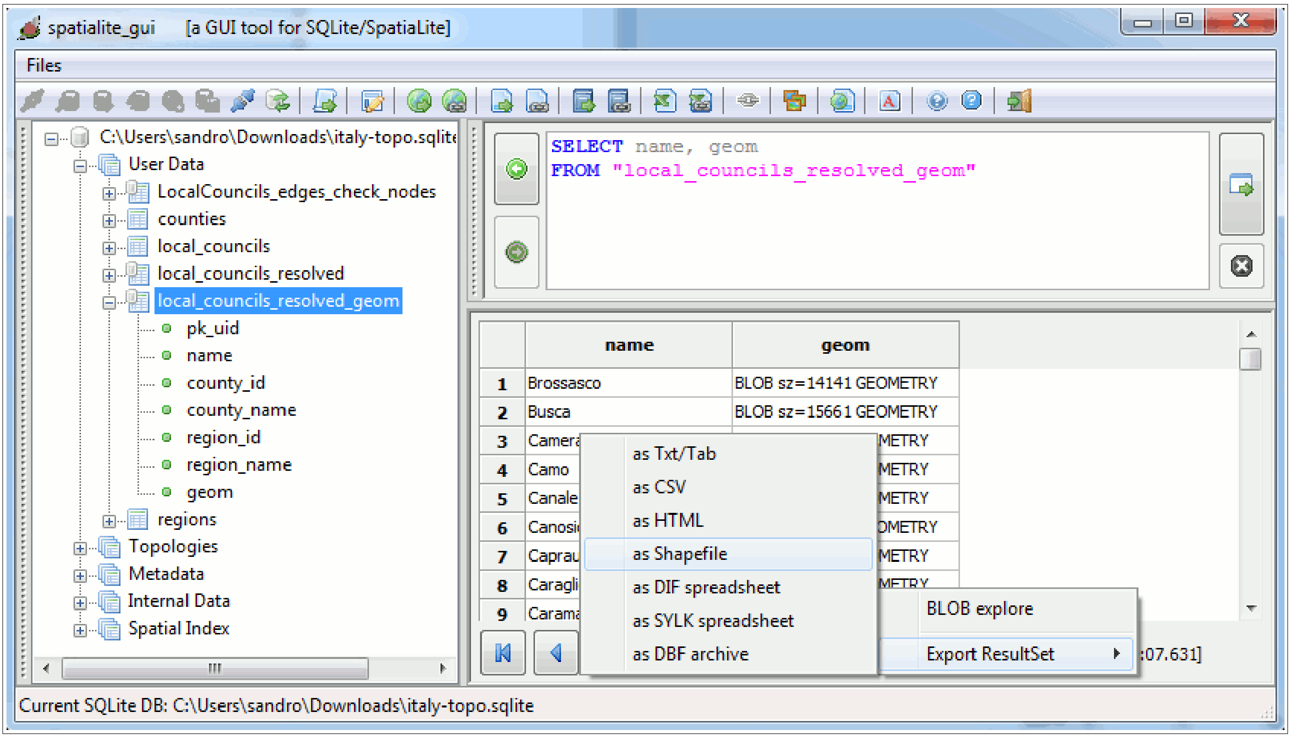
	uid	name	county_id	county_name	region_id	region_name
1	1	Brossasco	4	CUNEO	1	PIEMONTE
2	2	Busca	4	CUNEO	1	PIEMONTE
3	3	Camerana	4	CUNEO	1	PIEMONTE
4	4	Camo	4	CUNEO	1	PIEMONTE
5	5	Canale	4	CUNEO	1	PIEMONTE
6	6	Canosio	4	CUNEO	1	PIEMONTE
7	7	Caprauna	4	CUNEO	1	PIEMONTE
8	8	Caraglio	4	CUNEO	1	PIEMONTE

current block: 1 / 500 [8094 rows] [fetched in 00:00:00.037]

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

local_councils_resolved

this one isn't a Table, it's a View instead; and is simply intended to resolved in the explicit form any relational reference between **local_councils**, **counties** and **regions**.



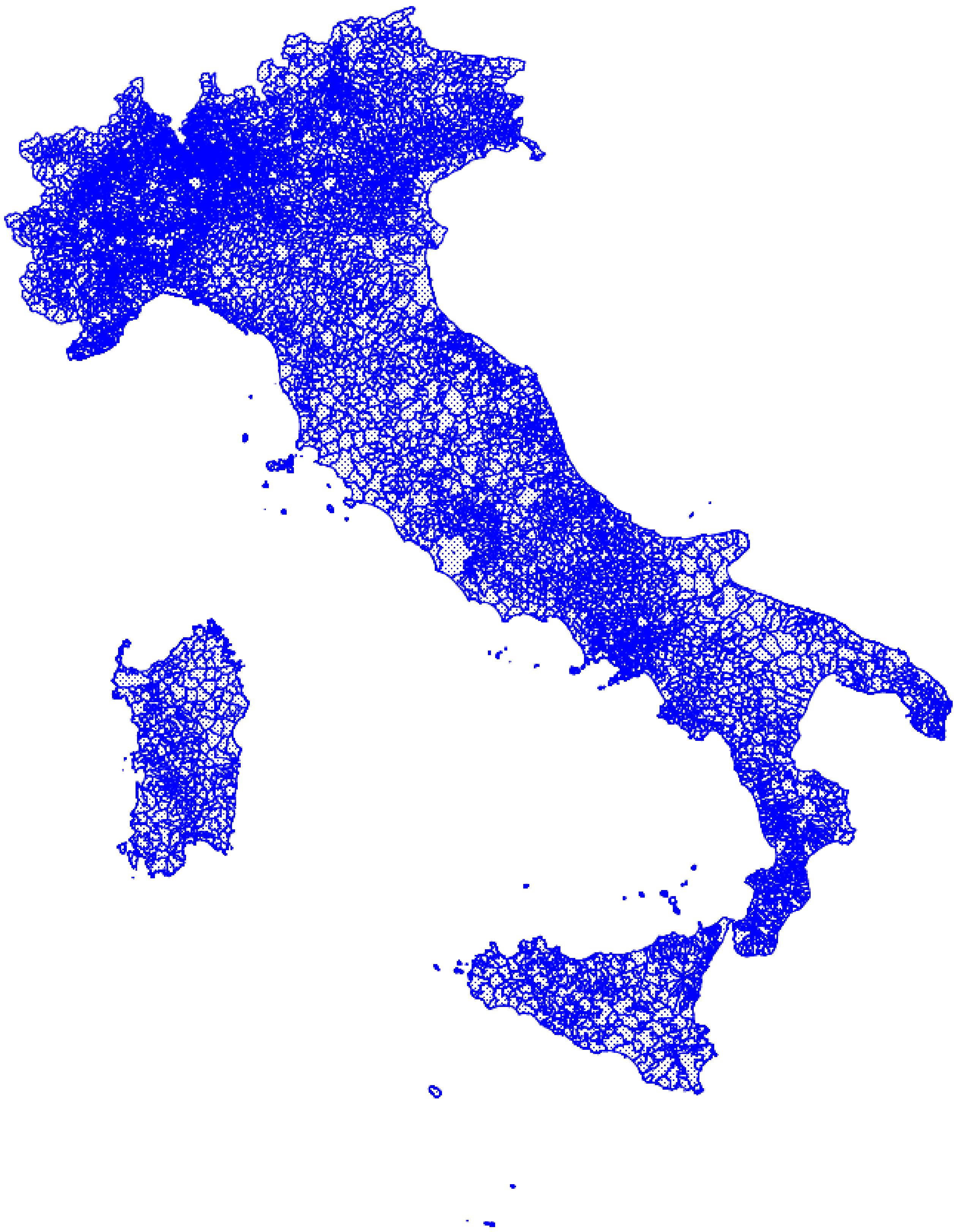
```
SELECT name, geom
FROM "local_councils_resolved_geom"
```

	name	geom
1	Brossasco	BLOB sz=14141 GEOMETRY
2	Busca	BLOB sz=15661 GEOMETRY
3	Camera	METRY
4	Camo	METRY
5	Canale	METRY
6	Canosi	OMETRY
7	Caprau	METRY
8	Caragli	METRY
9	Carama	

