fusion-gis

Purpose:

- Load two RDF datasets into a PostGIS database separating geometries and metadata.

  (In the following *<subjectRegex>* is a user provided regex that matches the URI of those nodes that are of interest to the user.)

  - Metadata is defined as those triples matching the following SPARQL restriction:
    {
    ?s ?p ?o
    FILTER (regex(?s, "<subjectRegex>", "i"))
    FILTER (!regex(?p, "http://www.opengis.net/ont/geosparql#hasGeometry", "i"))
    }
  - Geometries are defined as those triples matching the following SPARQL restriction:
    {
    ?s ?p1 _:a .
    _:a ?p2 ?g
    FILTER(regex(?s, "<subjectRegex>, "i"))
    FILTER(regex(?p1, "http://www.opengis.net/ont/geosparql#hasGeometry", "i"))
    FILTER(regex(?p2, "http://www.opengis.net/ont/geosparql#asWKT", "i"))
    }
    The geometry string (?g) is expected to be in the WKT serialisation format.

- Provide transformations for fusing the geometries of the two loaded datasets and for a given list of links between the two datasets and a specific transformation:
  - score the transformation on its suitability for those links
  - apply the transformation on the geometries belonging to the linked nodes

Requirements

Fusion-gis requires that postgreSQL is installed along with the PostGIS extension.

Testing

Testing was performed using PostgreSQL v9.1.9 w/ PostGIS v2 and Virtuoso v07.00.3203.

PostGIS Schema

```
--Database creation script for the importer PostGIS schema


--Drop all tables if they exist
DROP TABLE IF EXISTS dataset_a_info;
DROP TABLE IF EXISTS dataset_a_metadata;
```

```
DROP TABLE IF EXISTS dataset_a_geometries;
DROP TABLE IF EXISTS dataset_b_info;
DROP TABLE IF EXISTS dataset_b_metadata;
DROP TABLE IF EXISTS dataset_b_geometries;
DROP TABLE IF EXISTS fused_geometries;


--Create a table to hold datasetA's info
CREATE TABLE dataset_a_info (
      endpoint text NOT NULL,
      graph text NOT NULL
);

--Create a table to hold datasetA's metadata
CREATE TABLE dataset_a_metadata (
      id serial PRIMARY KEY,
      subject text NOT NULL,
      predicate text NOT NULL,
      object text NOT NULL,
      object_lang text,
      object_datatype text
);

CREATE INDEX idx_dataset_a_metadata_subject ON dataset_a_metadata USING btree
(subject);

--Create a table to hold datasetA's geometries
CREATE TABLE dataset_a_geometries (
      id serial PRIMARY KEY,
      subject text NOT NULL
);

SELECT AddGeometryColumn('dataset_a_geometries', 'geom', 4326, 'GEOMETRY', 2);
CREATE INDEX idx_dataset_a_geometries_geom ON dataset_a_geometries USING gist
(geom);
CLUSTER dataset_a_geometries USING idx_dataset_a_geometries_geom;

--Create a table to hold datasetB's info
CREATE TABLE dataset_b_info (
```

```
	endpoint text NOT NULL,
	graph text NOT NULL
);

--Create a table to hold datasetB's metadata
CREATE TABLE dataset_b_metadata (
	id serial PRIMARY KEY,
	subject text NOT NULL,
	predicate text NOT NULL,
	object text NOT NULL,
	object_lang text,
	object_datatype text
);

CREATE INDEX idx_dataset_b_metadata_subject ON dataset_b_metadata USING btree
(subject);

--Create a table to hold datasetB's geometries
CREATE TABLE dataset_b_geometries (
	id serial PRIMARY KEY,
	subject text NOT NULL
);

SELECT AddGeometryColumn('dataset_b_geometries', 'geom', 4326, 'GEOMETRY', 2);
CREATE INDEX idx_dataset_b_geometries_geom ON dataset_b_geometries USING gist
(geom);
CLUSTER dataset_b_geometries USING idx_dataset_b_geometries_geom;

--Create a table to hold fused geometries
CREATE TABLE fused_geometries (
	id serial PRIMARY KEY,
	subject_A text NOT NULL,
	subject_B text NOT NULL
);

SELECT AddGeometryColumn('fused_geometries', 'geom', 4326, 'GEOMETRY', 2);
CREATE INDEX idx_fused_geometries_geom ON fused_geometries USING gist (geom);
```

Use cases & execution flow

Import dataset

1. User provides PostGIS connection parameters via *DatabasePanel*.
2. User provides endpoint, graph and regex for matching root nodes (subject regex) for each graph and commands start of import operation via *ImporterPanel*.
3. *ImporterPanel* creates a new instance of *DatabaseInitialiser* and invokes *DatabaseInitialiser.initialise(...)* to create a new PostGIS db and initialise it with the schema.
4. *ImporterPanel* creates two instances of *ImporterWorker,* one for each dataset, and executes them.
5. Each *ImporterWorker* creates an instance of *Importer* that contains the import logic and invokes the *Importer.importMetadata(...), Importer.importGeometries(...)* methods.
6. The *Importer* methods extract the triples from the provided endpoint by executing SPARQL queries via the Jena library and load them in the PostGIS db using methods defined in *PostGISImporter* that make use of the PostgreSQL JDBC driver.
7. User is provided with progress bar updates throughout the import process and is notified when each worker completes execution.

Score transformation

1. User provides PostGIS connection parameters via *DatabasePanel*.
2. User provides a set of links via *FuserPanel* in the form of an N-TRIPLES file with triples in the format:
   <dataset A URI> owl:sameAs <dataset B URI> .
   Datasets A & B correspond to the dataset_a_*, dataset_b_* tables in the SQL schema.
3. Links are parsed using static method *GeometryFuser.parseLinksFile(...)* and returned to *FuserPanel* for display.
4. User selects a transformation via *FuserPanel* and commands start of scoring operation.
5. *FuserPanel* creates an instance of *ScoreWorker* and executes.
6. *ScoreWorker* creates an instance of *GeometryFuser* and invokes *GeometryFuser.score(...)*, which in turn invokes the *score* method of the transformation for each provided link with the linked root node URIs as parameters. The resulting score for each link is entered into a map and returned.
7. *FuserPanel* colours each link according to their score. Magenta for a score >= 0.5, black otherwise.

Apply transformation (fuse)

1. User provides PostGIS connection parameters via *DatabasePanel*.
2. User provides a set of links via *FuserPanel* in the form of an N-TRIPLES file with triples in the format:
   <dataset A URI> owl:sameAs <dataset B URI> .
   Datasets A & B correspond to the dataset_a_*, dataset_b_* tables in the SQL schema.
3. Links are parsed using static method *GeometryFuser.parseLinksFile(...)* and returned to *FuserPanel* for display.
4. User selects a transformation and a set of links to be fused via *FuserPanel* and commands start of fusion operation.
5. *FuserPanel* creates an instance of Fuse*Worker* and executes.
6. *FuseWorker* creates an instance of *GeometryFuser* and invokes *GeometryFuser.fuse(...)*, which in turn invokes the *fuse* method of the transformation for each provided link with the linked root node URIs as parameters.
7. User is notified when the fusion operation has completed for all links.