

Esercitazione su PostGIS

Master in Sistemi Informativi Geografici per il Monitoraggio
e la Gestione del Territorio

Claudio Rocchini – Istituto Geografico Militare

Introduzione

- ▶ *Postgres con PostGIS* è un database server professionale Open Source e Gratuito
- ▶ Il supporto spaziale è veramente completo e fa concorrenza a software commerciali ben più costosi (es. *Oracle*)
- ▶ In questa esercitazione faremo una breve introduzione al supporto spaziale di Postgres.
- ▶ Prerequisiti:
 - ▶ Cenni sulle basi di dati
 - ▶ SQL
 - ▶ Interfaccia di Postgres

Nota su questi lucidi

- ▶ Le query SQL da eseguire sono sempre scritte con il carattere **Courier**
- ▶ PowerPoint trasforma talvolta il primo apicetto di una frase in apicetto inverso. In SQL bisogna sempre usare l'apicetto dritto (l'apostrofo).
- ▶ Si ricorda che per eseguire una query bisogna:
 - ▶ Aprire pgAdminIII
 - ▶ Connettersi al server giusto
 - ▶ Selezionare un database
 - ▶ Aprire SQL editor
 - ▶ Digitare la query nella finestra di sinistra
 - ▶ Premere il pulsante Play (execute)

Supporto Spaziale di Postgres

Vediamo in questa sezione cosa effettivamente PostGIS aggiunge
a Postgres

Supporto Spaziale a Postgres

- ▶ PostGIS aggiunge il supporto geografico a Postgres
- ▶ Tale supporto si compone di varie parti e si basa principalmente su standard Open GIS Consortium (Oracle invece, che è più vecchio, ha dovuto sviluppare un suo standard).
- ▶ Principali componenti:
 - ▶ Un nuovo tipo di dato: *GEOMETRY*
 - ▶ Tabelle di supporto: *spatial_ref_sys* e *geometry_columns*
 - ▶ Funzioni SQL di supporto (circa 700)
 - ▶ Tool esterni (es. *pgsql2shp* e *shp2pgsql*) che vedremo durante l'importazione di dati
- ▶ **Nota: Postgres non ha di per sé un visualizzatore**

GEOMETRY

- ▶ I dati di una colonna di database sono associati ad un tipo (es. INTEGER, CHARACTER, BOOLEAN)
- ▶ Il supporto spaziale introduce un nuovo tipo di dato: GEOMETRY, questo tipo è un tipo di dato ad oggetti (complesso)
- ▶ Questo tipo di dato contiene la geometria di un singolo oggetto geografico e eventualmente il sistema di riferimento associato (codice SRID)
- ▶ Il tipo è multiforme:
 - ▶ può contenere dati a 2, 3 o 4 dimensioni (x,y,z e il campo M)
 - ▶ Vari tipi di geometrie: punti, linee, aree, curve, etc.

spatial_ref_sys: introduzione

- ▶ E' una tabella del sistema spaziale: memorizza l'elenco dei sistemi di riferimento supportati
- ▶ Si basa principalmente sullo standard EPSG (<http://www.epsg.org>)
- ▶ Per ogni sistema previsto è indicato:
 - ▶ il codice numerico
 - ▶ la sorgente (si solito EPSG)
 - ▶ la definizione testuale (equivale al testo contenuto nei file prg)
 - ▶ la definizione proj4
- ▶ Nota: proj4 è una libreria di trasformazione di coordinate utilizzata da PostGIS

spatial_ref_sys: contenuto

- ▶ Esempio lo srid 4326 corrisponde a coordinate geografiche WGS84 (GPS)
- ▶ Ecco la definizione testuale:

```
GEOGCS[
  "WGS 84",
  DATUM[
    "WGS_1984",
    SPHEROID[
      "WGS84",
      6378137,298.257223563,
      AUTHORITY["EPSG","7030"]
    ],
    AUTHORITY["EPSG","6326"]
  ],
  PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
  UNIT["degree",0.01745329251994328,AUTHORITY["EPSG","9122"]],
  AUTHORITY["EPSG","4326"]
]
```

- ▶ SRID 32632 e 32633 corrispondono a UTM WGS84 fusi 32 e 33 Nord.
- ▶ SRID 23032 e 23033 corrispondono a UTM ED50 fusi 32 e 33 Nord.

spatial_ref_sys: estensione

- ▶ Se un sistema di riferimento non è previsto può essere aggiunto alla tabella, definendo l'opportuna stringa proj4, ovviamente il sistema non sarà standard
- ▶ Ad esempio Fuso Italia, usato per i cataloghi web, (una media fra fuso 32 e fuso 33) può essere definito come:
 - ▶ SRID: 9000;
 - ▶ auth_name: "IGMI";
 - ▶ auth_srid: 90000;
 - ▶ proj=tmerc +lat_0=0 +lon_0=12 +k=0.9985 +x_0=7000000.0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs“
- ▶ Postgres è in grado di utilizzare correttamente i dati geografici con sistemi definiti dall'utente.

