ORACLE®

Realizing the Benefits of Linked Geospatial Data with R2RML and GeoSPARQL

Souripriya Das
Matthew Perry

Oracle Spatial and Graph
January 30, 2017

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE

Agenda

1. Why Linked Geospatial Data?
2. Review of RDF/OWL/SPARQL
3. RDB2RDF Standard
4. GeoSPARQL Standard
5. Adding GeoSPARQL Support to RDB2RDF
6. Demo

ORACLE

Agenda

1. Why Linked Geospatial Data
2. Review of RDF/OWL/SPARQL
3. RDB2RDF Standard
4. GeoSPARQL Standard
5. Adding GeoSPARQL Support to RDB2RDF
6. Demo

ORACLE

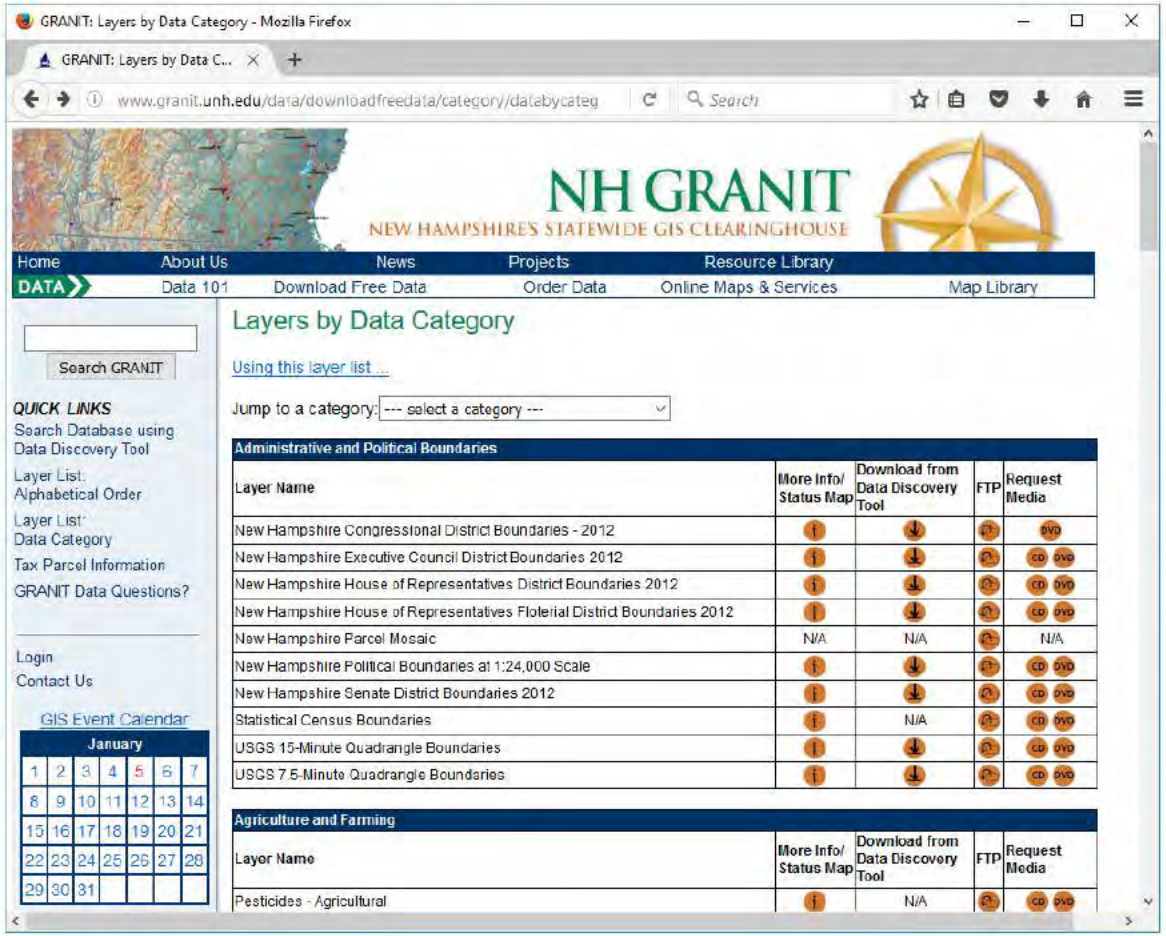
Why Linked Geospatial Data?

- Linked Data is beneficial
  - Web of data rather than documents
  - Based on URIs, links
  - Helps discoverability, accessibility, interoperability, reusability
- Spatial Data is important
  - Almost everything is related to location somehow (especially with ubiquity of GPS-enabled mobile devices).
  - Lots of GIS data is available but hard for non-experts to find and use
- Goal: to make spatial data more discoverable, accessible and interoperable

ORACLE

Traditional Spatial Data Infrastructure (SDI)

- Intended for expert users
- Map/dataset oriented
- Data not accessible by general Web searches
- Depends on metadata catalog, which is often sparsely populated
- Cannot just follow links, usually complex APIs or search interface
- Difficult to understand and use the data (multiple CRS issues, proprietary formats)