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

local_councils_resolved_geom

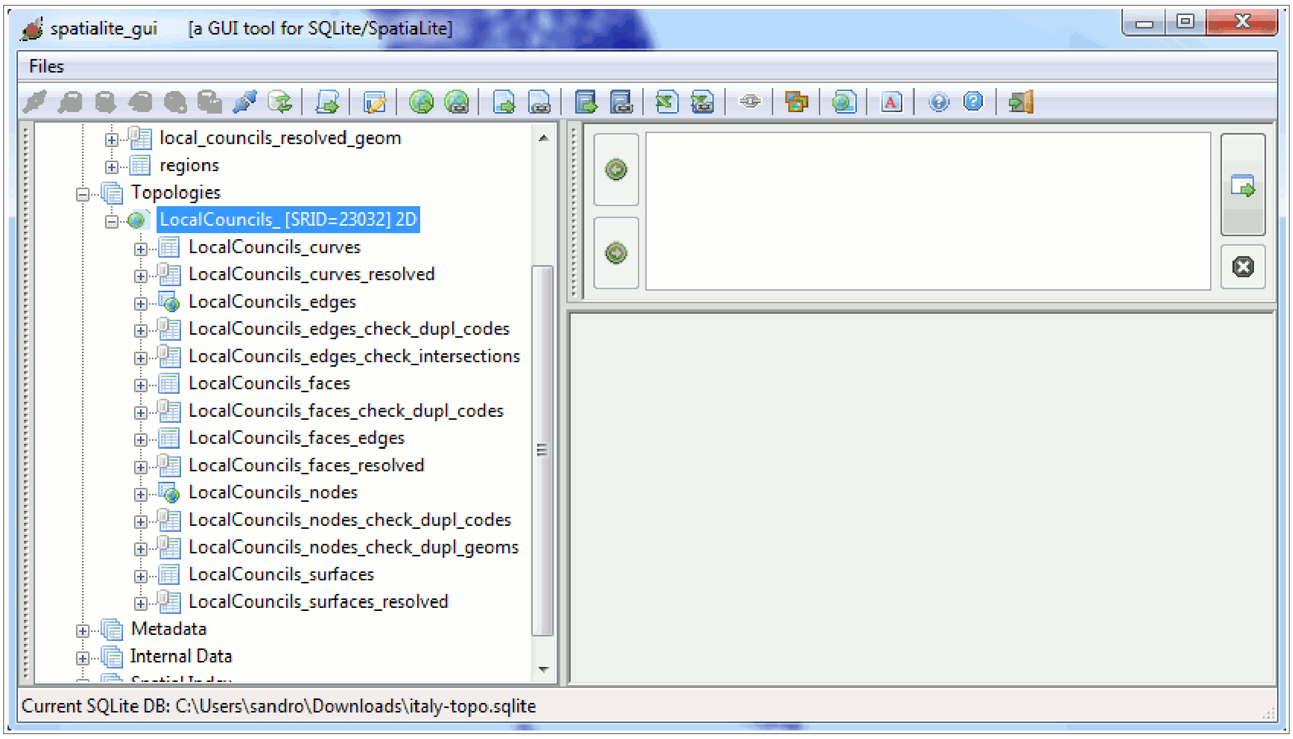
yet another View, strictly similar to the previous one: except in that this time we (finally) have a Geometry column; and we can export the corresponding Shapefile using the context menu (*right click*).



We can now visualize the Shapefile previously exported using any GIS viewer.
All right, this one is a full map layer representing Italy's Local Councils.

So a question now naturally arises:

from where come out these Geometries, considering that no other Table explored since now defines any Geometry at all ?


spatialite_gui [a GUI tool for SQLite/SpatiaLite]

Files

- local_councils_resolved_geom
- regions
- Topologies
  - LocalCouncils_ [SRID=23032] 2D
    - LocalCouncils_curves
    - LocalCouncils_curves_resolved
    - LocalCouncils_edges
    - LocalCouncils_edges_check_dupl_codes
    - LocalCouncils_edges_check_intersections
    - LocalCouncils_faces
    - LocalCouncils_faces_check_dupl_codes
    - LocalCouncils_faces_edges
    - LocalCouncils_faces_resolved
    - LocalCouncils_nodes
    - LocalCouncils_nodes_check_dupl_codes
    - LocalCouncils_nodes_check_dupl_geoms
    - LocalCouncils_surfaces
    - LocalCouncils_surfaces_resolved
- Metadata
- Internal Data

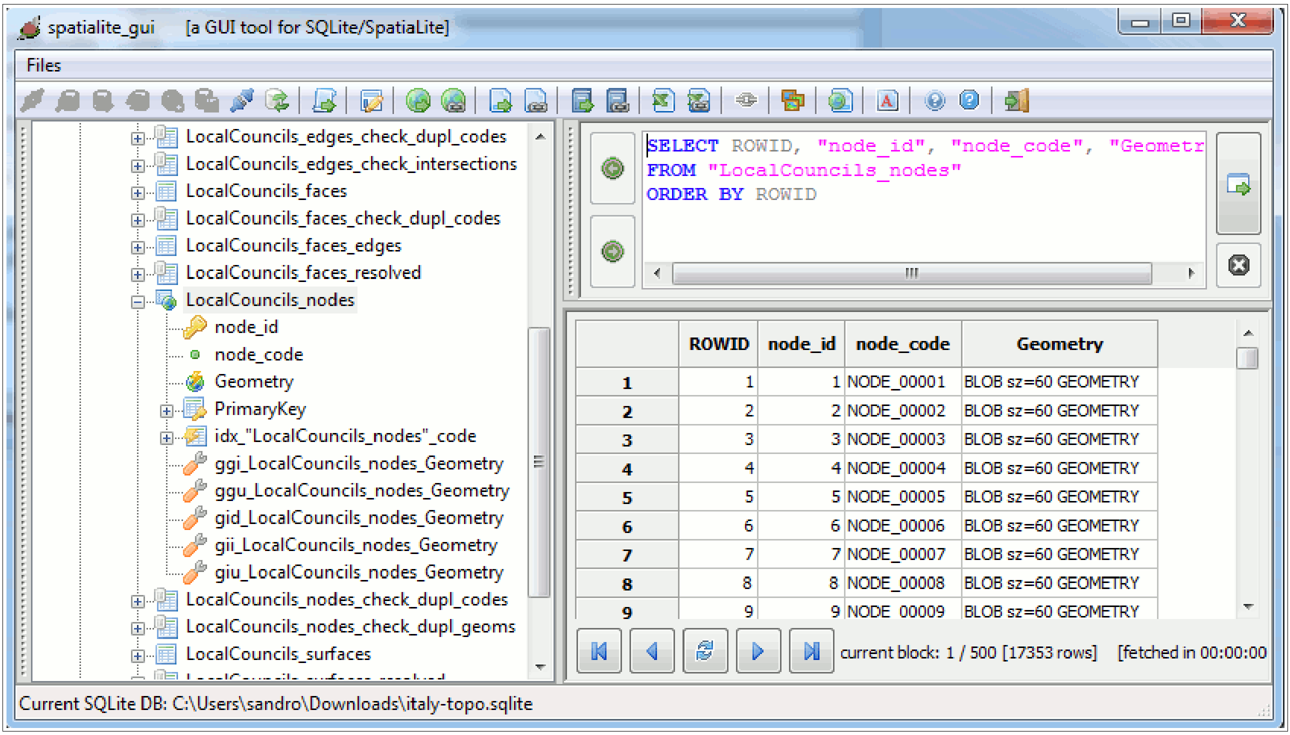
Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

Here is the answer to the previous question.