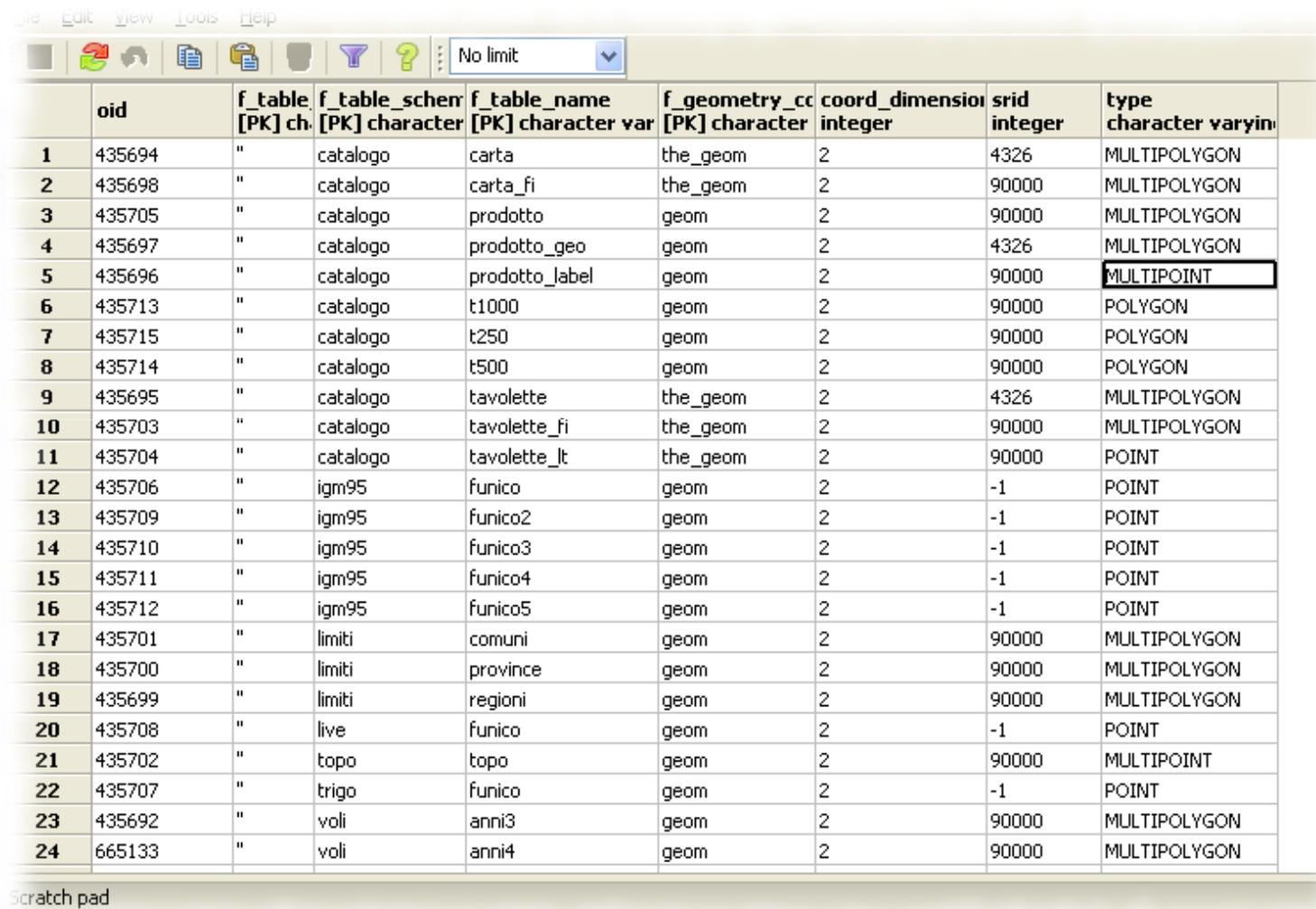
Un inciso: Roma40

- ▶ Alcuni sistemi di riferimento non possono essere trattati in modo analitico
- ▶ Ad esempio Roma40, ha bisogno (per una sua eventuale trasformazione), di griglie di trasformazione punto punto, che memorizzano la differenza di coordinate.
- ▶ L'IGM vende queste griglie di trasformazione a blocchi
- ▶ Proj4 (e quindi Postgres) supportano le griglie di trasformazione nel formato standard NAD (ntv2)
- ▶ Ad esempio la seguente definizione costruisce un sistema geografico Roma40 (avendo i grigliati opportuni):
 - ▶ `+proj=longlat +ellps=WGS84 +to +proj=longlat +ellps=WGS84 +nadgrids=./wgs84_to_roma40.gsb`

geometry_columns

- ▶ Questa è la seconda tabella di sistema
- ▶ Contiene i “metadati” minimi per tutte le colonne geometriche:
 - ▶ schema: nome dello schema che contiene la tabella
 - ▶ table_name: nome della tabella geometrica
 - ▶ geometry_column: nome della colonna di tipo GEOMETRY
 - ▶ coord_dimension: numero dimensioni (2,3, o 4)
 - ▶ srid: codice del sistema di riferimento, può essere -1 = non definito
 - ▶ type: sottotipo geometrico; rappresenta un vincolo ai tipi geometrici contenuti. Può valere GEOMETRY=nessun vincolo

geometry_columns dal sito WEB IGM



	oid	f_table [PK] character	f_table_schem [PK] character	f_table_name [PK] character var	f_geometry_column [PK] character	coord_dimension integer	srid integer	type character varying
1	435694	"	catalogo	carta	the_geom	2	4326	MULTIPOLYGON
2	435698	"	catalogo	carta_fi	the_geom	2	90000	MULTIPOLYGON
3	435705	"	catalogo	prodotto	geom	2	90000	MULTIPOLYGON
4	435697	"	catalogo	prodotto_geo	geom	2	4326	MULTIPOLYGON
5	435696	"	catalogo	prodotto_label	geom	2	90000	MULTIPOINT
6	435713	"	catalogo	t1000	geom	2	90000	POLYGON
7	435715	"	catalogo	t250	geom	2	90000	POLYGON
8	435714	"	catalogo	t500	geom	2	90000	POLYGON
9	435695	"	catalogo	tavolette	the_geom	2	4326	MULTIPOLYGON
10	435703	"	catalogo	tavolette_fi	the_geom	2	90000	MULTIPOLYGON
11	435704	"	catalogo	tavolette_lt	the_geom	2	90000	POINT
12	435706	"	igm95	funico	geom	2	-1	POINT
13	435709	"	igm95	funico2	geom	2	-1	POINT
14	435710	"	igm95	funico3	geom	2	-1	POINT
15	435711	"	igm95	funico4	geom	2	-1	POINT
16	435712	"	igm95	funico5	geom	2	-1	POINT
17	435701	"	limiti	comuni	geom	2	90000	MULTIPOLYGON
18	435700	"	limiti	province	geom	2	90000	MULTIPOLYGON
19	435699	"	limiti	regioni	geom	2	90000	MULTIPOLYGON
20	435708	"	live	funico	geom	2	-1	POINT
21	435702	"	topo	topo	geom	2	90000	MULTIPOINT
22	435707	"	trigo	funico	geom	2	-1	POINT
23	435692	"	voli	anni3	geom	2	90000	MULTIPOLYGON
24	665133	"	voli	anni4	geom	2	90000	MULTIPOLYGON

Scratch pad

Note su `geometry_columns` 1

- ▶ Definire i metadati minimi è fondamentale per il corretto funzionamento di PostGIS (e dei GIS connessi ad esso!)
- ▶ Una tabella può avere più colonne geometriche (corrisponde ad una feature class in cui ogni oggetto a più rappresentazioni: es. città puntuali o areali)
- ▶ Una colonna geometrica:
 - ▶ può contenere dati geometrici di tipo uniforme (scelta consigliata per non mettere in crisi i sistemi GIS), ad esempio punti, in questo caso `geometry_columns.type` contiene il valore POINT
 - ▶ Può contenere dati di tipo misto (punti, linee ed aree), in questo caso `geometry_columns.type` contiene il valore GEOMETRY

Note su `geometry_columns` 2

- ▶ Il sistema di riferimento può essere:
 - ▶ Non indicato: SRID = -1
 - ▶ Indicato in `geometry_columns`, quindi per l'intera colonna (feature)
 - ▶ Indicato dentro il campo `geometry`, quindi per ogni elemento. In questo caso può variare da elemento ad elemento della stessa colonna (cosa sconsigliata)
 - ▶ Alcuni sistemi esterni non riconoscono il sistema di riferimento correttamente:
 - ▶ Shp2pgsql non è in grado di decodificare il file prj
 - ▶ ArcGIS non è in grado di interpretare lo SRID Postgres, il sistema di riferimento deve essere comunicato dall'utente.

funzioni

- ▶ PostGIS definisce circa 700 funzioni SQL
- ▶ Vedremo alcune delle funzioni in dettaglio durante l'esercitazione. Le funzioni sono divise nei gruppi:
 - ▶ Di Gestione: es. *AddGeometryColumn*
 - ▶ Di Costruzione: es. *ST_MakePoint*
 - ▶ Di accesso: es. *ST_Dimension*
 - ▶ Di modifica: es. *ST_Transform*
 - ▶ Di output: es. *ST_AsGML*
 - ▶ Di relazione e misura: es. *ST_Area* e *ST_Intersects*
 - ▶ Di processamento: es. *ST_Buffer*
 - ▶ Altro: es. Linear Referencing, *ST_Xmax*, etc.

Lavorare con Postgres

In questa sezione iniziamo a fare qualche esercizio con SQL e i dati geografici

Valori letterali 1

- ▶ I valori geometrici possono essere specificati in modo letterale, seguendo lo standard OGC WKT:
- ▶ Punto 2D:
 - ▶ 'POINT(0 0)'
- ▶ Polilinea 3D:
 - ▶ 'LINESTRING(0 0 0,1 1 0,1 2 0)'
- ▶ Poligono (con buco):
 - ▶ 'POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1))'
- ▶ Punto multiplo:
 - ▶ 'MULTIPOINT(0 0,1 2)'
- ▶ Punto con sistema di riferimento:
 - ▶ 'SRID=32632;POINT(0 0)'

Valori letterali 2

- ▶ I valori letterali di tipo **GEOMETRY** non possono essere utilizzati così come sono, altrimenti il db li scambia per stringhe, ma:
 - ▶ Si può utilizzare la funzione **ST_GeomFromEWKT** per convertirli in geometria
 - ▶ Si può accodare la dicitura **::GEOMETRY** per informare il sistema che il valore è una geometria.
- ▶ Es. provate ad eseguire la query:

```
SELECT ST_XMax(  
    ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)')  
);
```
- ▶ Nella query prededente abbiamo costruito una geometria e poi abbiamo utilizzato la funzione **ST_XMAX** per estrarre il valore x massimo.

Valori letterali 3

- ▶ Vogliamo proiettare le coordinate geografiche del centro di Firenze in UTM fuso32N, proviamo:

```
SELECT ST_Transform(  
    'SRID=4326;POINT(11.25 43.75)'::GEOMETRY  
    ,32632  
)
```

- ▶ Abbiamo costruito la geometria di un punto in coord. Geografiche (con l'opzione ::GEOMETRY) e l'abbiamo trasformato tramite la funzione ST_Transform a cui bisogna specificare il sistema di arrivo.
- ▶ La risposta di default è in binario e quindi faticiamo a leggerla. Per leggerla in testo applichiamo la funzione ST_AsEWKT a ST_Transform.