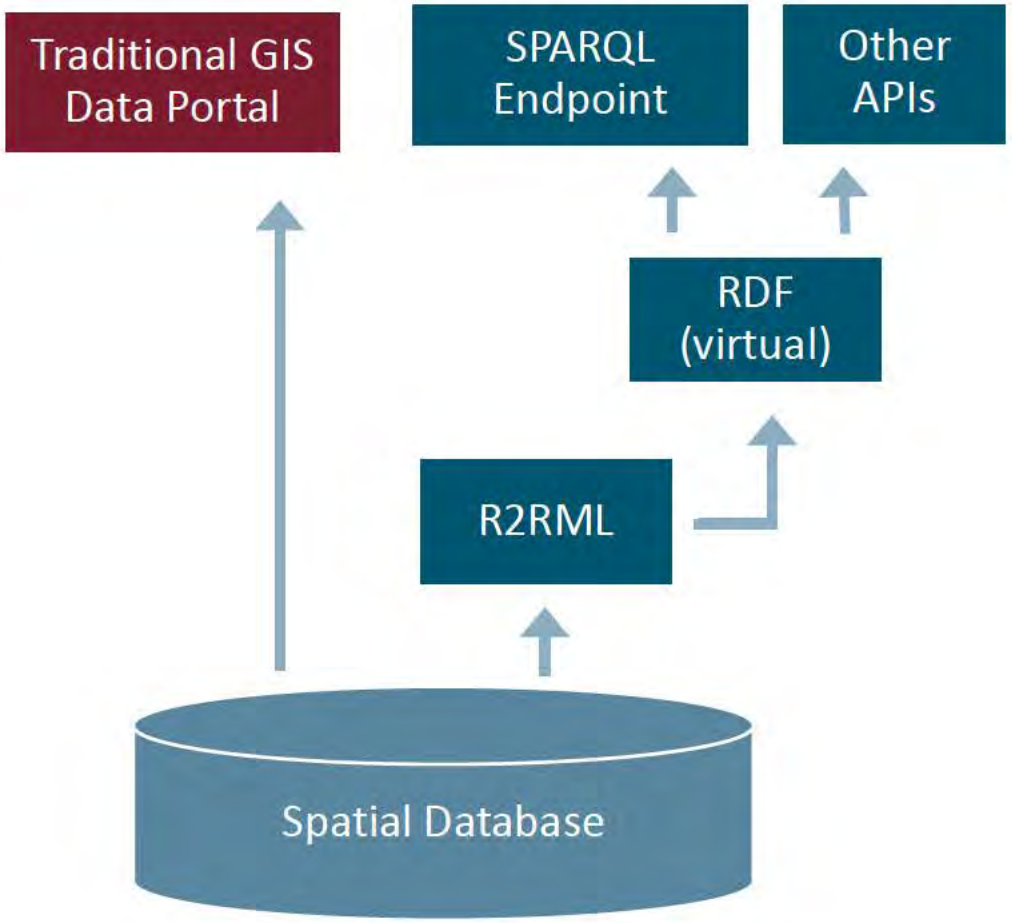
Some issues with SDIs could be alleviated with Linked Data principles

- Globally unique, persistent HTTP URIs for Spatial Things
  - Resource-centric instead of map-centric
  - Facilitates linking, discoverability, interoperability
  - Can help make data crawlable
- Linked Data friendly formats like RDF, JSON-LD
  - Supports web linking
  - More processable by Web infrastructure
- Web-friendly Data access
  - SPARQL endpoint
  - REST APIs that return RDF, JSON-LD

ORACLE

One way to get there

- Most traditional SDIs are underpinned by a Relational Database (Oracle Spatial, PostGIS, etc.)
- W3C RDB2RDF standard (enhanced with spatial capabilities) can be particularly useful to expose this data as Linked Data
  - Provides virtual RDF while allowing data to stay in existing relational form (no ETL)
  - R2RML mappings can be used to provide meaningful HTTP URIs
  - SPARQL endpoint can be used to drive many Web APIs



Agenda

1. Introduction and Motivation
2. Review of RDF/OWL/SPARQL
3. RDB2RDF Standard
4. GeoSPARQL Standard
5. Adding GeoSPARQL Support to RDB2RDF
6. Demo

ORACLE

What is RDF?

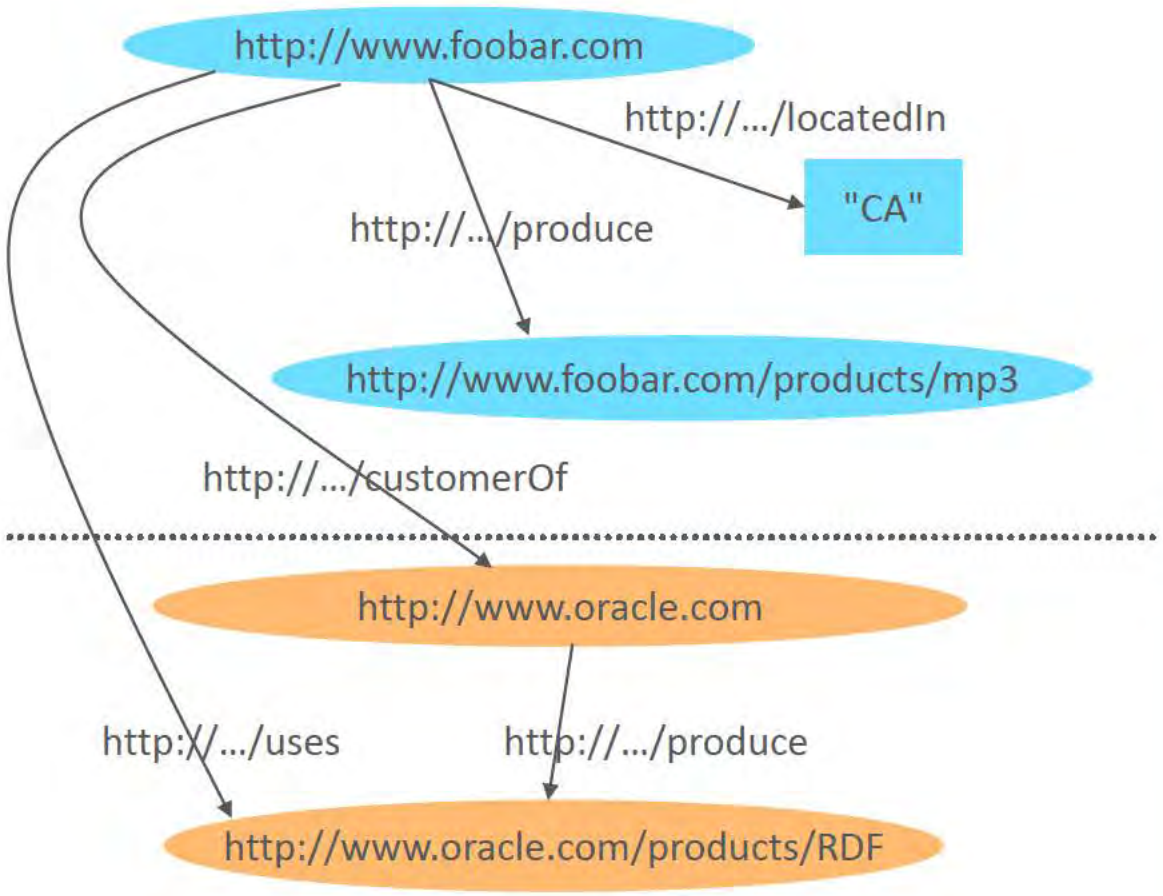
- A graph data model for (web) resources and their relationships
- The graph can be serialized into
  - RDF/XML, N3, N-TRIPLE, N-Quads …
- Construction unit: **Triple**

