Developing GeoStreaming API

Introduction

In order to perform spatio-temporal query, spatial data and temporal data should be indexed and spatio-temporal operations should be supported. Supporting these capabilities can be achieved via adapting the way that Alejandro Rodriguez developed Data Stream API 2.0). The goal of this project is

- Developing Tupelo context to support spatio-temporal query

Note that the ontology for Spatial information and query is based on the GeoSPARQL proposal by Oracle. It needs to be updated with the current draft of OGC GeoSPARQL.

Design I

The Figure 1 shows the Spatio-Temporal composite context that supports spatio-temporal queries. The composite context will analyze the query and store temporal data and temporal data to database tables that supports temporal index and spatial index.

![Figure 1. Spatio-temporal Composite Context](image)

Here are how the spatio-temporal query is treated and handled:

- process the part contains Time-annotated resources by DSAPI
- process the rest of the part. If the part contains geospatial predicates
  - `<subject> ogc:hasWKT "POINT(123.1 -23.1)"
  - the literal in WKT format will be store in spatial table
- `<subject> ogc:within "POLYGON(1 2,3 4,2 1,0 0)"
  - perform spatial query with the literal in WKT format

Implementation I

The PostGIS, spatial extension for PostgreSQL dbms is used for spatial database. Since PostGIS follows the OGC spec, the name of spatial SQL functions defined in PostGIS are same as predicates we are using in this development.

Note that temporal query will be performed first then the spatial query will be performed later. Because of the design and implementation of DSAPI, the spatial query will be performed with triple matcher instead of Union query. And it will be the performance bottleneck.

The design is implemented by extending Postgresqlp implementation of StreamingContext and override methods:

**doPerform(TripleWriter tw)**

If there is “ogc:hasWKT” predicate, then write WKT geometry to the PostGIS table
doPerform(TripleMatcher tm)

If the predicate is “ogc:within”, then do geospatial query. Otherwise, perform regular triple-matching

- With unbounded subject
- With bounded subject
- In order to improve the performance, HashSet of query results are maintained

unify(Unifier u, Boolean useComplexJoins)

- Classify patterns into spatial pattern and non-spatial pattern
- Create regular sql statement with non-spatial pattern
- Wrap the sql statement for non-spatial pattern with spatial sql statement

Design II - Better Performance

As mentioned above about the performance bottleneck, the spatial unifier is not used efficiently. In order to overcome this problem, the temporal query needs to be combined with normal query. In other words, the temporal query needs to be translated into SQL statement and combined with other SQL statement for normal query.

Assumption:

- all attributes are stored with token URI, not with stream URI
- tokens table has uid field to store uid of token URI from sym table
  * It need to make new triplewriter for TimeAnnotatedResource
  * write the token resource and get uid
  * write the uid to tokens table

Here are the example unifier query and expected SQL statement to be generated:

<table>
<thead>
<tr>
<th>TimeAnnotatedVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>columns: ?s, ?flavor, ?temp</td>
</tr>
<tr>
<td>pattern 1: ?s urn:flavor &quot;pistachio&quot;</td>
</tr>
<tr>
<td>pattern 3: ?s urn:flavor ?flavor</td>
</tr>
<tr>
<td>pattern 4: ?s[*37..&quot;39&quot;] urn:temp ?temp</td>
</tr>
</tbody>
</table>
select distinct
(select sym from sym where uid=p0.sub) as c0, tok1.token_id as c0_frame_id, tok1.start as c0_time,
(select sym from sym where uid=p1.obj) as c1, (select sym from sym where uid=p1.typ) as c1_typ,
(select sym from sym where uid=p2.obj) as c2, (select sym from sym where uid=p2.typ) as c2_typ
from tup p0
join tup p1 on p1.sub=p0.sub and p1.pre=(select uid from sym where md5(sym)=md5('uurn:flavor'))
join tup p2 on p2.sub=p0.sub and p2.pre=(select uid from sym where md5(sym)=md5('uurn:temperature'))
join tokens tok1 on tok1.uid=p1.sub and tok1.start >= 37 and tok1.end <= 39
where p0.pre=(select uid from sym where md5(sym)=md5('uurn:flavor'))
and p0.obj=(select uid from sym where md5(sym)=md5('l pistachio'));