Creazione di una feature class: dati

- ▶ Creiamo adesso una feature a mano
- ▶ Questo vuol dire creare una tabella con una colonna di tipo geometry + i metadati associati

- ▶ Creiamo la feature edificio eseguendo:

```
CREATE TABLE edificio
(
    id INTEGER PRIMARY KEY,
    descr CHARACTER VARYING
);
```

- ▶ Nota: il vincolo chiave primaria è stato inserito direttamente dopo la definizione del tipo
- ▶ Nota2: per ora il campo geometrico non c'è. Potevamo inserirlo subito (aggiungendo la colonna "shape GEOMETRY") ma preferiamo farlo in un altro modo...

Creazione di una feature: geometria

- ▶ Postgres contiene la funzione *AddGeometryColumn* che permette di aggiungere la colonna geometrica, i metadati e alcuni vincoli aggiuntivi in un sol colpo.
- ▶ La funzione ha i seguenti parametri:
 - ▶ Tabella su cui operare
 - ▶ Nome della colonna geometrica
 - ▶ Sistema di riferimento
 - ▶ Tipo geometrico
 - ▶ Numero dimensioni
- ▶ Provate ad eseguire (ormai dovrebbe essere chiaro):

```
SELECT  
AddGeometryColumn('edificio','shape',4623,'POLYGON',2);
```

Creazione dell'indice spaziale

- ▶ Perché il funzionamento delle query spaziali sia veloce è fondamentale creare un indice sulla colonna *shape*

- ▶ La sintassi generica per creare indici è:

CREATE INDEX nome_indice ON nome_tabella (colonne)

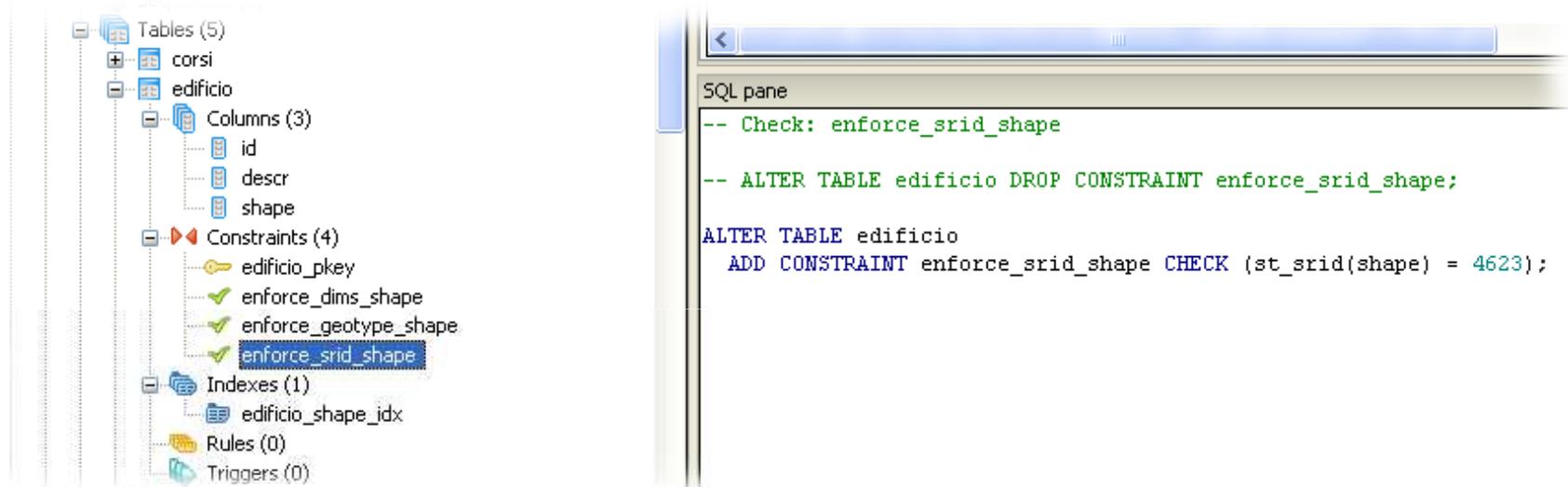
- ▶ Gli indici spaziali però sono diversi da quelli su numeri e parole: in questo caso bisogna specificarne il tipo (che si chiama *gist*): provate quindi ad eseguire:

```
CREATE INDEX edificio_shape_idx  
ON edificio USING gist (shape);
```

- ▶ Il nostro indice spaziale è in funzione

Uno sguardo alla nostra tabella

- ▶ Diamo un sguardo alla nostra tabella nell'albero grafico



- ▶ Oltre agli attributi e all'indice, la funzione *AddGeometryColumn* ha aggiunto vincoli (constraints) aggiuntivi che bloccano il sistema di riferimento, il numero di dimensioni e il tipo geometrico

Uno sguardo ai metadati

- ▶ Selezionate nell'albero grafico la tabella (dello schema public) *geometry_column*
- ▶ Visualizzate la tabella e vedrete che questa contiene i metadati minimi della nostra feature:
 - ▶ Nome dello schema
 - ▶ Nome della tabella
 - ▶ Nome del campo geometrico
 - ▶ Dimensioni (2)
 - ▶ Sistema di riferimento (4623=WGS84)
 - ▶ Tipo geometrico

	[PK] character	[PK] character	[PK] character	[PK] character	integer	integer	character var
24595	"	public	edificio	shape	2	4623	POLYGON
17245	"	public	r010101	shape	4	32533	MULTIPOLYGON

Popolamento della tabella

- ▶ Creiamo adesso qualche oggetto geografico, provate ad eseguire:

```
INSERT INTO edificio
```

```
VALUES
```

```
(1, 'Ospedale',
```

```
'SRID=4623;POLYGON((6 42, 8 42, 8 43, 6 43, 6 42))'::GEOMETRY  
);
```

- ▶ I primi due valore sono la chiave primaria e la descrizione, il terzo è la geometria (un poligono rettangolare) del giusto sistema di riferimento
- ▶ Nota: le aree vanno “chiuso” vale a dire che l’ultimo vertice deve coincidere con il primo (ovvero i quadrati hanno 5 vertici...)

Ancora altri dati

- ▶ Divertiamoci anche a creare un secondo oggetto:

```
INSERT INTO edificio
```

```
VALUES
```

```
(2, 'Industria',
```

```
'SRID=4623;POLYGON((10 43, 13 43, 13 46, 10 46, 10 43),(11  
44, 12 44, 12 45, 11 45, 11 44))'::GEOMETRY);
```

- ▶ Nota: la riga della geometria qui sopra va scritta tutta di seguito (PowerPoint invece va a capo)
- ▶ La geometria della nostra industria è un rettangolo con un buco nel centro: la seconda serie di coordinate definisce il buco centrale.

Semplici analisi sulle nostre feature

- ▶ Proviamo adesso a tabellare le aree dei nostri edifici con la query:

```
SELECT id,descr,  
       ST_Area(shape)  
FROM edificio;
```

- ▶ L'area viene misurata in gradi quadrati... proviamo invece a proiettare queste coordinate nel fuso 32:

```
SELECT id,descr,  
       ST_Area( St_Transform(shape,32632) )/10000  
FROM edificio;
```

- ▶ In questo caso il risultato è in ettari
- ▶ Nei semplici esempi fino a qui visti, le funzioni vengono applicate ad ogni singolo oggetto

Semplici funzioni aggreganti

- ▶ Vediamo ora qualche esempio di funzione aggregante.
Provate ad eseguire:

```
SELECT ST_Extent(shape) FROM edificio;
```

- ▶ Il risultato è:

```
"BOX(6 42,13 46)"
```

- ▶ La funzione *ST_Extent* calcola l'estensione massima dell'unione degli elementi della tabella. Un altro esempio:

```
SELECT St_AseWKT(ST_Union(shape))  
FROM edificio;
```

- ▶ La funzione *ST_Union* calcola l'unione di tutte le geometrie

Connessione con QGIS (oppure UDIG)