  (or assertion, or fact)

  <http://foobar> <…/produce> <…/mp3>

  Subject Predicate Object
- **Quads** (**named graphs**) add context, provenance, identification, etc. to assertions

  <http://foobar> <:produces> <:mp3 > **<:ProductGraph>**



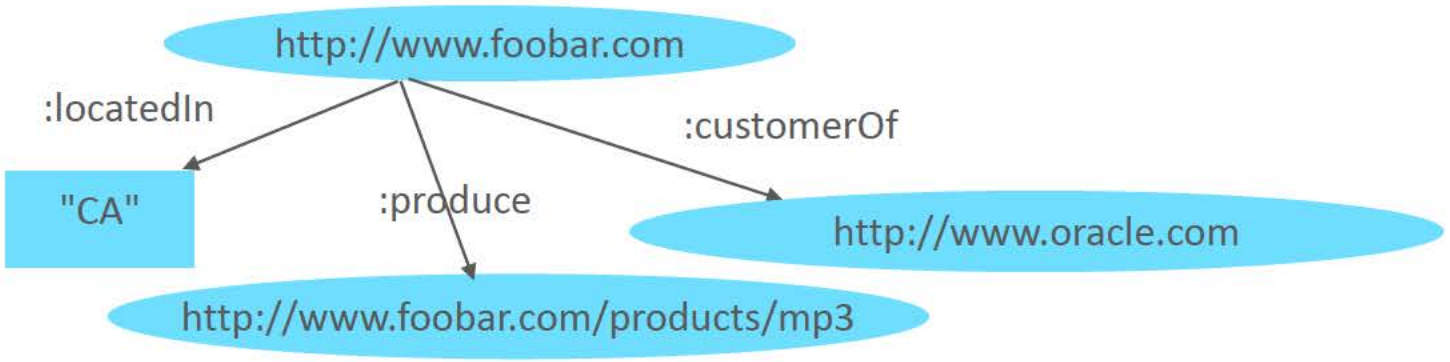
Basic Elements of RDF

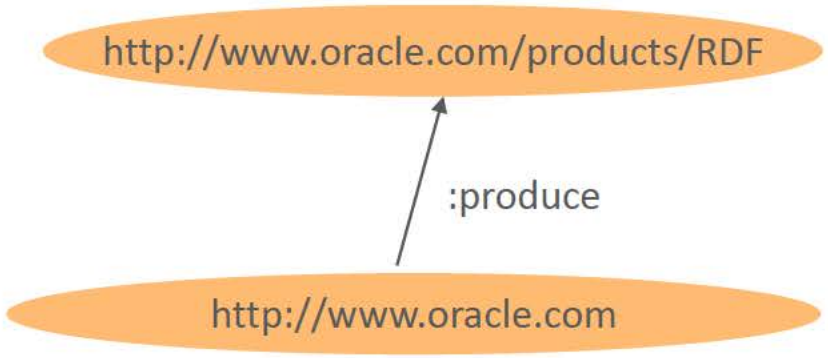
- Instances
  - Examples → :John, :MovieXYZ, :PurchaseOrder432
- Classes
  - Class represents a group/category/categorization of instances
  - Examples → :John rdf:type :Student
- Properties
  - Linking data together
  - Examples → :Mary :hasBrother :John .
    :Mary :wasBornOn "1980-06-08"^^xsd:date .
    :John :graduatedFrom :CalTech .

ORACLE

Triples Are Easy. But Why?

- Graph modeling is flexible
  - Adding and removing edges is simple
- Standard based graph representation
  - RDF was defined by W3C. Allows interoperability.
- Computers can understand the semantics RDF graphs (triples)
  - Same URI means same resource



