dbcxml2splite

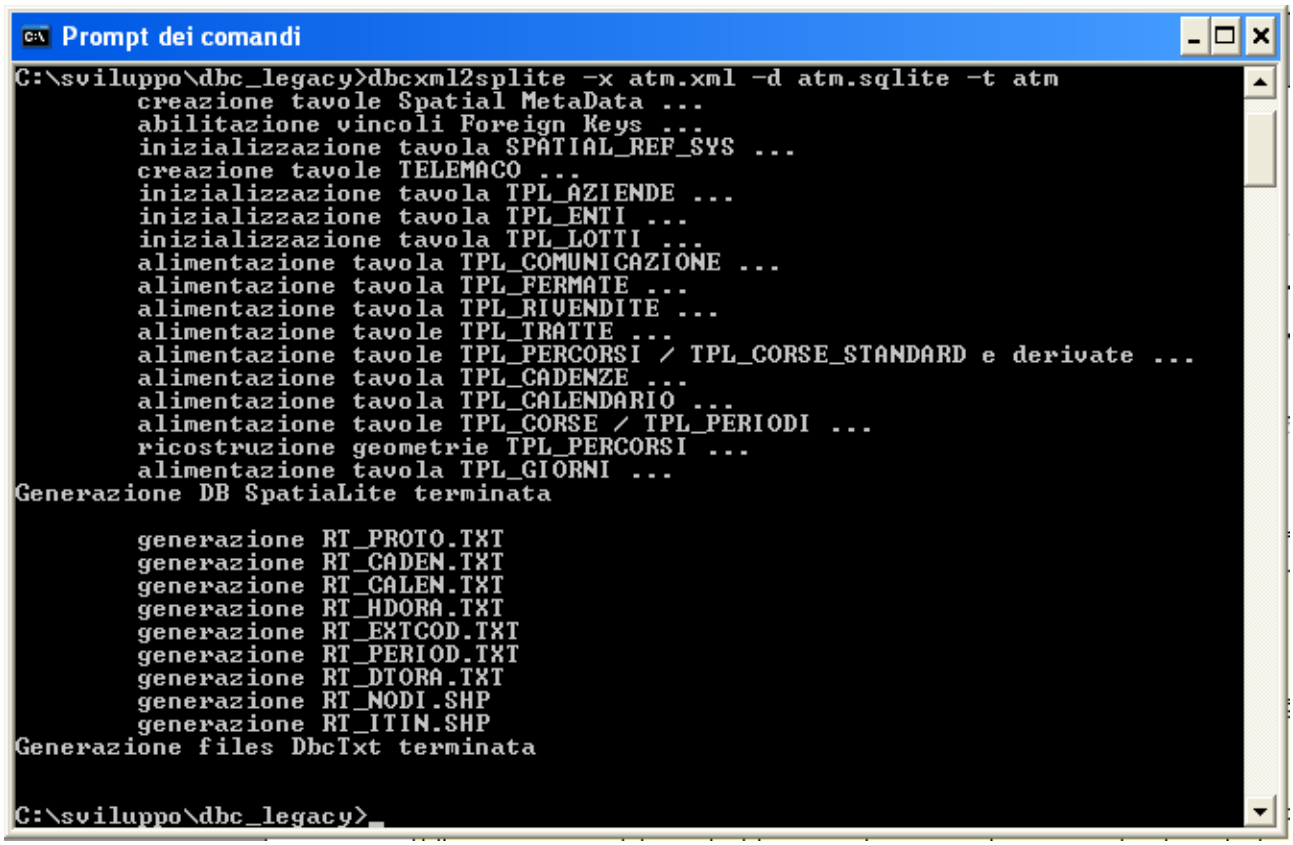
un *tool command line* che consente di elaborare una comunicazione nel formato DbcXML

---

dbcxml2splite è un semplice tool di tipo *command line*
Quindi non presenta nessuna interfaccia di tipo grafico: pertanto occorre lanciarlo direttamente dall'**ambiente di shell**
[probabilmente gli utenti Windows sono abituati a chiamarla finestra MS-DOS, prompt dei comandi ed altre amenità del genere ...]

dbcxml2splite è rilasciato sotto GNU Public License [GPL], quindi è *open source*.
Chiunque può utilizzarlo liberamente senza alcun vincolo di sorta [anzi, siete caldamente invitati ad utilizzarlo ...]

l'eseguibile di **dbcxml2splite** è di tipo monolitico [*statically linked*], e quindi non necessita di nessun processo di installazione: è sufficiente copiare l'eseguibile in una posizione qualsiasi del file system per poterlo immediatamente utilizzare senza altre complicazioni di sorta.