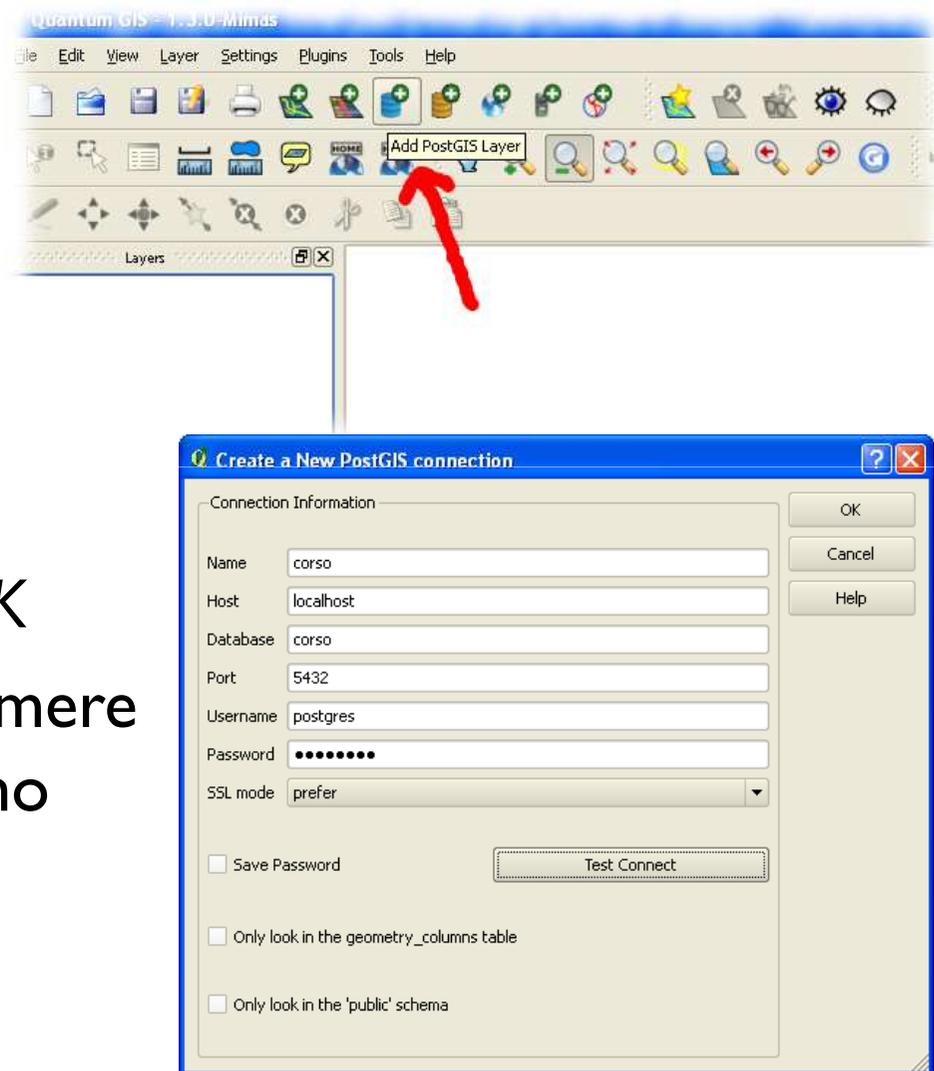
Ed ora un piacevole intermezzo: visualizzazione di dati tramite
QGIS

Visualizzare i dati: introduzione

- ▶ Questa è un'esercitazione su Postgres-PostGIS
- ▶ Questo software non ha di per sé un visualizzatore grafico.
- ▶ E' difficile però continuare senza visualizzare i nostri dati.
- ▶ Introdurremo quindi i concetti minimi per poter visualizzare i dati con QGIS
- ▶ Una descrizione di QGIS esula dagli scopi di questa dispensa.

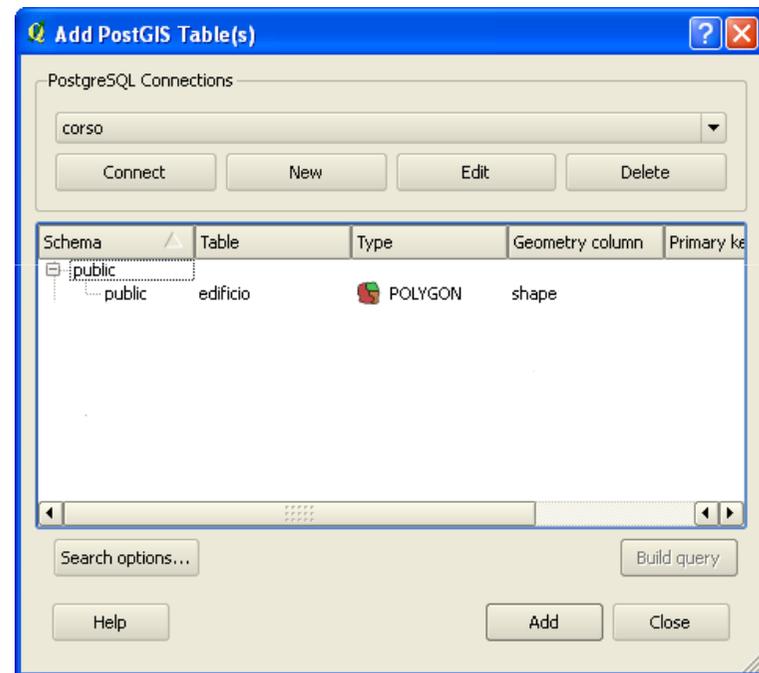
Procedura di visualizzazione con QGIS 1

- ▶ Lanciare QGIS
- ▶ Premere il pulsante: Add PostGIS Layer
- ▶ Nel dialogo che si apre premere il tasto *new*
- ▶ Inserire i valori corretti di connessione e premere *OK*
- ▶ Nel dialogo principale premere il tasto *Connect*: appariranno tutte le feature disponibili