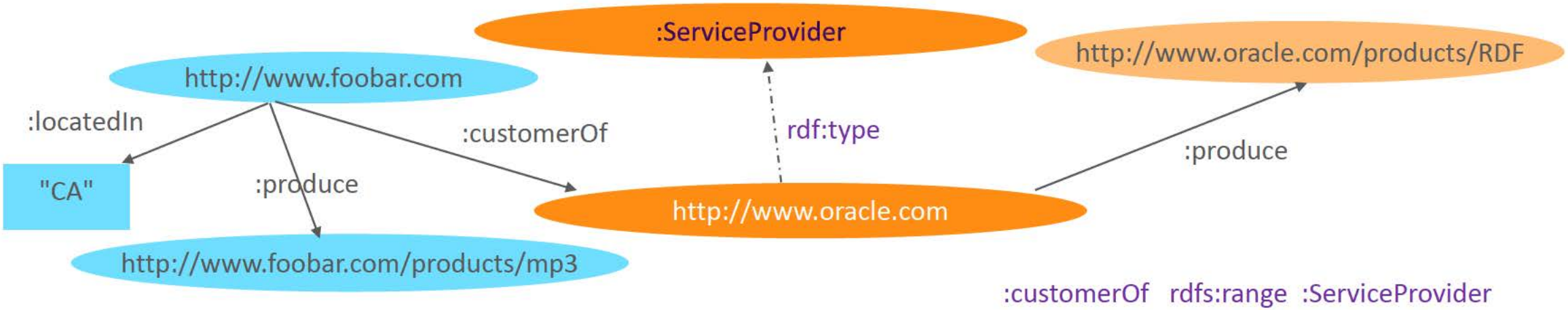


Triples Are Easy. But Why? (2)

- Graph modeling is flexible
  - Adding and removing edges is simple
- Standard based graph representation
  - RDF was defined by W3C. Allows interoperability.
- Computers can understand the semantics RDF graphs (triples)
  - Same URI means same resource

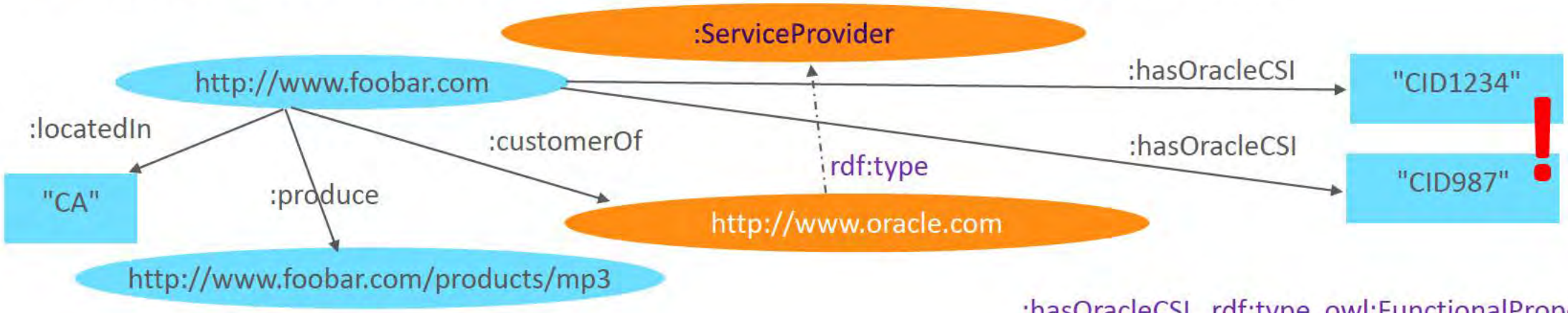
Triples Are Easy. But Why? (3)

- Graph modeling is flexible
  - Adding and removing edges is simple
- Standard based graph representation
  - RDF was defined by W3C. Allows interoperability.
- Computers can understand the semantics RDF graphs (triples)
  - Discover hidden relationships, or detect inconsistency via logical inference



Triples Are Easy. But Why? (4)

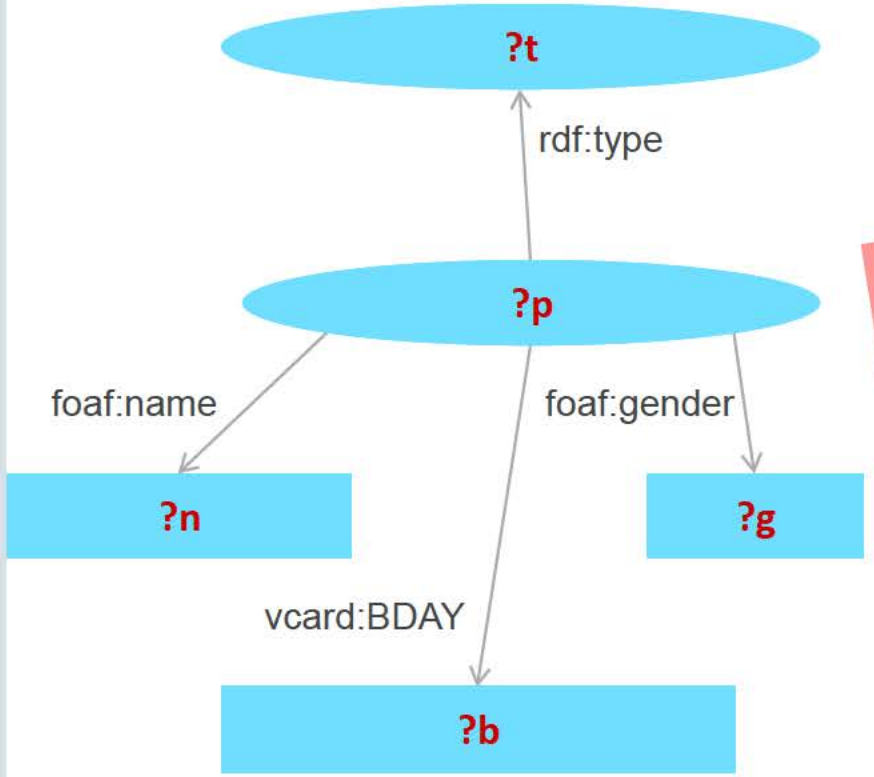
- Graph modeling is flexible
  - Adding and removing edges is simple
- Standard based graph representation
  - RDF was defined by W3C. Allows interoperability.
- Computers can understand the semantics RDF graphs (triples)
  - Discover hidden relationships, or detect inconsistency via logical inference