The current DB contains a full set of Topology Tables:

- a set of Topology Tables is identified by a common prefix: i.e. ***LocalCouncils_*** in this example.
- any Geometry in the same Topology Set must always adopt the same SRID and Dimensions.
- the same DB can eventually contain more than one single Topology set (each one of them being identified by a different prefix).
- when more Topology sets exist into the same DB, each one of them is completely insulated and independent from any other one.
- any Topology set always implements the same general layout for Tables and Views.

Let now go exploring the Topology Set layout in full detail.



LocalCouncils_edges_check_dupl_codes
LocalCouncils_edges_check_intersections
LocalCouncils_faces
LocalCouncils_faces_check_dupl_codes
LocalCouncils_faces_edges
LocalCouncils_faces_resolved
LocalCouncils_nodes
node_id
node_code
Geometry
PrimaryKey
idx_"LocalCouncils_nodes"_code
ggi_LocalCouncils_nodes_Geometry
ggu_LocalCouncils_nodes_Geometry
gid_LocalCouncils_nodes_Geometry
gii_LocalCouncils_nodes_Geometry
giu_LocalCouncils_nodes_Geometry
LocalCouncils_nodes_check_dupl_codes
LocalCouncils_nodes_check_dupl_geoms
LocalCouncils_surfaces

```
SELECT ROWID, "node_id", "node_code", "Geometr
FROM "LocalCouncils_nodes"
ORDER BY ROWID
```

	ROWID	node_id	node_code	Geometry
1	1	1	NODE_00001	BLOB sz=60 GEOMETRY
2	2	2	NODE_00002	BLOB sz=60 GEOMETRY
3	3	3	NODE_00003	BLOB sz=60 GEOMETRY
4	4	4	NODE_00004	BLOB sz=60 GEOMETRY
5	5	5	NODE_00005	BLOB sz=60 GEOMETRY
6	6	6	NODE_00006	BLOB sz=60 GEOMETRY
7	7	7	NODE_00007	BLOB sz=60 GEOMETRY
8	8	8	NODE_00008	BLOB sz=60 GEOMETRY
9	9	9	NODE_00009	BLOB sz=60 GEOMETRY

current block: 1 / 500 [17353 rows] [fetched in 00:00:00

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

*prefix*_nodes

this Table contain any Node used by the current Topology:

- a Node is *real*, so a corresponding Geometry (of the POINT type) must always be defined for each Node.
- the ***node_id*** simply is a Primary Key uniquely identifying the Node.
- the ***node_code*** is a symbolic name identifying the Node: it's equivalent to the **gml:id** field required by GML 3 Topology.

related Views (supporting validity checks):

```
CREATE VIEW "LocalCouncils_nodes_check_dupl_codes" AS
SELECT node_code AS node_code, Count(node_id) AS count
FROM "LocalCouncils_nodes"
GROUP BY node_code
HAVING count > 1
```

*prefix*_nodes_check_dupl_codes

this View will identify any duplicate ***node_code*** value.

```
CREATE VIEW "LocalCouncils_nodes_check_dupl_geoms" AS
SELECT n1.node_id AS node1_id, n1.node_code AS node1_code, n2.node_id AS
node2_id, n2.node_code AS node2_code
FROM "LocalCouncils_nodes" AS n1
JOIN "LocalCouncils_nodes" AS n2 ON (
  n1.node_id <> n2.node_id AND
  ST_Equals(n1.Geometry, n2.Geometry) = 1 AND
  n2.node_id IN (
      SELECT ROWID FROM SpatialIndex
      WHERE f_table_name = 'LocalCouncils_nodes' AND
        search_frame = n1.Geometry))
```

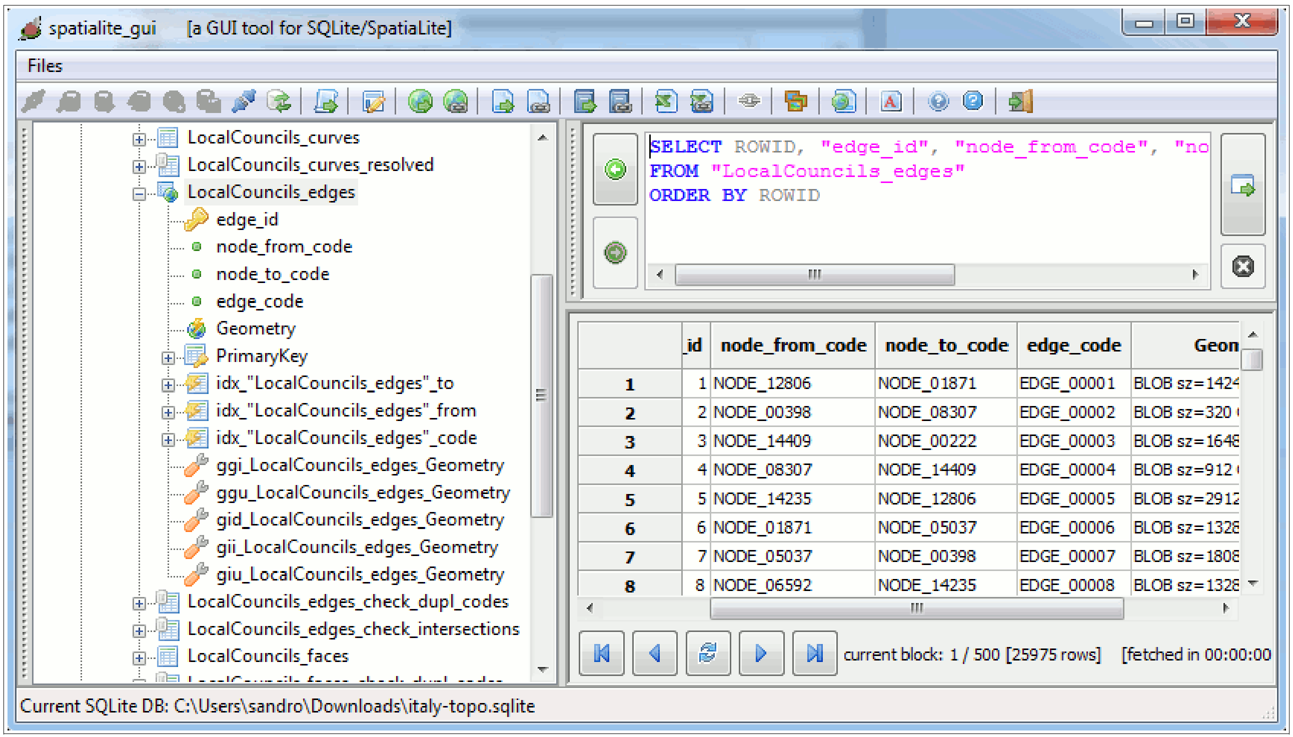
*prefix*_nodes_check_dupl_geoms

this View will identify any couple of overlapping Nodes.

```
CREATE VIEW "LocalCouncils_dangling_nodes" AS
SELECT n.node_id AS node_id
FROM "LocalCouncils_nodes" AS n
LEFT JOIN "LocalCouncils_edges" AS e ON (n.node_code = e.node_from_code)
WHERE e.edge_id IS NULL
UNION
SELECT n.node_id AS node_id
FROM "LocalCouncils_nodes" AS n
LEFT JOIN "LocalCouncils_edges" AS e ON (n.node_code = e.node_to_code)
WHERE e.edge_id IS NULL
```