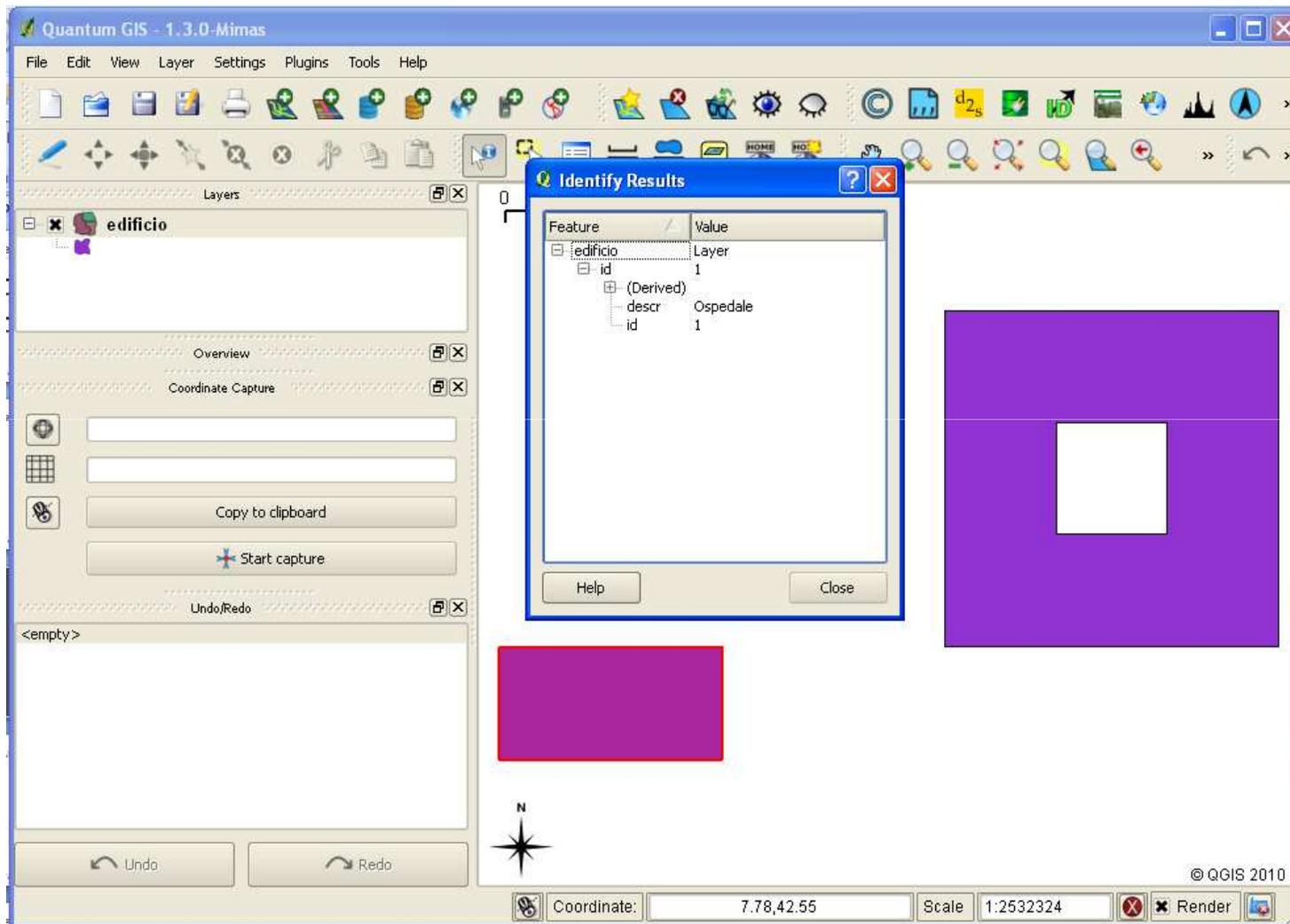


Procedura di visualizzazione con QGIS 2

- ▶ Selezionare la feature public – edifici e premere il tasto *Add*
- ▶ Se tutto va bene apparirà nella mappa l'elenco degli oggetti da noi manualmente creati (vedi prossimo lucido)
- ▶ Selezionando lo strumento Identify Feature (I) è possibile anche interrogare gli attributi degli oggetti.
- ▶ Possiamo mantenere QGIS aperto mentre continuiamo a lavorare con Postgres

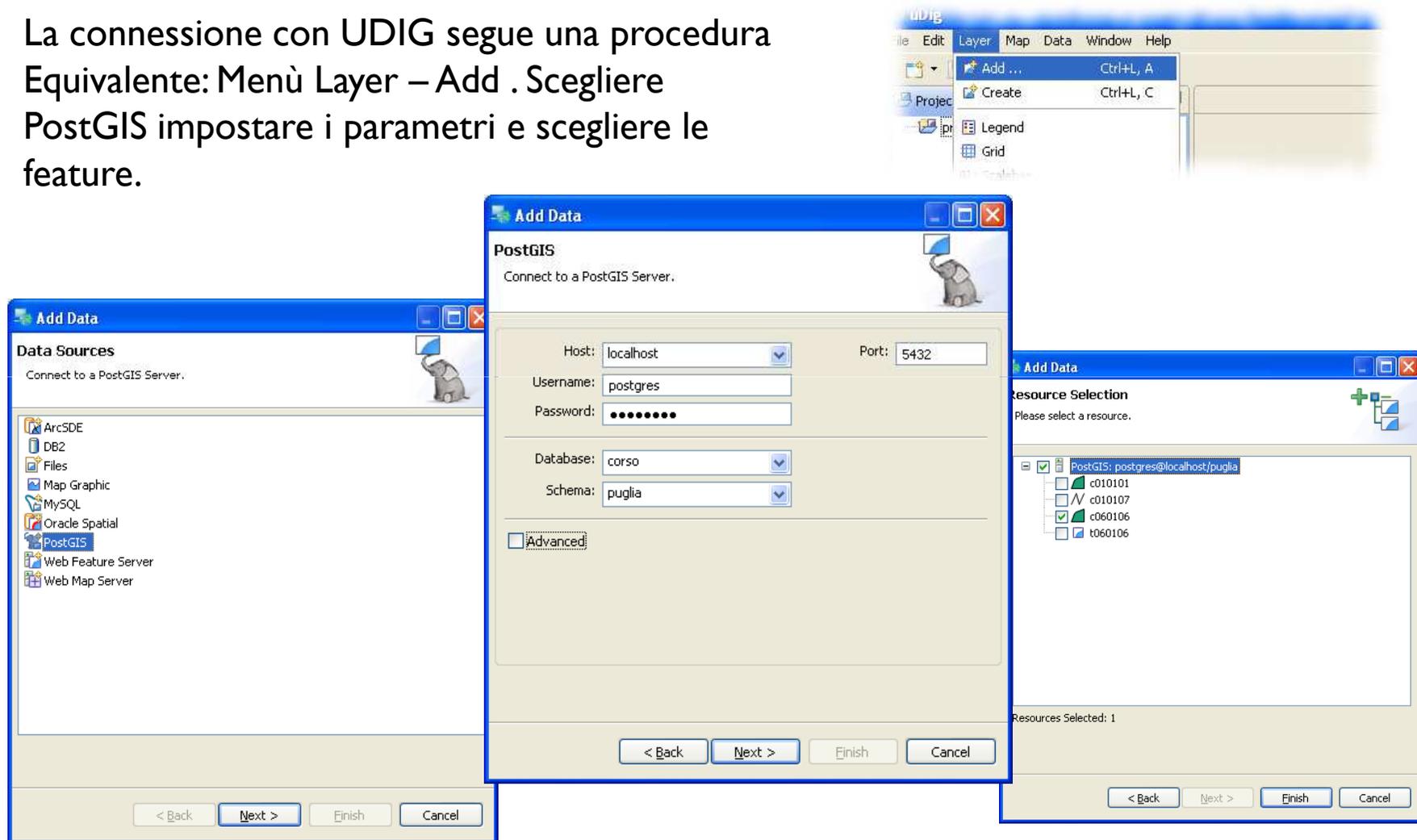


Risultato della visualizzazione



Un inciso connessione con UDig

La connessione con UDig segue una procedura Equivalente: Menù Layer – Add . Scegliere PostGIS impostare i parametri e scegliere le feature.



Importazione di dati Esterni (shape)

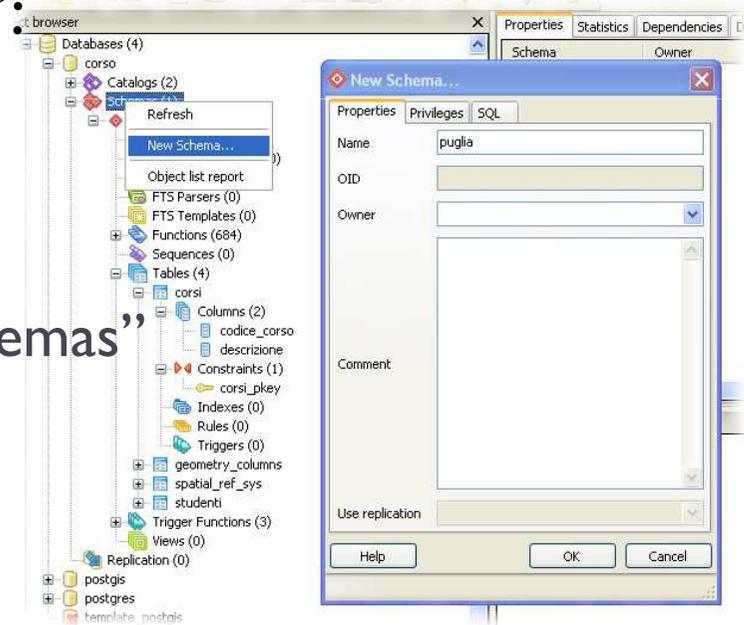
In questa sezione si importano dei dati esterni nel formato
shape

Dati di Esempio

- ▶ Come dati di esempio, utilizzeremo il DB topografico della regione Puglia, in particolare i dati del foglio 383 e solo le feature C010101 (area stradale), C010107 (elemento stradale) e C060106 (Coltura Agricola).
- ▶ In particolare l'ultima feature contiene un attributo multivalore
- ▶ L'importazione dei file shape avverrà attraverso lo strumento "shp2pgsql", un tool di PostGIS. Lo stesso strumento può importare tabelle di dati puri (non geografici).