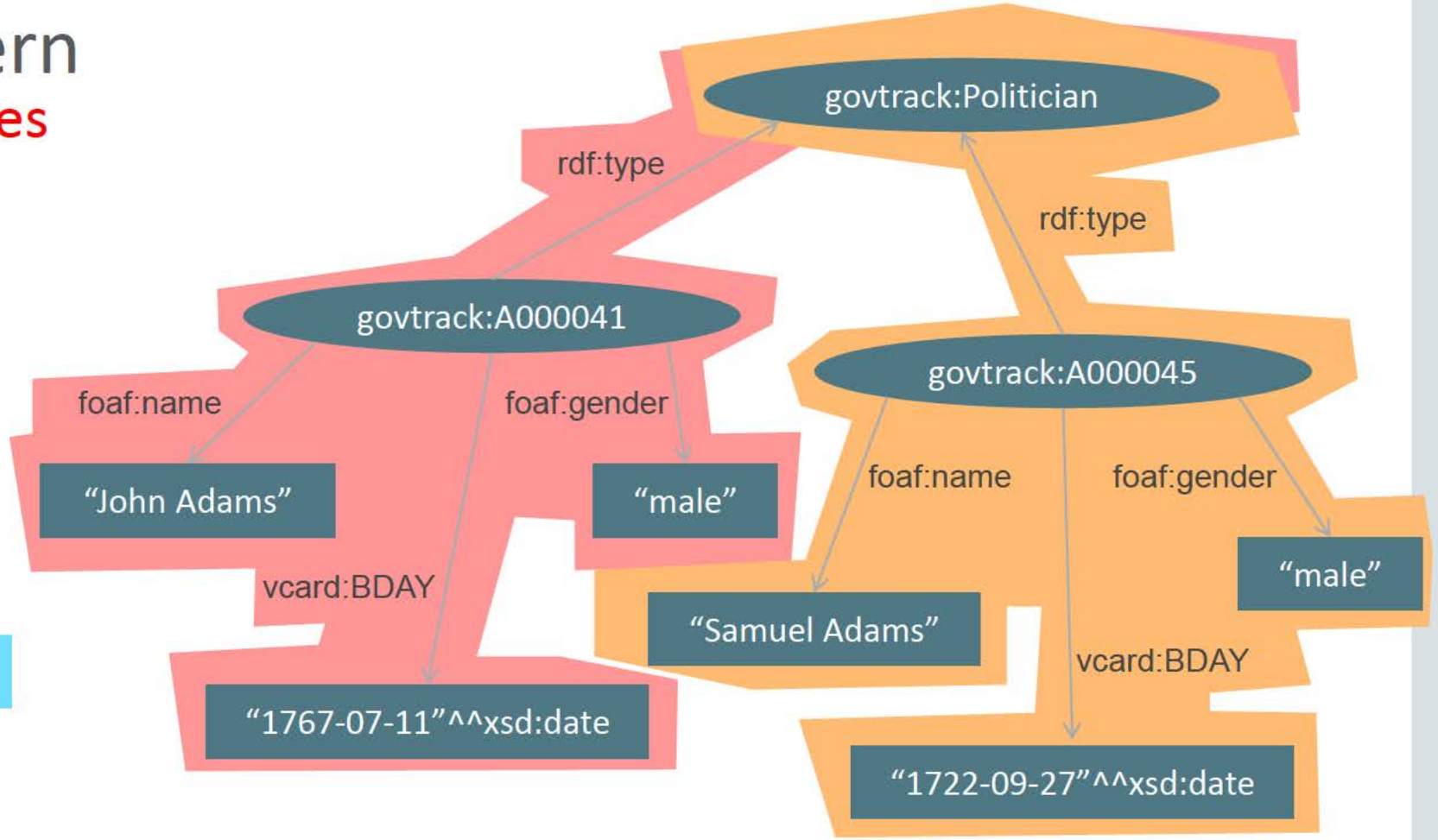


:hasOracleCSI rdf:type owl:FunctionalProperty

SPARQL Graph Pattern

Basic unit of SPARQL queries





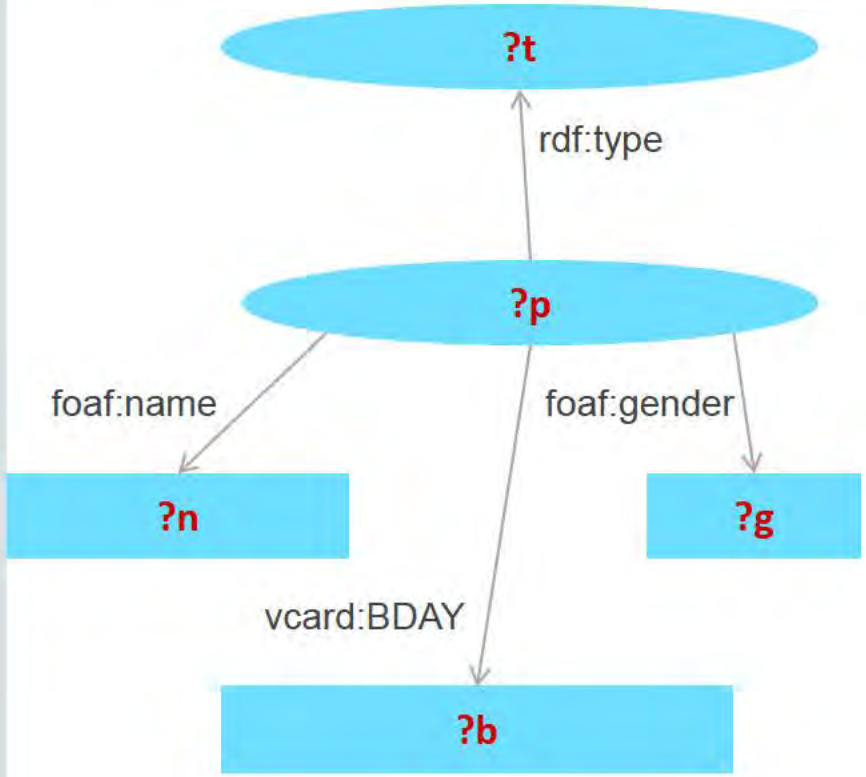
Result 1: {?t=govtrack:Politician, ?p=govtrack:A000041, ?n="John Adams", ?g="male", ?b="1767-07-11"^^xsd:date}