*prefix*_dangling_nodes

this View will identify any dangling Node (i.e. insulated Nodes not connected to any Edge).



```
LocalCouncils_curves
LocalCouncils_curves_resolved
LocalCouncils_edges
    edge_id
    node_from_code
    node_to_code
    edge_code
    Geometry
    PrimaryKey
    idx_"LocalCouncils_edges"_to
    idx_"LocalCouncils_edges"_from
    idx_"LocalCouncils_edges"_code
    ggi_LocalCouncils_edges_Geometry
    ggu_LocalCouncils_edges_Geometry
    gid_LocalCouncils_edges_Geometry
    gii_LocalCouncils_edges_Geometry
    giu_LocalCouncils_edges_Geometry
LocalCouncils_edges_check_dupl_codes
LocalCouncils_edges_check_intersections
LocalCouncils_faces
```

```
SELECT ROWID, "edge_id", "node_from_code", "no
FROM "LocalCouncils_edges"
ORDER BY ROWID
```

	id	node_from_code	node_to_code	edge_code	Geon
1	1	NODE_12806	NODE_01871	EDGE_00001	BLOB sz=1424
2	2	NODE_00398	NODE_08307	EDGE_00002	BLOB sz=320
3	3	NODE_14409	NODE_00222	EDGE_00003	BLOB sz=1648
4	4	NODE_08307	NODE_14409	EDGE_00004	BLOB sz=912
5	5	NODE_14235	NODE_12806	EDGE_00005	BLOB sz=2912
6	6	NODE_01871	NODE_05037	EDGE_00006	BLOB sz=1328
7	7	NODE_05037	NODE_00398	EDGE_00007	BLOB sz=1808
8	8	NODE_06592	NODE_14235	EDGE_00008	BLOB sz=1328

current block: 1 / 500 [25975 rows]   [fetched in 00:00:00

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

*prefix*_edges

this Table contain any Edge used by the current Topology:

- an Edge is *real*, so a corresponding Geometry (of the LINESTRING type) must always be defined for each Edge.
- the ***edge_id*** simply is a Primary Key uniquely identifying the Edge.
- the ***edge_code*** is a symbolic name identifying the Edge: it's equivalent to the **gml:id** field required by GML 3 Topology.
- the ***node_from_code*** and ***node_to_code*** relationally identify the Nodes respectively corresponding to the first and last Point of the Edge.

related Views (supporting validity checks):

```
CREATE VIEW "LocalCouncils_edges_check_dupl_codes" AS
SELECT edge_code AS edge_code, Count(edge_id) AS count
FROM "LocalCouncils_edges"
GROUP BY edge_code
HAVING count > 1
```

*prefix*_edges_check_dupl_codes

this View will identify any duplicate ***edge_code*** value.

```
CREATE VIEW "LocalCouncils_edges_check_intersections" AS
SELECT e1.edge_id AS edge1_id, e1.edge_code AS edge1_code,
  e2.edge_id AS edge2_id, e2.edge_code AS edge2_code
FROM "LocalCouncils_edges" AS e1
JOIN "LocalCouncils_edges" AS e2 ON (
  e1.edge_id <> e2.edge_id AND
NOT (e1.node_from_code = e2.node_from_code AND e1.node_to_code =
e2.node_to_code) AND
  ST_Crosses(e1.Geometry, e2.Geometry) = 1 AND
  e2.edge_id IN (
      SELECT ROWID FROM SpatialIndex
      WHERE f_table_name = 'LocalCouncils_edges' AND
        search_frame = e1.Geometry))
```

*prefix*_edges_check_intersections

this View will identify any couple of intersecting Edges.
Remember: two Edges can legitimately intersect only at their extreme points (i.e. where a Node is defined).

```
CREATE VIEW "LocalCouncils_edges_check_nodes" AS
SELECT e.edge_id AS edge_id, n.node_id AS node_id
FROM "LocalCouncils_edges" AS e,
"LocalCouncils_nodes" AS n
WHERE ST_Intersects(e.Geometry, n.Geometry)
  AND ST_Equals(ST_StartPoint(e.Geometry), n.Geometry) = 0
  AND ST_Equals(ST_EndPoint(e.Geometry), n.Geometry) = 0
  AND n.ROWID IN (
    SELECT ROWID FROM SpatialIndex
    WHERE f_table_name = 'LocalCouncils_nodes'
      AND search_frame = e.Geometry)
```

*prefix*_edges_check_nodes

this View will identify any Node overlapping some Edge (excluding the Edge's extreme Points, where a Node is surely expected to exist).

```
CREATE VIEW "LocalCouncils_dangling_edges" AS
SELECT e.edge_id AS edge_id
FROM "LocalCouncils_edges" AS e
LEFT JOIN "LocalCouncils_faces_edges" AS f ON (e.edge_code = f.edge_code)
WHERE f.edge_code IS NULL
INTERSECT
SELECT e.edge_id AS edge_id
FROM "LocalCouncils_edges" AS e
LEFT JOIN "LocalCouncils_curves" AS c ON (e.edge_code = c.edge_code)
WHERE c.edge_code IS NULL
```