Preparazione all'importazione

- ▶ Importeremo i nostri file shape nel database Postgres di lavoro
- ▶ I dati però non verranno caricati nella schema principale, ma in un nuovo schema denominato “puglia”.
- ▶ Creazione dello schema “puglia”:
 - ▶ Aprite pgAdmin III
 - ▶ Selezionate il server da utilizzare
 - ▶ Selezionate il database “corso”
 - ▶ Clicca col bottone destro su “Schemas”
 - ▶ Selezionate “New Schema”
 - ▶ Digitate “puglia” come nome
 - ▶ Premete OK



Funzionamento di shp2pgsql

- ▶ Shp2pgsql è un software dos (a riga di comando) funziona digitando il comando in una finestra dos, con le opportune opzioni

USO: shp2pgsql [<options>]
<shapefile>
[<schema>.]<table>

OPTIONS:

- s <srid> Seleziona lo SRID.
- d Cancella e ricrea le tabelle

- a Appende i dati
- c Crea nuove tabelle e dati
- p Crea le tabelle senza dati
- g <geometry_column>
specifica il nome del campo geometrico
- I Crea l'indice spaziale
- W <encoding> Specifica la codifica carattere
- n Importa solo i dbf

Procedura di importazione

- ▶ Creare la cartella c:\lavoro
- ▶ Copiarci dentro i file shape e dbf da importare
- ▶ Aprire una shell DOS (ad esempio premendo Start – Esegui, digitando cmd e premendo OK)

- ▶ Digitare il comando:

- ▶ cd \lavoro

- ▶ Digitare il comando:

- ▶ shp2pgsql -s 32633 -c -g shape -I C010101_POL.shp
puglia.C010101 > C010101.sql

- ▶ Viene creato il file c010101.sql



Analisi dei parametri

- ▶ **-s 32633** : seleziona il sistema di riferimento (32633 = UTM Fuso 32N, WGS84, il sistema utilizzato dalla Puglia), il sistema va indicato perché Postgres non riesce a leggere i file prj
- ▶ **-c** : crea tabella e carica i dati
- ▶ **-g shape** : seleziona “shape” come nome della colonna geometrica
- ▶ **-l** : crea anche l’indice spaziale
- ▶ **C010101_POL.shp** : il file da importare
- ▶ **puglia.C010101** : nome della tabella da creare (nello schema “puglia”)

Importazione delle altre feature

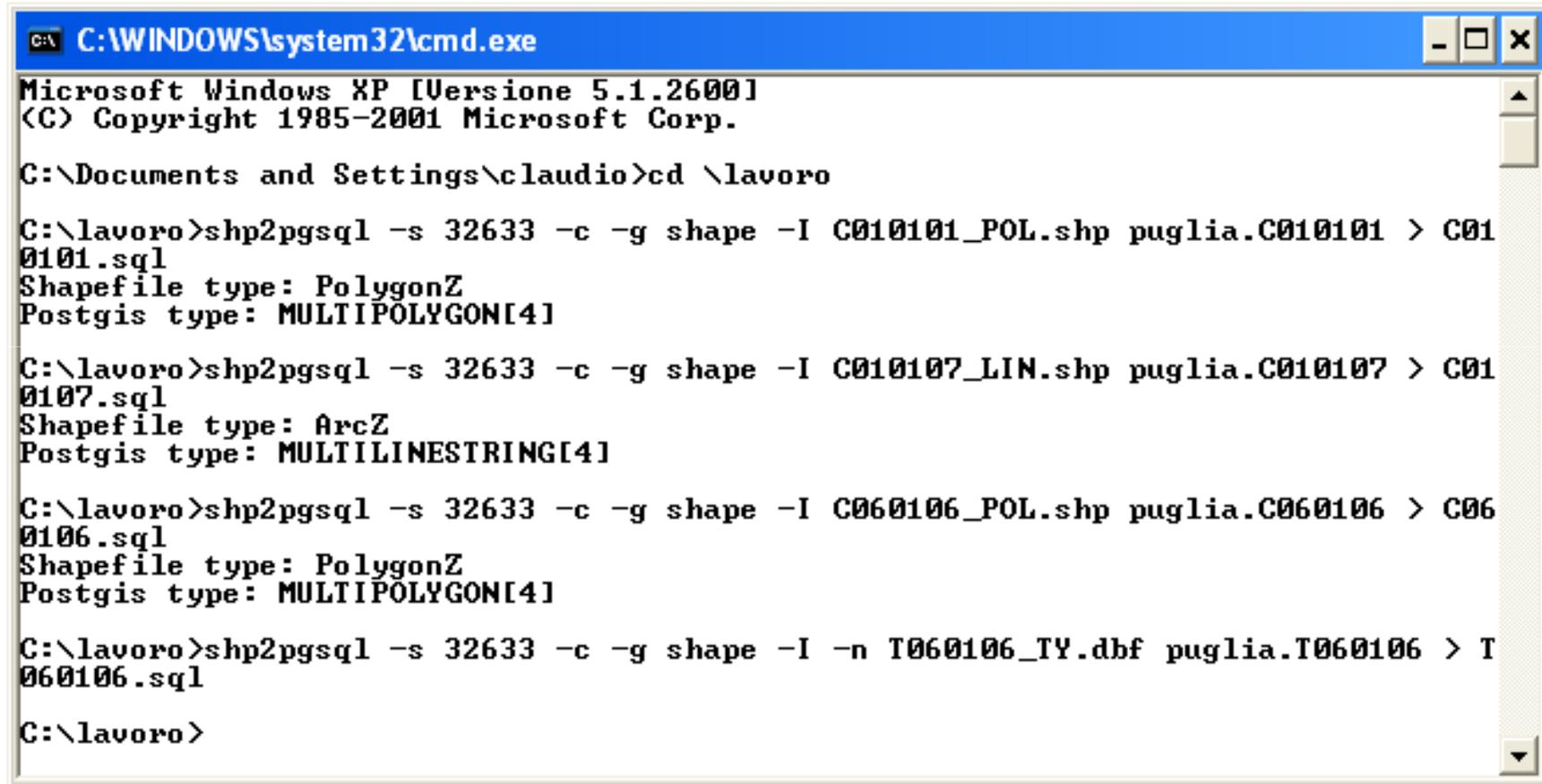
- ▶ Si continua con il comando:
 - ▶ `shp2pgsql -s 32633 -c -g shape -l C010107_LIN.shp puglia.C010107 > C010107.sql`
- ▶ Quindi con il comando:
 - ▶ `shp2pgsql -s 32633 -c -g shape -l C060106_POL.shp puglia.C060106 > C060106.sql`
- ▶ Infine rimane da importare la tabella di soli dati T060106: questa non è una feature geografica ma una tabella pura, utilizzare per implementare un attributo multivalore. In questo caso va aggiunto l'opzione `-n` al posto dell'opzione `-l` (che non serve). Digitate il comando:
 - ▶ `shp2pgsql -s 32633 -c -g shape -n T060106_TY.dbf puglia.T060106 > T060106.sql`

Uno sguardo ai file prodotti

- ▶ Lo strumento utilizzato produce dei comandi sql

```
SET STANDARD_CONFORMING_STRINGS TO ON;
BEGIN;
CREATE TABLE "puglia"."c010101" (gid serial PRIMARY KEY,
"file_id" int4,
"codice_in" varchar(50),
"fonte" varchar(8),
"rilievo" varchar(8),
"tipo_elab" varchar(8),
"data_note" varchar(254),
"livello" int2,
"scala" varchar(10),
"ac_vei_zon" varchar(8),
"ac_vei_fon" varchar(8),
"ac_vei_sed" varchar(8),
"ac_vei_liv" varchar(8),
"shape_leng" numeric,
"shape_area" numeric);
SELECT AddGeometryColumn('puglia','c010101','shape','32633','MULTIPOLYGON',4);
INSERT INTO "puglia"."c010101" ("file_id","codice_in","fonte","rilievo" ...
```

Finestra DOS durante l'importazione



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\claudio>cd \lavoro

C:\lavoro>shp2pgsql -s 32633 -c -g shape -I C010101_POL.shp puglia.C010101 > C010101.sql
Shapefile type: PolygonZ
Postgis type: MULTIPOLYGON[4]

C:\lavoro>shp2pgsql -s 32633 -c -g shape -I C010107_LIN.shp puglia.C010107 > C010107.sql
Shapefile type: ArcZ
Postgis type: MULTILINESTRING[4]

C:\lavoro>shp2pgsql -s 32633 -c -g shape -I C060106_POL.shp puglia.C060106 > C060106.sql
Shapefile type: PolygonZ
Postgis type: MULTIPOLYGON[4]