Result 2: {?t=govtrack:Politician, ?p=govtrack:A000045, ?n="Samuel Adams",?g="male", ?b="1722-09-27"^^xsd:date}

ORACLE

SPARQL Graph Pattern

Basic unit of SPARQL queries



How do we express this with SPARQL?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?t ?n ?b ?g
WHERE
{ ?p rdf:type ?t .
  ?p foaf:name ?n .
  ?p vcard:BDAY ?b .
  ?p foaf:gender ?g }
```

Basic Graph Pattern (BGP)

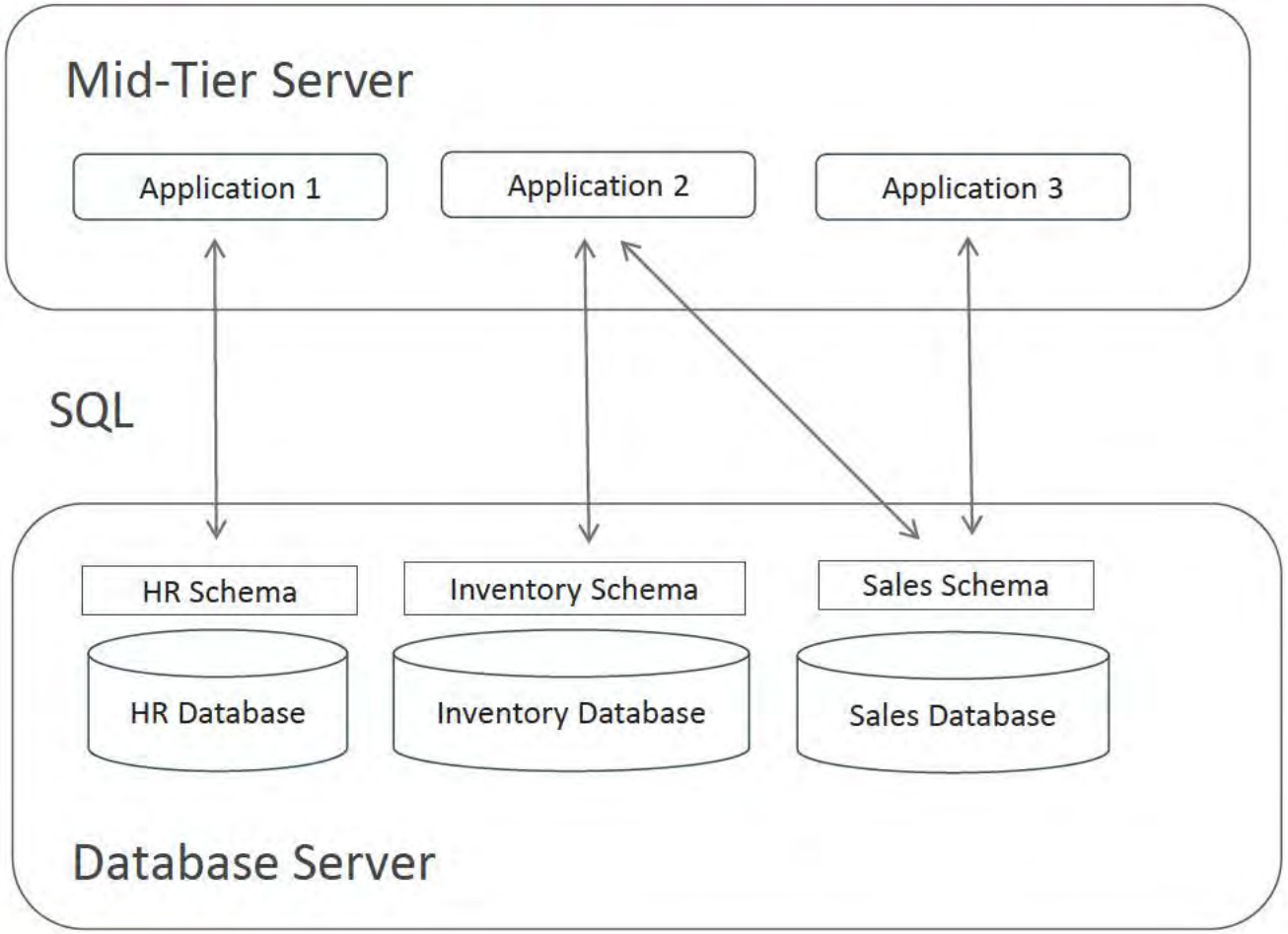
Agenda

1. Introduction and Motivation
2. Review of RDF/OWL/SPARQL
3. RDB2RDF Standard
4. GeoSPARQL Standard
5. Adding GeoSPARQL Support to RDB2RDF