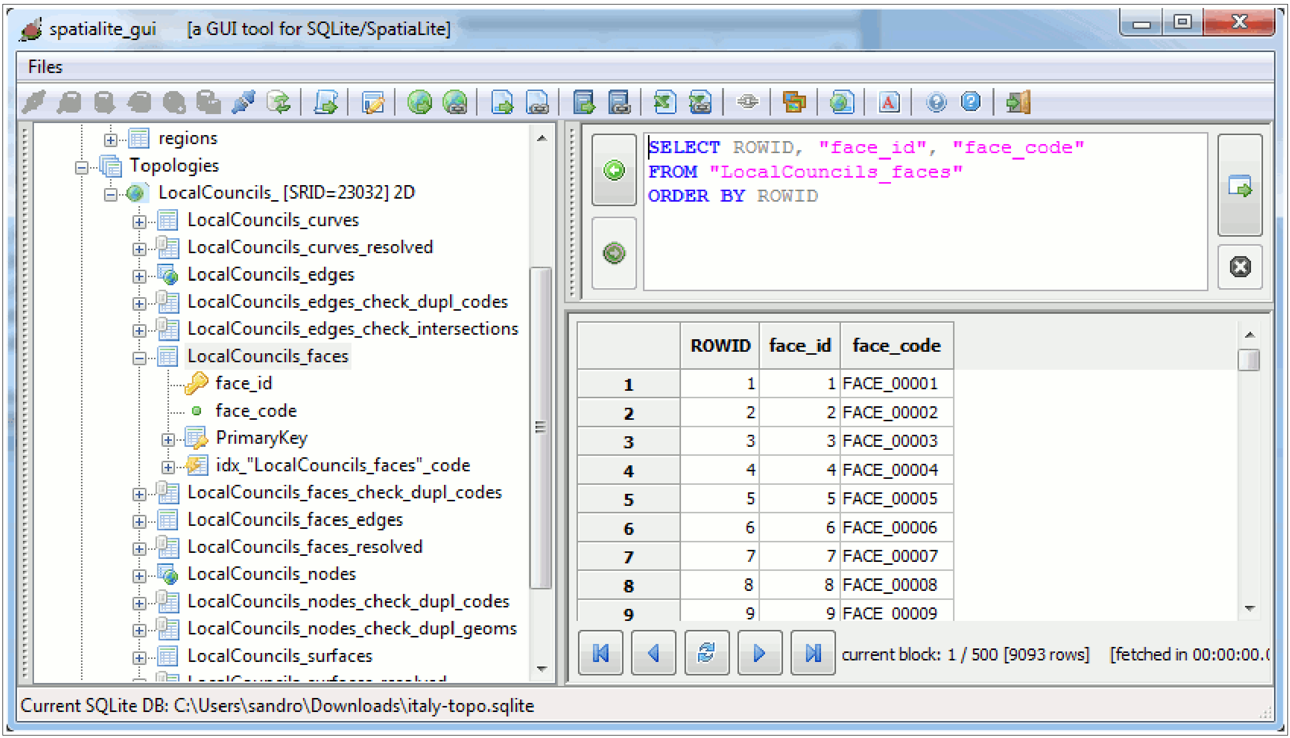
*prefix*_dangling_edges

this View will identify any dangling Edge (i.e. insulated Edges not used by any Face or TopoCurve).

```
CREATE VIEW "LocalCouncils_edges_check_from_to" AS
SELECT e.edge_id AS edge_id, n.node_id AS node_id,
  n.node_code AS node_code,
  'Mismatching coords' AS error_cause
FROM "LocalCouncils_edges" AS e
JOIN "LocalCouncils_nodes" AS n ON (e.node_from_code = n.node_code)
WHERE ST_Equals(ST_StartPoint(e.Geometry), n.Geometry) = 0
UNION
SELECT e.edge_id AS edge_id, n.node_id AS node_id,
  n.node_code AS node_code,
  'Mismatching coords' AS error_cause
FROM "LocalCouncils_edges" AS e
JOIN "LocalCouncils_nodes" AS n ON (e.node_to_code = n.node_code)
WHERE ST_Equals(ST_EndPoint(e.Geometry), n.Geometry) = 0
UNION
SELECT e.edge_id AS edge_id, n.node_id AS node_id,
  n.node_code AS node_code,
  'Unresolved Node reference' AS error_cause
FROM "LocalCouncils_edges" AS e
LEFT JOIN "LocalCouncils_nodes" AS n ON (e.node_from_code = n.node_code)
WHERE n.node_id IS NULL
UNION
SELECT e.edge_id AS edge_id, n.node_id AS node_id,
  n.node_code AS node_code,
  'Unresolved Node reference' AS error_cause
FROM "LocalCouncils_edges" AS e
LEFT JOIN "LocalCouncils_nodes" AS n ON (e.node_to_code = n.node_code)
WHERE n.node_id IS NULL
```

*prefix*_edges_check_from_to

this View will identify any mismatching Node-Edge relationship: i.e. will identify any Edge whose extreme coordinates does not exactly match the ones declared by the corresponding Node.


spatialite_gui [a GUI tool for SQLite/SpatiaLite]

Files

- regions
- Topologies
  - LocalCouncils_ [SRID=23032] 2D
    - LocalCouncils_curves
    - LocalCouncils_curves_resolved
    - LocalCouncils_edges
    - LocalCouncils_edges_check_dupl_codes
    - LocalCouncils_edges_check_intersections
    - LocalCouncils_faces
      - face_id
      - face_code
      - PrimaryKey
      - idx_"LocalCouncils_faces"_code
    - LocalCouncils_faces_check_dupl_codes
    - LocalCouncils_faces_edges
    - LocalCouncils_faces_resolved
    - LocalCouncils_nodes
    - LocalCouncils_nodes_check_dupl_codes
    - LocalCouncils_nodes_check_dupl_geoms
    - LocalCouncils_surfaces

```
SELECT ROWID, "face_id", "face_code"
FROM "LocalCouncils_faces"
ORDER BY ROWID
```

	ROWID	face_id	face_code
1	1	1	FACE_00001
2	2	2	FACE_00002
3	3	3	FACE_00003
4	4	4	FACE_00004
5	5	5	FACE_00005
6	6	6	FACE_00006
7	7	7	FACE_00007
8	8	8	FACE_00008
9	9	9	FACE_00009

current block: 1 / 500 [9093 rows] [fetched in 00:00:00.

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

*prefix*_faces

this Table contain any Face used by the current Topology:

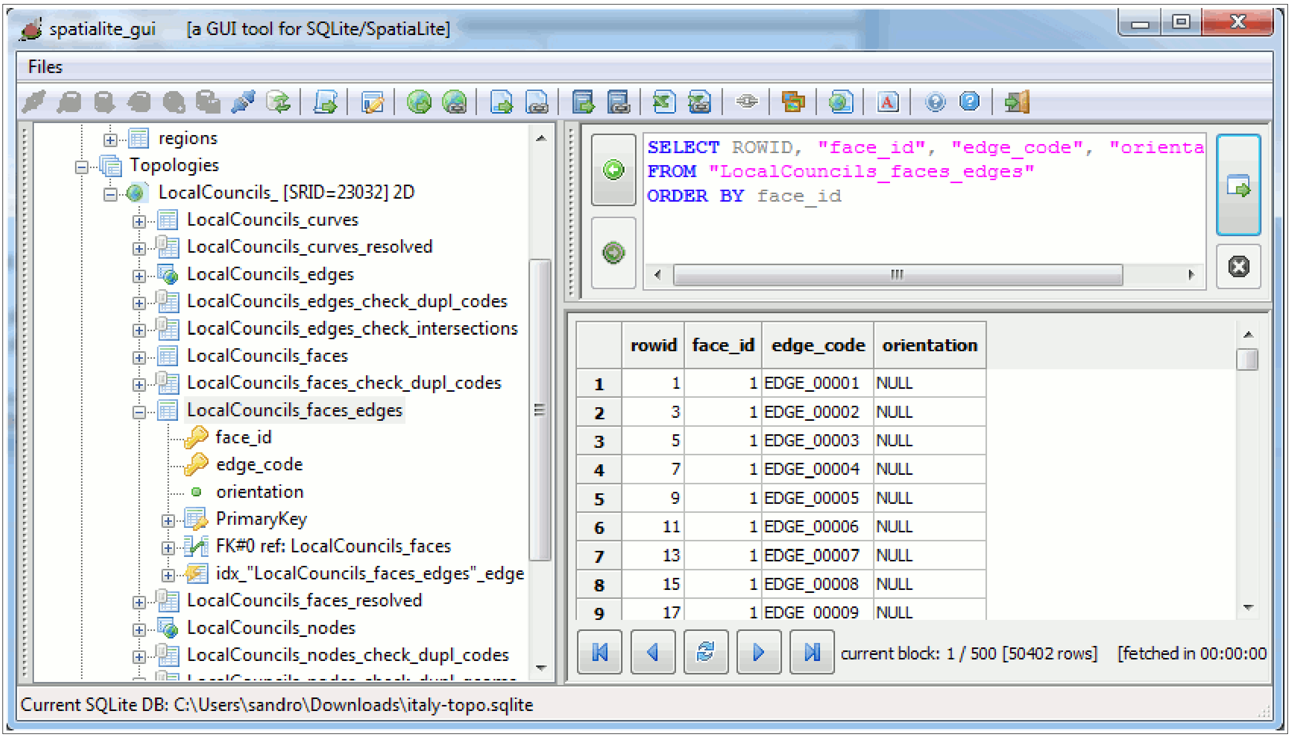
- a Face is <u>*virtual*</u>, so no corresponding explicit Geometry can be declared.
- the ***face_id*** simply is a Primary Key uniquely identifying the Face.
- the ***face_code*** is a symbolic name identifying the Edge: it's equivalent to the **gml:id** field required by GML 3 Topology.