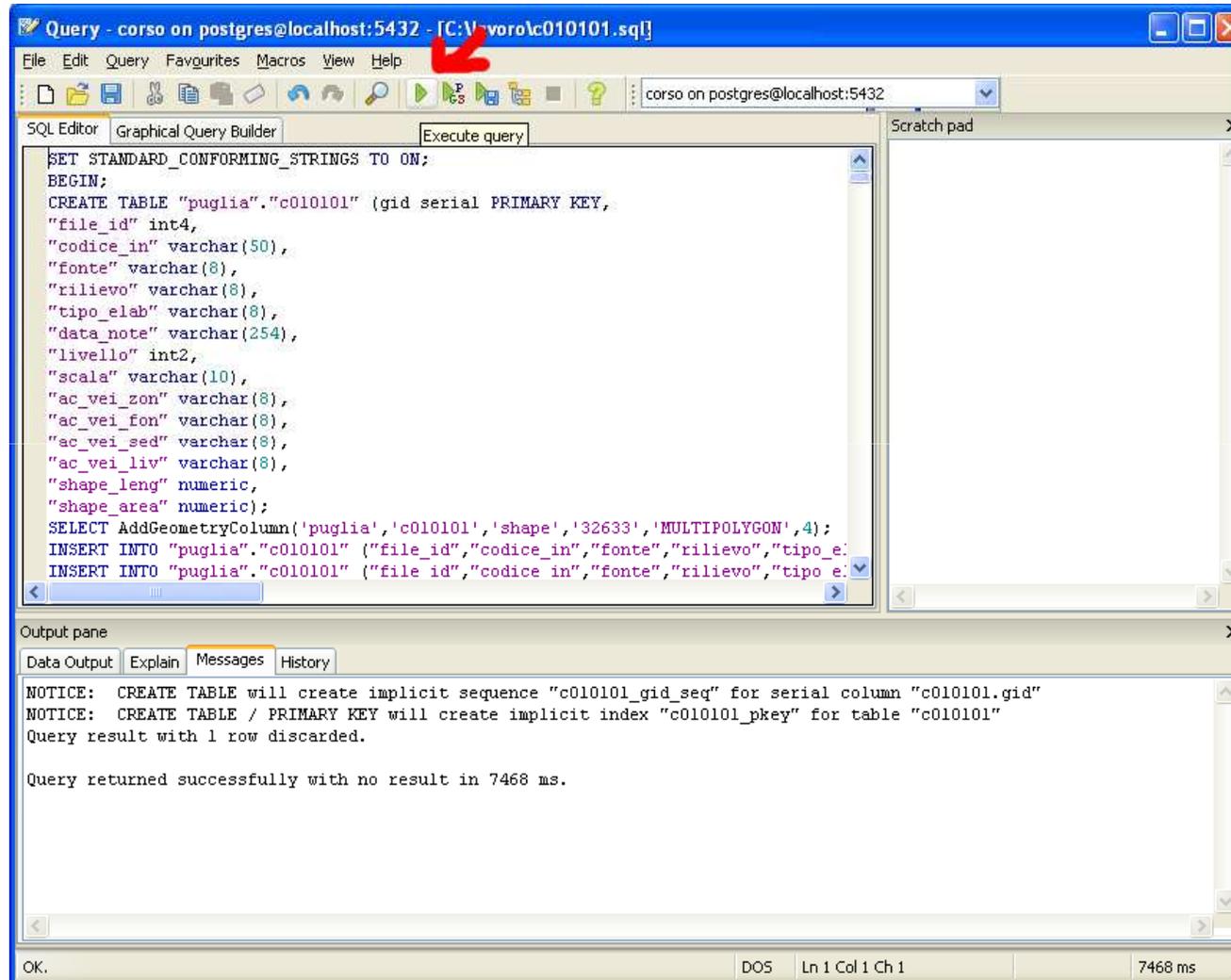
C:\lavoro>shp2pgsql -s 32633 -c -g shape -I -n T060106_TY.dbf puglia.T060106 > T060106.sql

C:\lavoro>
```

Esecuzione dei file SQL

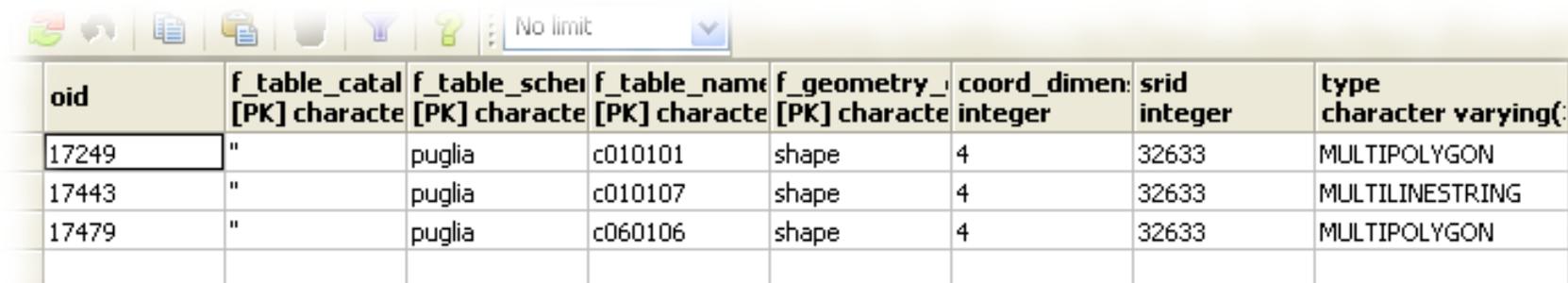
- ▶ Adesso bisogna eseguire i file SQL prodotti.
- ▶ Un modo può essere quello di utilizzare il comando DOS “psql”, ma noi invece utilizzeremo l’interfaccia grafica.
- ▶ Da pgAdmin III, lanciate la finestra SQL (*Execute*)
- ▶ Nella finestra SQL Editor, Selezionate il menù *File – Open ...*, quindi selezionate il file “c010101.sql”
- ▶ Premete il pulsante play (*Execute*) e attendete l’esecuzione dei comandi SQL.
- ▶ Se tutto va bene il messaggio finisce con la frase: “*Query returned successfully with no result in xxx ms.*”

Schermata di esecuzione dell'SQL



Esecuzione degli altri file

- ▶ Ripetete il ciclo di operazioni (Open – Execute) con gli altri file: C060106.sql, C010107.sql e T060106.sql.
- ▶ Attendete pazientemente il risultato di ogni esecuzione.
- ▶ Il caricamento dei dati è finito.
- ▶ Potete dare uno sguardo ai metadati: nell'interfaccia grafica selezionate la tabella public.geometry_columns, quindi premete il pulsante “View data” (Tabella):



The screenshot shows a database query result window with a toolbar at the top and a table of data below. The toolbar includes icons for refresh, print, and help, along with a dropdown menu set to "No limit". The table has the following columns and data:

oid	f_table_catalog [PK] character	f_table_schema [PK] character	f_table_name [PK] character	f_geometry_type [PK] character	coord_dimension integer	srid integer	type character varying(255)
17249	"	puglia	c010101	shape	4	32633	MULTIPOLYGON
17443	"	puglia	c010107	shape	4	32633	MULTILINESTRING
17479	"	puglia	c060106	shape	4	32633	MULTIPOLYGON

Utilizzo dell'importatore grafico

The screenshot shows the 'Shape File to PostGIS Importer' dialog box. It is divided into several sections: 'Shape File', 'PostGIS Connection', 'Configuration', and 'Import Log'. Blue arrows point from text labels on the right to specific fields in the dialog box.

- File**: Points to the 'Shape File' field containing 'gps_fuso32.shp'.
- Utente**: Points to the 'Username' field containing 'pippo'.
- Password**: Points to the 'Password' field containing '*****'.
- Nome DB**: Points to the 'Database' field containing 'geo'.
- Schema**: Points to the 'Destination Schema' field containing 'public'.
- Sistema Riferim.**: Points to the 'SRID' field containing '-1'.
- Colonna geometr.**: Points to the 'Geometry Column' field containing 'the_geom'.

Other fields visible include 'Server Host' (localhost) and 'Port' (5432). Buttons for 'Options...', 'Import', 'About', and 'Cancel' are at the bottom. The 'Import Log' section is empty.

Lavorare con i dati

Adesso che abbiamo dei dati interessanti possiamo eseguire
operazioni reali

Premessa

- ▶ La quantità di operazioni che è possibile eseguire con un database spaziale
- ▶ Il notevole numero di funzioni di base può essere combinato in modi veramente sorprendenti
- ▶ I pochi e banali esempi che seguono servono solo a spronare la fantasia dell'utente.

Semplici dati sui valori geometrici

- ▶ Esiste una serie di funzioni per estrarre le informazioni dei dati geometrici
- ▶ Note:
 - ▶ Le funzioni vengono applicate su tutte le righe della tabella
 - ▶ Aggiungere LIMIT n: gli elementi sono troppi, meglio limitarci ai primi 32
 - ▶ Nota come si scrive una tabella dentro uno schema: schema.tabella