6. Demo

ORACLE

A Look at Some System Architectures

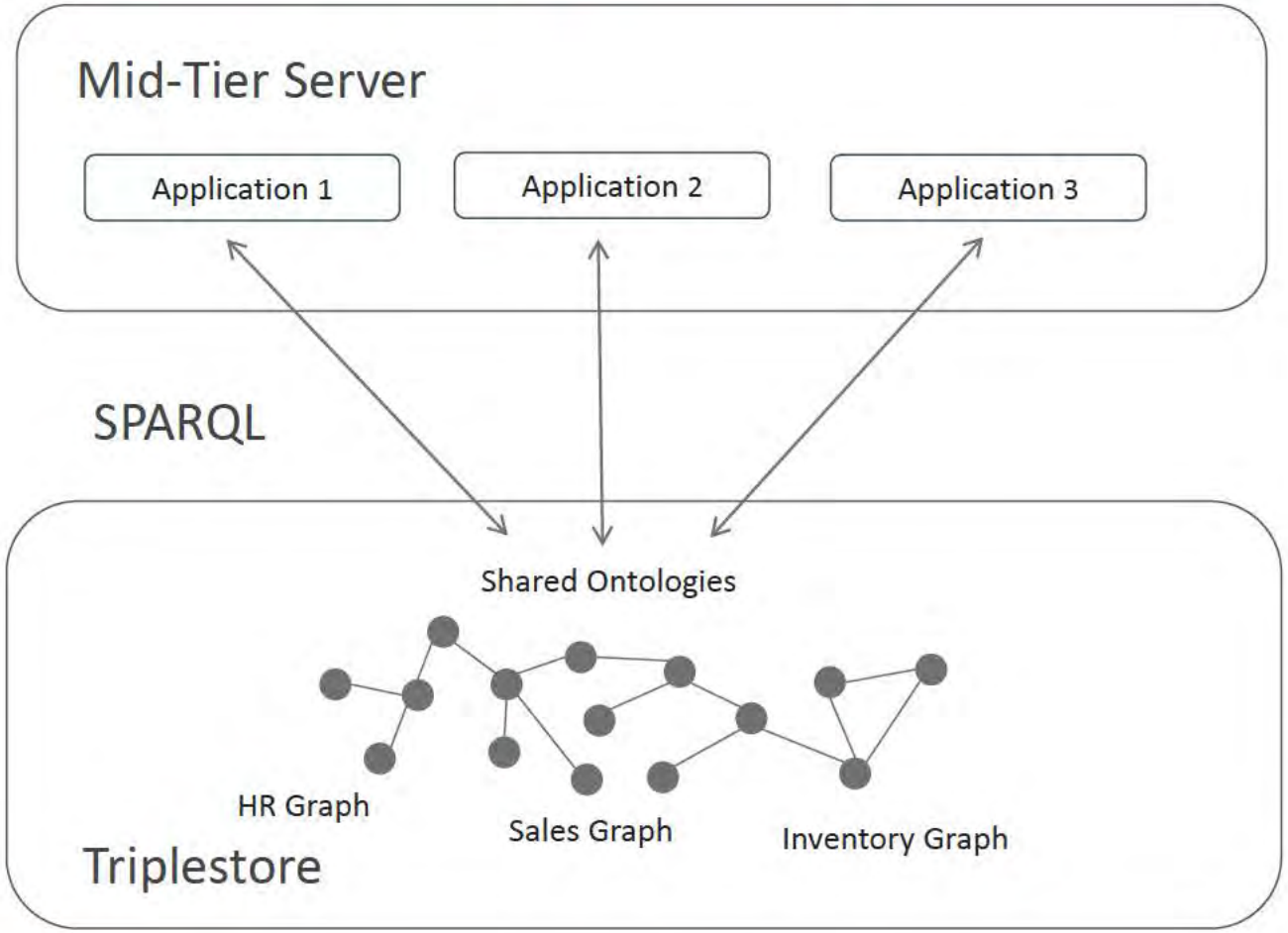
Typical Traditional Application Architecture



- Mid-tier server and database server
- Applications communicate via SQL/JDBC with RDBMS backend
- Multiple traditional relational schemas
- Issues
  - Inflexible schema
  - Limited semantics
  - Limited interoperability

A Look at Some System Architectures

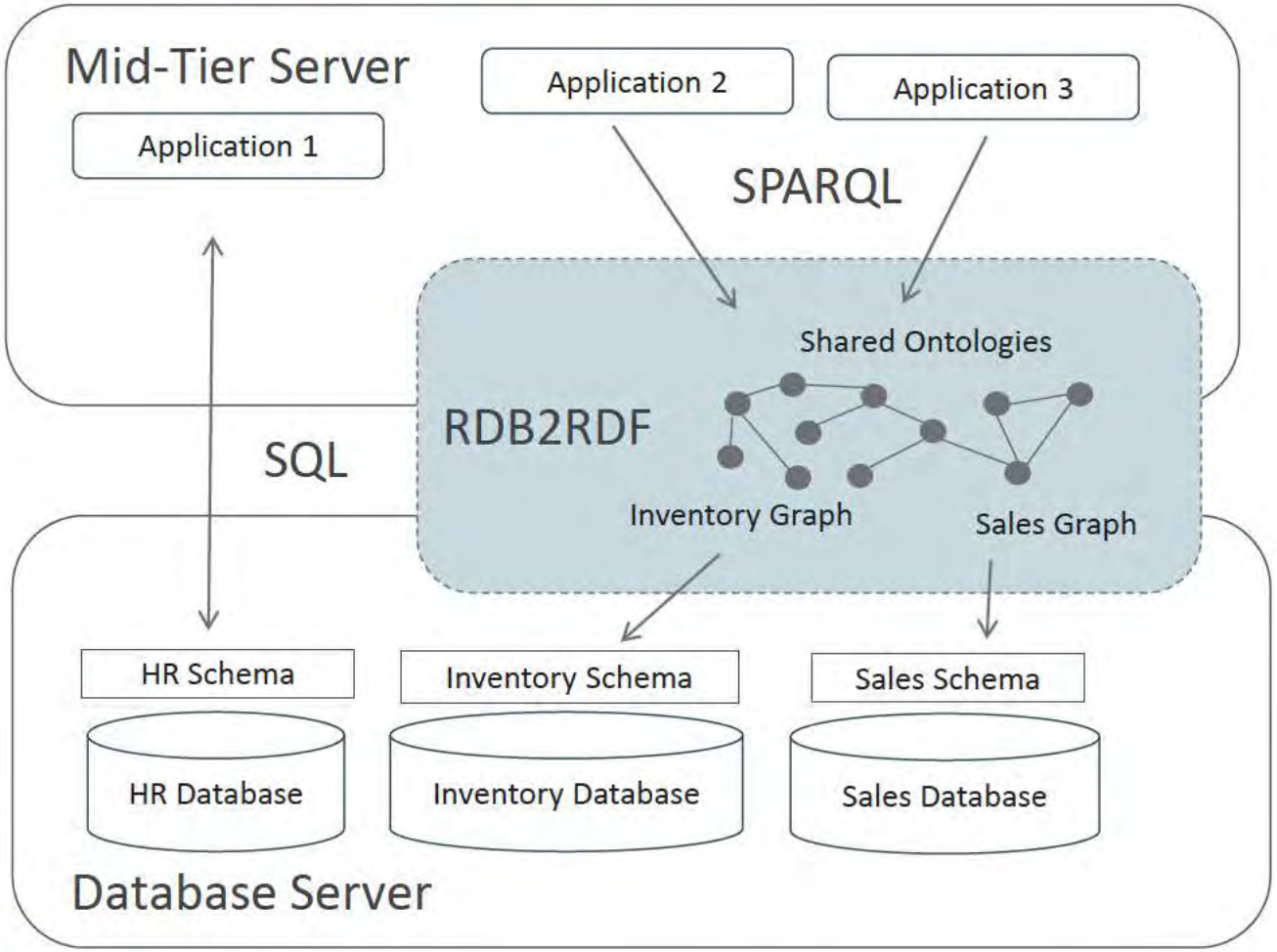
Typical Semantic Application Architecture



- Common ontologies used to integrate datasets
- Applications communicate via SPARQL / HTTP with native triplestore backend