<u>related Views (supporting validity checks):</u>

```
CREATE VIEW "LocalCouncils_faces_check_dupl_codes" AS
SELECT face_code AS face_code, Count(face_id) AS count
FROM "LocalCouncils_faces"
GROUP BY face_code
HAVING count > 1
```

*prefix*_faces_check_dupl_codes

this View will identify any duplicate ***face_code*** value.


spatialite_gui [a GUI tool for SQLite/SpatiaLite]
Files
regions
Topologies
LocalCouncils_ [SRID=23032] 2D
LocalCouncils_curves
LocalCouncils_curves_resolved
LocalCouncils_edges
LocalCouncils_edges_check_dupl_codes
LocalCouncils_edges_check_intersections
LocalCouncils_faces
LocalCouncils_faces_check_dupl_codes
LocalCouncils_faces_edges
face_id
edge_code
orientation
PrimaryKey
FK#0 ref: LocalCouncils_faces
idx_"LocalCouncils_faces_edges"_edge
LocalCouncils_faces_resolved
LocalCouncils_nodes
LocalCouncils_nodes_check_dupl_codes

```
SELECT ROWID, "face_id", "edge_code", "orienta
FROM "LocalCouncils_faces_edges"
ORDER BY face_id
```

	rowid	face_id	edge_code	orientation
1	1	1	EDGE_00001	NULL
2	3	1	EDGE_00002	NULL
3	5	1	EDGE_00003	NULL
4	7	1	EDGE_00004	NULL
5	9	1	EDGE_00005	NULL
6	11	1	EDGE_00006	NULL
7	13	1	EDGE_00007	NULL
8	15	1	EDGE_00008	NULL
9	17	1	EDGE_00009	NULL

current block: 1 / 500 [50402 rows] [fetched in 00:00:00

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

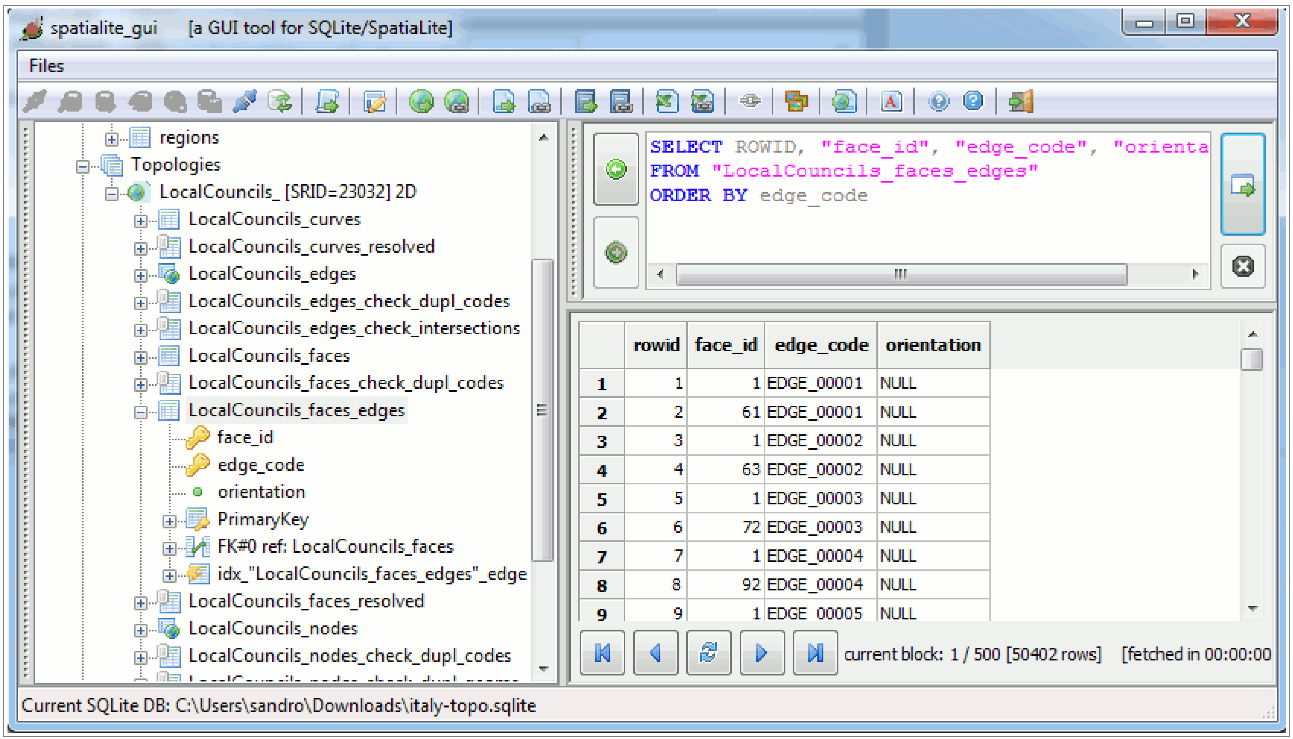
*prefix*_faces_edges

this Table contain the list of Edges delimiting each Face:

- ***face_id*** establishes a relational reference to the corresponding Face.
- ***edge_code*** establishes a second relational reference to the corresponding Edge

orientation is equivalent to the **gml:directedEdge orientation** field required by GML 3 Topology, and is never used by SpatiaLite itself.

Remember: A single Face may require more than a single Edge to be delimited.


spatialite_gui [a GUI tool for SQLite/SpatiaLite]
Files
regions
Topologies
LocalCouncils_ [SRID=23032] 2D
LocalCouncils_curves
LocalCouncils_curves_resolved
LocalCouncils_edges
LocalCouncils_edges_check_dupl_codes
LocalCouncils_edges_check_intersections
LocalCouncils_faces
LocalCouncils_faces_check_dupl_codes
LocalCouncils_faces_edges
face_id
edge_code
orientation
PrimaryKey
FK#0 ref: LocalCouncils_faces
idx_"LocalCouncils_faces_edges"_edge
LocalCouncils_faces_resolved
LocalCouncils_nodes
LocalCouncils_nodes_check_dupl_codes

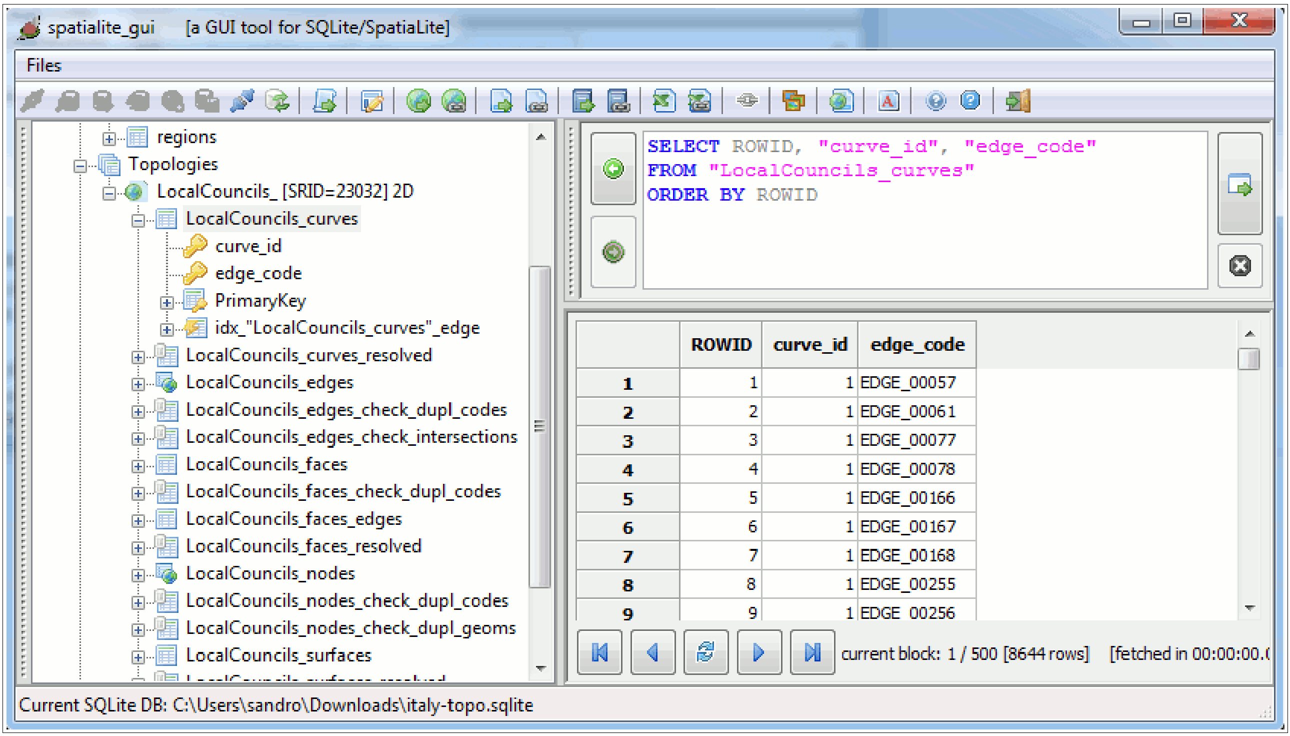
```
SELECT ROWID, "face_id", "edge_code", "orienta
FROM "LocalCouncils_faces_edges"
ORDER BY edge_code
```

	rowid	face_id	edge_code	orientation
1	1	1	EDGE_00001	NULL
2	2	61	EDGE_00001	NULL
3	3	1	EDGE_00002	NULL
4	4	63	EDGE_00002	NULL
5	5	1	EDGE_00003	NULL
6	6	72	EDGE_00003	NULL
7	7	1	EDGE_00004	NULL
8	8	92	EDGE_00004	NULL
9	9	1	EDGE_00005	NULL

current block: 1 / 500 [50402 rows] [fetched in 00:00:00

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

Please note: the same Edge is (usually) referenced by two different Faces.