Argomenti di invocazione:

`-x file-path` oppure **`--xml-path file-path`**

argomento **obbligatorio**
deve indicare il *path* corrispondente al file XML che di desidera elaborare [**input**]
Questo file deve esistere, e l'utente deve godere di abilitazione alla lettura.

`-d file-path` oppure **`--db-path file-path`**

argomento **obbligatorio**
deve indicare il *path* corrispondente al DB SQLite/SpatiaLite che di desidera generare [**output**]
E' preferibile che il DB indicato non esista affatto: in questo caso verrà automaticamente creato.
Qualora si utilizzi un DB già esistente le operazione di alimentazione del DB potrebbero fallire a causa di incompatibilità con gli elementi già presenti all'interno del DB.

`-t dir-path` oppure **`--dir-txt-path dir-path`**

argomento **facoltativo**
Qualora presente deve indicare il *path* corrispondente ad una directory all'interno della quale verrano generati i files nel vecchio formato DbcTxt [**output**]
E' preferibile che la directory indicata non esista affatto: in questo caso verrà automaticamente creata.
Qualora si utilizzi una directory già esistente i files in essa eventualmente presenti verranno sovrascritti se del caso.
Nel caso in cui l'argomento non venga dichiarato verrà semplicemente omessa la generazione dei files nel formato DbcTxt, mentre il DB SQLite/SpatiaLite verrà generato in tutti i casi.

Esempi:

`dbcxml2splite -x arezzo.xml -d arezzo.sqlite -t arezzo_txt`

- acquisisce il file **arezzo.xml**
- e quindi genera un DB corrispondente di nome **arezzo.sqlite**
- al termine del processo verrà creata la directory **arezzo_txt**
- al cui interno verranno generati tutti i files che compongono la comunicazione DbcTxt corrispondente

`dbcxml2splite -x siena.xml -d siena.sqlite`

- acquisisce il file **siena.xml**
- e quindi genera un DB corrispondente di nome **siena.sqlite**
- i files della comunicazione DbcTxt non verranno generati

Avvertenze:

Il file XML **deve** essere già stato validato precedentemente [da Telemaco oppure da Ermes].

dbcxml2splite utilizza una tecnica nota come *non-validating parsing*, che è estremamente veloce e di semplice elaborazione, ma che non fornisce alcun tipo di supporto diagnostico.
Eventulmente il processo si bloccherà al primo errore fatale riscontrato nel codice XML.

Supporto GIS

Ciascun DB SQLite/SpatiaLite generato da **dbcxml2splite** può essere immediatamente utilizzato tramite uno dei seguenti applicativi GIS:

- **spatialite-gis**
- **QGis**

Le seguenti tavole del DB possono essere utilizzate come layer GIS:

Tavola	Contenuto	Note
tpl_fermate	POINT EPSG 3003 fermate TPL	Obbligatorio - Presente in ogni caso
tpl_rivendite	POINT EPSG 3003 rivendite, biglietterie	Facoltativo - Presente solo se l'Azienda TPL ha fornito l'informazione relativa
tpl_tratte	LINESTRING EPSG 3003 tratte fermata-fermata	Obbligatorio per le comunicazioni Level.2 Assente nelle comunicazioni Level.1
tpl_percorsi	LINESTRING EPSG 3003 percorsi TPL [da capolinea a capolinea]	Obbligatorio – Presente in ogni caso

Tavole del DB

Avvertenza: tutte le tavole interessanti sono identificate dal prefisso **tpl_**
potete tranquillamente ignorare qualsiasi altra tavola diversa da queste, dato che si tratta di strutture interne di natura tecnica tanto indispensabili per il funzionamento del DB quanto assolutamente prive di contenuti interessanti per un operatore umano.