```
SELECT ST_IsSimple(shape),  
       ST_IsValid(shape),  
       ST_NPoints(shape),  
       ST_GeometryType(shape),  
       ST_SRID(shape)  
FROM   puglia.c010101  
LIMIT 32;
```

Semplici controlli statistici

- ▶ Un semplice controllo statistico: numero di elementi e media della superficie degli elementi stradali (raggruppati per tipo di fondo).
- ▶ Note:
 - ▶ count e avg sono funzioni aggregati (Area no!)
 - ▶ E' possibile selezionare ac_vei_fon perché si raggruppa per questo valore

```
SELECT    count (*),  
          ac_vei_fon,  
          avg(ST_Area(shape))  
FROM      puglia.c010101  
GROUP BY ac_vei_fon;
```

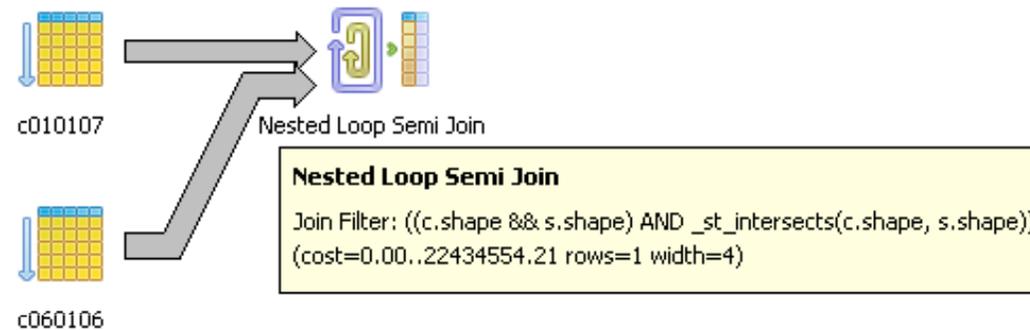
Controllo sovrapposizione

- ▶ La seguente query ritorna l'elenco delle strade che intersecano una qualsiasi coltura.

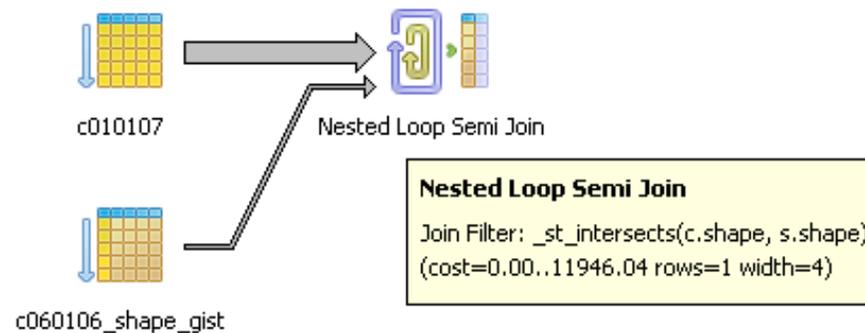
```
SELECT s.gid
FROM   puglia.c010107 s
WHERE EXISTS
( SELECT c.*
  FROM   puglia.c060106 c
  WHERE ST_Intersects(s.shape, c.shape)
)
```

Un inciso: query planner

- ▶ Query senza indice: tempo di esecuzione gigantesco



- ▶ Query in con indice: tempo di esecuzione accettabile



Correttezza con se stessi

- ▶ La seguente query ritorna l'elenco di colture sovrapposte ad altre (sicuro errore di topologia):

```
SELECT c1.gid
FROM   puglia.c060106 c1
WHERE EXISTS
( SELECT c2.*
  FROM   puglia.c060106 c2
  WHERE ST_Intersects(c1.shape,c2.shape)
  AND   c1.gid < c2.gid
)
```



- ▶ Perché questo controllo? (suggerimento: un'area tocca sé stessa?)
- ▶ Perché allora non si è usato il diverso?

Salvare il risultato come Feature

- ▶ Il risultato di una interrogazione può essere importante;
- ▶ È possibile salvare una query creando una nuova tabella (e quindi una nuova feature geografica):

```
CREATE TABLE puglia.ingombro AS
SELECT c010107.gid,
       st_buffer(c010107.shape,
                20::double precision) AS shape
FROM puglia.c010107;
```

Salvare il risultato come Vista

- ▶ Per mantenere il risultato dinamico, invece di una tabella è possibile creare una vista.
- ▶ I dati della vista saranno visti da un GIS (es. QGIS) come una nuova feature.
- ▶ Modificando la tabella di origine, si modifica automaticamente anche la vista (i dati non sono duplicati):

```
CREATE OR REPLACE VIEW puglia.centri AS
SELECT c060106.gid,
       st_centroid(c060106.shape) AS shape
FROM puglia.c060106;
```

Esempi Reali di Interrogazioni

Le query spaziali del mondo reale tendono ad essere un po' più compresse di quanto visto fino ad ora

Esempi reali di applicazioni 1

Collaudo topologico DB PUGLIA:

1.1.16

Agli archi valorizzati con “0201”
(di casello/barriera autostradale)
nell’Attributo “EL_STR_TY”,
devono corrispondere punti
appartenenti alla “Classe 010108
Giunzione Stradale” valorizzati
con “02” nell’Attributo
“GZ_STR_TY”.

```
SELECT a.file_id
FROM XSHEMA.C010107_LIN a
WHERE a.EL_STR_TY = '0201'
AND NOT EXISTS
( SELECT b.*
  FROM XSHEMA.C010108_POI b
  WHERE
    (a.shape && SetSRID(ST_MakeBox2D( ST_MakePoint(
      ST_X(b.shape)-XLTOLL, ST_Y(b.shape)-XLTOLL ),
      ST_MakePoint( ST_X(b.shape)+XLTOLL,
      ST_Y(b.shape)+XLTOLL ) ),ST_SRID(a.shape))
    AND
    ( ST_Distance(
      ST_StartPoint(b.shape),a.shape)<=XLTOLL OR
      ST_Distance(ST_EndPoint
      (b.shape),a.shape)<=XLTOLL )
    )
  AND EXISTS
    ( SELECT c.* FROM XSHEMA.t010108_ty c
      WHERE c.id_gz_str= b.file_id AND c.GZ_STR_TY =
        '02' )
  )
ORDER BY a.file_id;
```

Esempi reali di applicazioni 2

```
INSERT INTO db25.LAP020 (LAB,LOC,MED,NAM,TXT,WD1,GEOMETRY)
  SELECT CASE
    WHEN f.el_str_cf = '01' THEN 'L301A' -- autostrada
    ELSE 'L301B' -- stradale
  END AS LAB,
  x.loc AS LOC,
  '997' AS MED, --
  todo: sarebbe possibile derivarlo dalla presenza di 020205 TY='01'
  'UNK' AS NAM,
  'UNK' AS TXT,
  CASE
    WHEN (SELECT min(ST_Area(g.shape)/ST_Length(f.shape))
          FROM f409.c010101_pol g
          WHERE ST_Contains(g.shape,f.shape)
          AND g.ac_vei_zon = f.el_str_ty
          ) < 20 THEN
      (SELECT min(ST_Area(g.shape)/ST_Length(f.shape))
        FROM f409.c010101_pol g
        WHERE ST_Contains(g.shape,f.shape)
        AND g.ac_vei_zon = f.el_str_ty
        )
    ELSE -9999
  END AS WD1, -- todo: derivabile da areale ???
  f.shape AS GEOMETRY
FROM f409.c010107_lin f,
     todb25.str_sed2loc x
WHERE f.el_str_ty = '0102' -- svincolo
AND f.el_str_sed = x.str_sed;
```