spatialite_gui [a GUI tool for SQLite/SpatiaLite]

- regions
- Topologies
  - LocalCouncils_ [SRID=23032] 2D
    - LocalCouncils_curves
      - curve_id
      - edge_code
      - PrimaryKey
      - idx_"LocalCouncils_curves"_edge
    - LocalCouncils_curves_resolved
    - LocalCouncils_edges
    - LocalCouncils_edges_check_dupl_codes
    - LocalCouncils_edges_check_intersections
    - LocalCouncils_faces
    - LocalCouncils_faces_check_dupl_codes
    - LocalCouncils_faces_edges
    - LocalCouncils_faces_resolved
    - LocalCouncils_nodes
    - LocalCouncils_nodes_check_dupl_codes
    - LocalCouncils_nodes_check_dupl_geoms
    - LocalCouncils_surfaces

```
SELECT ROWID, "curve_id", "edge_code"
FROM "LocalCouncils_curves"
ORDER BY ROWID
```

	ROWID	curve_id	edge_code
1	1	1	EDGE_00057
2	2	1	EDGE_00061
3	3	1	EDGE_00077
4	4	1	EDGE_00078
5	5	1	EDGE_00166
6	6	1	EDGE_00167
7	7	1	EDGE_00168
8	8	1	EDGE_00255
9	9	1	EDGE_00256

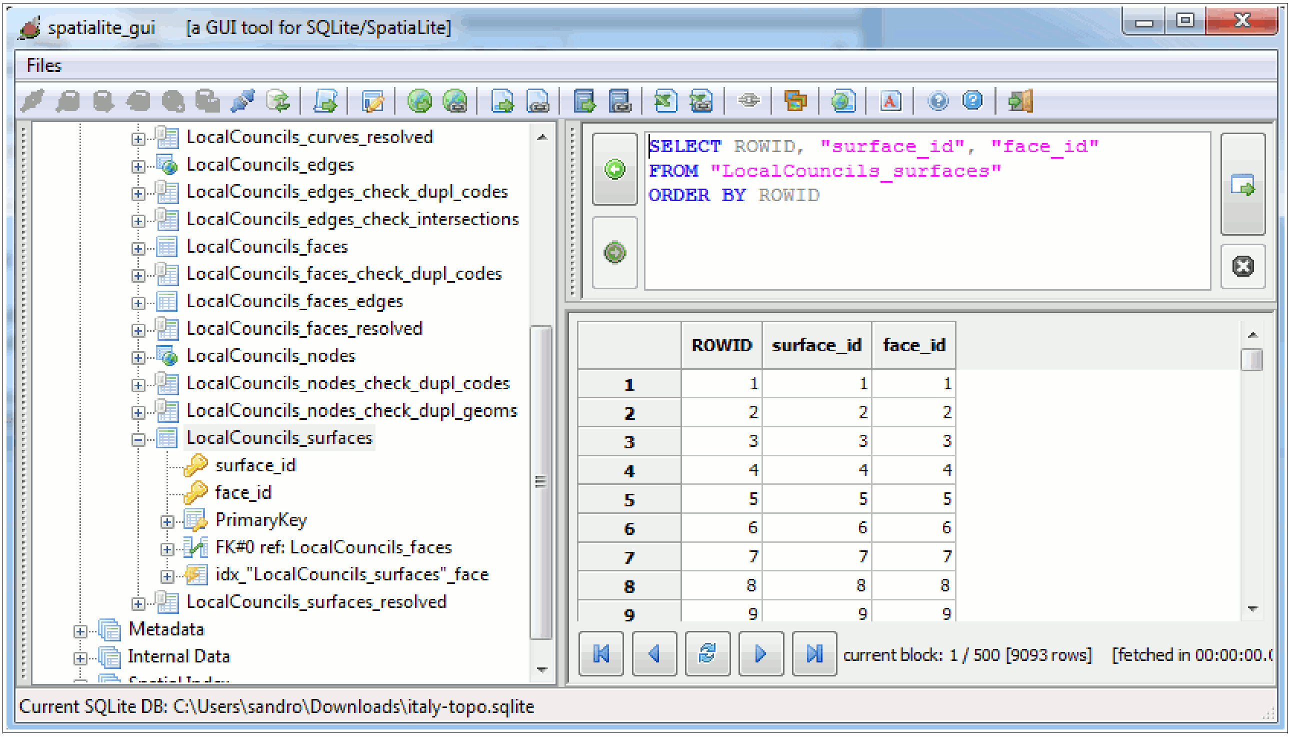
current block: 1 / 500 [8644 rows] [fetched in 00:00:00.

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

*prefix*_curves

this Table contain the list of Edges composing each TopoCurve:

- a TopoCurve is *virtual*, so no corresponding explicit Geometry can be declared.
- ***curve_id*** uniquely identifies each TopoCurve.
- ***edge_code*** establishes a relational reference to the corresponding Edge.


spatialite_gui [a GUI tool for SQLite/SpatiaLite]

- LocalCouncils_curves_resolved
- LocalCouncils_edges
- LocalCouncils_edges_check_dupl_codes
- LocalCouncils_edges_check_intersections
- LocalCouncils_faces
- LocalCouncils_faces_check_dupl_codes
- LocalCouncils_faces_edges
- LocalCouncils_faces_resolved
- LocalCouncils_nodes
- LocalCouncils_nodes_check_dupl_codes
- LocalCouncils_nodes_check_dupl_geoms
- LocalCouncils_surfaces
  - surface_id
  - face_id
  - PrimaryKey
  - FK#0 ref: LocalCouncils_faces
  - idx_"LocalCouncils_surfaces"_face
- LocalCouncils_surfaces_resolved
- Metadata
- Internal Data

```
SELECT ROWID, "surface_id", "face_id"
FROM "LocalCouncils_surfaces"
ORDER BY ROWID
```

	ROWID	surface_id	face_id
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7
8	8	8	8
9	9	9	9

current block: 1 / 500 [9093 rows] [fetched in 00:00:00.

Current SQLite DB: C:\Users\sandro\Downloads\italy-topo.sqlite

*prefix*_surfaces

this Table contain the list of Faces composing each TopoSurface:

- a TopoSurface is *virtual*, so no corresponding explicit Geometry can be declared.
- ***surface_id*** uniquely identifies each TopoSurface.
- ***face_code*** establishes a relational reference to the corresponding Face.