Tavola	Contenuto
tpl_comunicazione	Singola riga estremi identificativi della comunicazione
tpl_aziende	nomenclatore OTRT aziende e SCARL
tpl_enti	nomenclatore OTRT Enti
tpl_lotti	nomenclatore OTRT Lotti
tpl_cadenze	cadenza aziendali
tpl_calendario	calendario di servizio
tpl_fermate	punti di fermata GIS
tpl_tratte	percorsi che uniscono due punti di fermata consecutivi Solo per le comunicazioni Level.2 GIS
tpl_percorsi_tratte	articolazione analitica dei percorsi in tratte Solo per le comunicazioni Level.2
tpl_percorsi	percorsi TPL GIS
tpl_corse_standard	intestazioni corse standard
tpl_corse_standard_orari	dettagli passaggi orari corse standard
tpl_linee_standard	linee ufficiali
tpl_linee_aziendali	linee aziendali
tpl_corse	corse TPL
tpl_periodi	periodi effettuazione corsa [cadenzamento]
tpl_giorni	giorni effettuazione corsa

I vincoli relazionali tra le diverse tavole sono facilmente identificabili, dato che sono state esplicitamente dichiarate in ogni caso le relative **PRIMARY KEY / FOREIGN KEY**

Le VIEWs pre-definite

Avvertenza: dato che la struttura relazionale del DB è fortemente normalizzata, è necessario effettuare delle JOIN abbastanza complesse per portare a termine praticamente qualsiasi tipo do interrogazione routinaria.
Fortunatamente SQLite supporta in modo assai efficiente le VIEWs.
Quindi all'interno del DB trovate già presenti alcune VIEWs che vi renderanno sicuramente assai più semplice il compito di elaborare le informazioni.
Potete agevolmente riconoscere qualsiasi di queste VIEW in quanto presentano un prefisso **tpl_** ed un suffisso **_view**

Utilizzate una VIEW tutte le volte che potete.
Se non siete veramente dei maghi di SQL, usando le VIEW potete ridurre in modo assai drastico la difficoltà di elaborare le informazioni di cui avete bisogno.
E anche se in effetti siete dei maghi di SQL, studiare il codice delle VIEWs predefinite (e magari fare *copia&incola*) vi semplificherà sicuramente la vita.

VIEW	**Contenuto**
tpl_orari_view	orari corsa *in forma graziosamente de-normalizzata*
tpl_corse_view	informazioni generali corsa *in forma graziosamente de-normalizzata*
tpl_contratti_view	informazioni contrattuali corsa *in forma graziosamente de-normalizzata*

Esempi SQL

Tutti gli esempi che seguono sono estesamente basati sull'uso delle VIEWs predefinite.
Verranno affrontati i casi d'uso più facilmente ricorrenti.

Esistono svariati tools, connettori etc per eseguire una query SQL su un DB SQLite / SpatiaLite

La scelta di riferimento consigliata consiste comunque nell'app **spatialite-gui** che presenta una simpatica interfaccia grafica di tipo facilmente intuitivo

Il modo migliore per familiazzare con lo strumento è quello di provare concretamente ...
un buon punto di partenza può proprio essere quello di crearsi un DB di prova e quindi giocare per una decina di minuti via *copia&incolla*

Recuperare l'orario completo per una determinata corsa**:**

```
SELECT *
FROM tpl_orari_view
WHERE id_corsa = 3
ORDER BY sub
```

Come la precedente, ma includendo solo le fermate effettive:

```
SELECT *
FROM tpl_orari_view
WHERE id_corsa = 3 AND non_effettua_fermata = 0
ORDER BY sub
```

Recuperare la lista completa dei transiti per una determinata fermata in un giorno preciso con indicazione della destinazione finale
[ordinata per llinea, e per orario di passaggio all'interno di ciascuna linea]:

```
SELECT o.partenza AS Orario, c.codice_corsa_aziendale AS Corsa,
    c.linea_standard AS Linea, c.capolinea_arrivo AS Destinazione,
    c.arrivo AS "Arrivo"
FROM tpl_orari_view AS o, tpl_giorni AS g, tpl_corse_view AS c
WHERE o.codice_fermata = '0003' AND o.non_effettua_fermata = 0
    AND g.data = '2009-12-12' AND g.id_corsa = o.id_corsa
    AND c.id_corsa = o.id_corsa
ORDER BY c.linea_standard, o.arrivo
```

Recuperare l'elenco delle corse effettuate da un determinato subconcessionario [ordinate per linea e per orario di partenza all'interno di ciascuna linea]:

```
SELECT *
FROM tpl_corse_view
WHERE id_azienda_subappalto = 12
ORDER BY codice_linea_aziendale, partenza
```

Come la precedente, ma in un giorno ben determinato

```
SELECT *
FROM tpl_corse_view AS c, tpl_giorni AS g
WHERE id_azienda_subappalto = 12 AND g.data = '2009-12-08'
    AND g.id_corsa = c.id_corsa
ORDER BY c.codice_linea_aziendale, c.partenza
```

Recuperare l'elenco delle corse effettuate in un determinato giorno all'interno di una precisa fascia oraria e con partenza da un determinato capolinea:

```
SELECT *
FROM tpl_corse_view AS c, tpl_giorni AS g
WHERE g.data = '2009-12-08' AND g.id_corsa = c.id_corsa
    AND c.partenza >= '12:30' AND c.partenza <= '14:30'
    AND c.codice_capolinea_partenza = '0001'
ORDER BY c.partenza
```

Calcolare la percorrenza totale di ciascuna linea in un determinato giorno:

```
SELECT codice_linea_aziendale AS Linea, Count(*) AS "n.ro corse",
    Sum(c.lunghezza_km) AS Km
FROM tpl_corse_view AS c, tpl_giorni AS g
WHERE g.data = '2009-12-08' AND g.id_corsa = c.id_corsa
GROUP BY c.codice_linea_aziendale
```

Calcolare le percorrenza totali giornaliere in un determinato periodo:

```
SELECT g.data AS Giorno, Count(*) AS "n.ro corse",
    Sum(lunghezza_km) AS Km
FROM tpl_corse_view AS c, tpl_giorni AS g
WHERE g.data >= '2009-12-20' AND g.data <= '2009-12-31'
    AND g.id_corsa = c.id_corsa
GROUP BY g.data
```

Recuperare il contratto di servizio completo per un determinato Ente:

```
SELECT *
FROM tpl_contratti_view
WHERE id_ente_finanziatore_corsa = 207
ORDER BY linea_standard
```

Come la precedente, ma calcolando i totali riepilogativi per ciascuna linea:

```
SELECT linea_standard AS Linea,
    Sum(giorni_effettuazione) AS "n.ro corse",
    Sum(km_periodo) AS Km
FROM tpl_contratti_view
WHERE id_ente_finanziatore_corsa = 207
GROUP BY linea_standard
```

Calcolare le competenze contrattuali totali relative a ciascun Ente:

```
SELECT ente_finanziatore_corsa AS Ente,
    Sum(giorni_effettuazione) AS "n.ro corse",
    Sum(km_periodo) AS Km
FROM tpl_contratti_view
GROUP BY ente_finanziatore_corsa
```

Calcolare le competenze contrattuali totali relative a ciascuna Azienda di gestione:

```
SELECT denominazione_azienda_gestione AS "Azienda Gestione",
    Sum(giorni_effettuazione) AS "n.ro corse",
    Sum(km_periodo) AS Km
FROM tpl_contratti_view
GROUP BY denominazione_azienda_gestione
```

Calcolare le competenze contrattuali totali relativa a ciascuna Azienda subappalto:

```
SELECT denominazione_azienda_subappalto AS "Azienda Subappalto",
    Sum(giorni_effettuazione) AS "n.ro corse",
    Sum(km_periodo) AS Km
FROM tpl_contratti_view
GROUP BY denominazione_azienda_subappalto
```

Infine, giusto per assaggio, una semplice query spaziale: identificare le rivendite/biglietterie che cadono entro un determinato raggio rispetto ad una specifica fermata [ordinate per distanza crescente]:

```
SELECT r.codice_rivendita, r.ragione_sociale, r.ubicazione,
    ST_Distance(f.geometry, r.geometry) AS "Distanza m"
FROM tpl_rivendite AS r, tpl_fermate AS f
WHERE f.codice_fermata = '0001'
    AND ST_Distance(f.geometry, r.geometry) < 400.0
ORDER BY ST_Distance(f.geometry, r.geometry)
```

A buon intenditor, poche parole ...

Se non vi siete lasciati intimidere dall'apparente complessità di SQL, forse a questo punto iniziate a capire quanto questo strumento può esservi di concreta utilità nel vostro lavoro quotidiano.

Magari, chissà, almeno un'infarinatura di SQL vi è rimasta attaccata fin dai tempi degli studi ... questa può essere l'occasione buona per approfondire l'argomento.

Se invece siete già esperti informatici, allora magari sarete rimasti favorevolmente colpiti da quanto possa essere facile (e produttivo) utilizzare un Personal DB leggero, semplicissimo, ma anche estremamente potente, robusto e ricco di funzionalità.