Views supporting materialization of *virtual* Geometries

```
CREATE VIEW "LocalCouncils_faces_resolved" AS
SELECT f.face_id AS face_id, ST_Polygonize(e.Geometry) AS Geometry
FROM "LocalCouncils_faces" AS f
LEFT JOIN "LocalCouncils_faces_edges" AS fe ON (fe.face_id = f.face_id)
LEFT JOIN "LocalCouncils_edges" AS e ON (e.edge_code = fe.edge_code)
GROUP BY f.face_id
```

*prefix*_faces_resolved

this View is intended to build the canonical Geometry (of the POLYGON type) representing each Face:

- all Edges required to delimit any Face are retrieved using relational JOINs.
- the GROUP BY clause ensures that a distinct Edges packet will be aggregated for each single Face.
- ST_Polygonize() [*aggregate function*] is then invoked in order to reassemble a Polygon representing the Face from this collection of sparse Edges.
- Please note well: due to ST_Polygonize() implementation any arbitrary collection of Edges (if properly and correctly noded) can be reassembled into a valid Polygon. Thus making relative ordering of Edges absolutely not relevat. Even inner/outer rings can be freely inter-mixed with no special constraints.

```
CREATE VIEW "LocalCouncils_surfaces_resolved" AS
SELECT s.surface_id AS surface_id,
  CastToMultipolygon(ST_Collect(f.Geometry)) AS Geometry
FROM "LocalCouncils_surfaces" AS s
LEFT JOIN "LocalCouncils_faces_resolved" AS f ON (f.face_id = s.face_id)
GROUP BY s.surface_id
```

*prefix*_surfaces_resolved

this View is intended to build the canonical Geometry (of the MULTIPOLYGON type) representing each TopoSurface:

- all Faces required to build any TopoSurface are retrieved using a relational JOIN.
- the GROUP BY clause ensures that a distinct Faces packet will be aggregated for each single TopoSurface.
- ST_Collect() [*aggregate function*] is then invoked in order to reassemble any elementary POLYGON into a MULTIPOLYGON.
- CastToMultipolygon() is finally invoked in order to grant that even TopoSurfaces containing a single Face/POLYGON will then be correctly represented as MULTIPOLYGONs.
- Please note: this one is a *second order* View (doubly indirected), because it depends on the previous View resolving individual Faces.

```
CREATE VIEW "LocalCouncils_curves_resolved" AS
SELECT c.curve_id AS curve_id,
  CastToMultiLinestring(ST_Collect(e.Geometry)) AS Geometry
FROM "LocalCouncils_curves" AS c
LEFT JOIN "LocalCouncils_edges" AS e ON (e.edge_code = c.edge_code)
GROUP BY c.curve_id
```

*prefix*_curves_resolved

this View is intended to build the canonical Geometry (of the MULTILINESTRING type) representing each TopoCurve:

- all Edges required to build any TopoCurve are retrieved using a relational JOIN.
- the GROUP BY clause ensures that a distinct Edges packet will be aggregated for each single TopoCurve.
- ST_Collect() [*aggregate function*] is then invoked in order to reassemble any elementary LINESTRING into a MULTILINESTRING .
- CastToMultiLinestring() is finally invoked in order to grant that even TopoCurves containing a single Edge/LINESTRING will then be correctly represented as MULTILINESTRINGs.

Final recapitulation

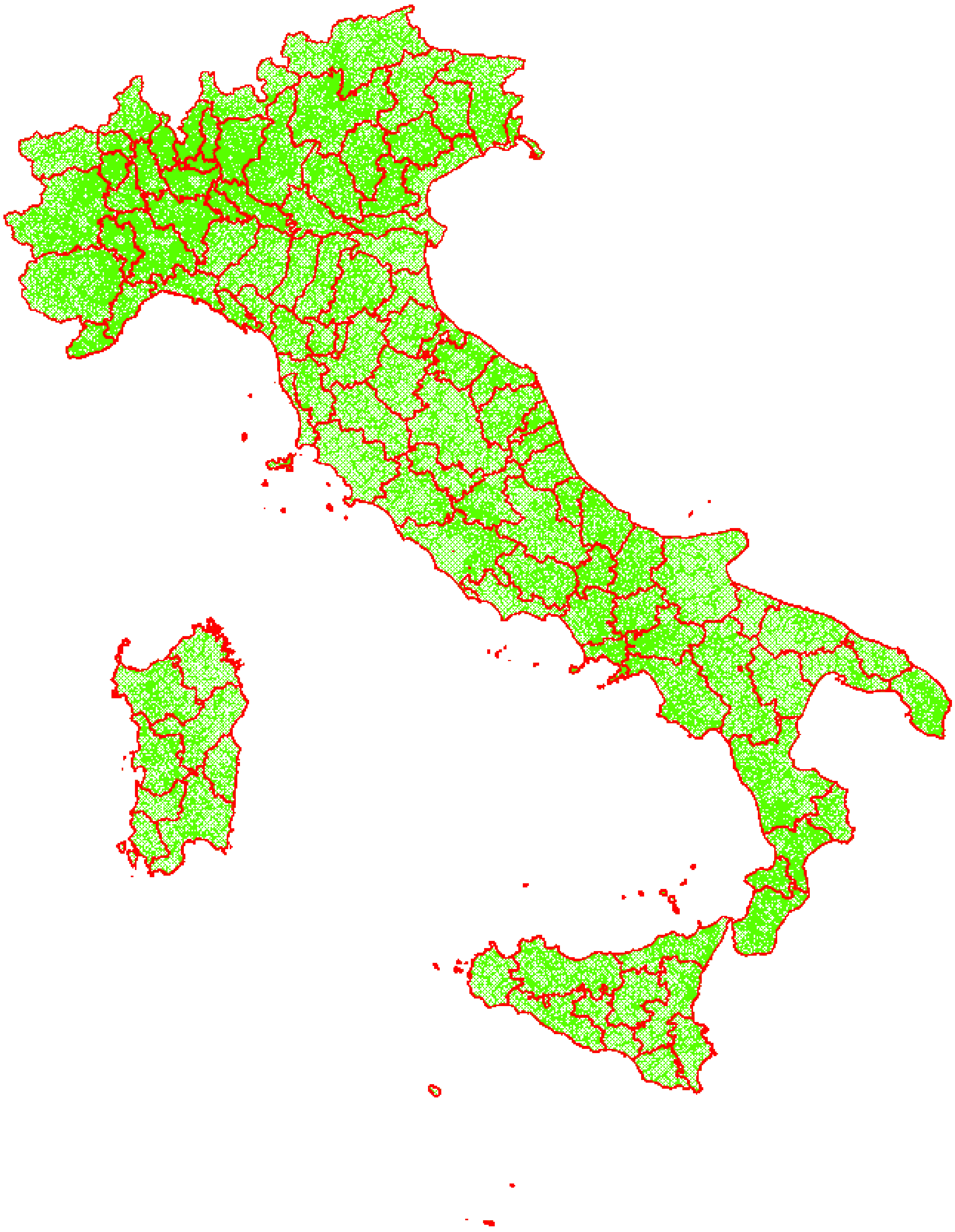
- the sample DB contains a full Topology Set representing Italian Local Councils.
- only Nodes and Edges are represented by *real* Geometries.
- Faces, TopoCurves and TopoSurfaces are represented by their corresponding topologies: i.e. all these items are *virtual* and have no explict Geometry directly set.
- anyway, a *real* Geometry corresponding to *virtual* items can be materialized on demand invonking the appropriate View.
- many useful Views are pre-defined, thus allowing to easily check if the Topology really is an a fully consistent state.
- the layout adopted for Topology Tables can easily support GML 3 Topology.

… oh, just a last curiosity to be shortly explained.

Usually TopoCurves correspond to some real cartographic feature (rivers, roads, mountain ridges, coastlines …): in this sample I wasn't able to identify any obvious correspondence.

So I've simply decided to arbitrarily overload TopoCurves in such a way to represent County borderlines; this was really simple to implement, because I simply had to select all the relevant Edges, then defining the Edge collection corresponding to each County.

Please note: following this approach County borderlines are intrinsically topological, because they correspond to TopoCurves.



You can easily attempt by yourself to export the Shapefile representing TopoCurves, then using any GIS viewer so to visualize both Local Councils (MultiPolygons) and County borderlines.