- Issues
  - Radical change for customer
  - Need for ETL to RDF
  - RDF/OWL may not be necessary for the entire data

A Look at Some System Architectures

How about RDB2RDF?



- Use virtual RDF data
- Benefits
  - Existing relational data stays in place and corresponding applications do not need to change
  - Use of virtual mapping eliminates synchronization issues
  - Common vocabulary helps with data integration issues

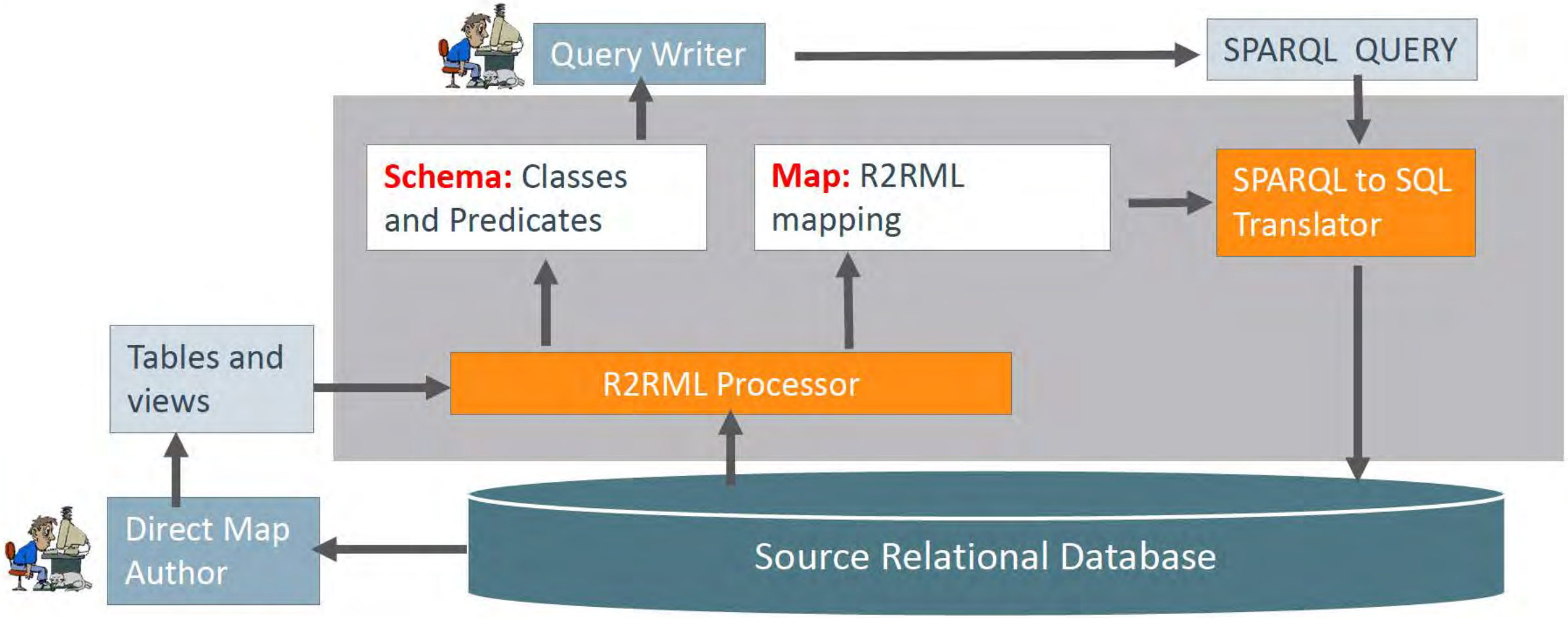
Relational Data to RDF (W3C RDB2RDF)

Two types of mapping: Direct and R2RML

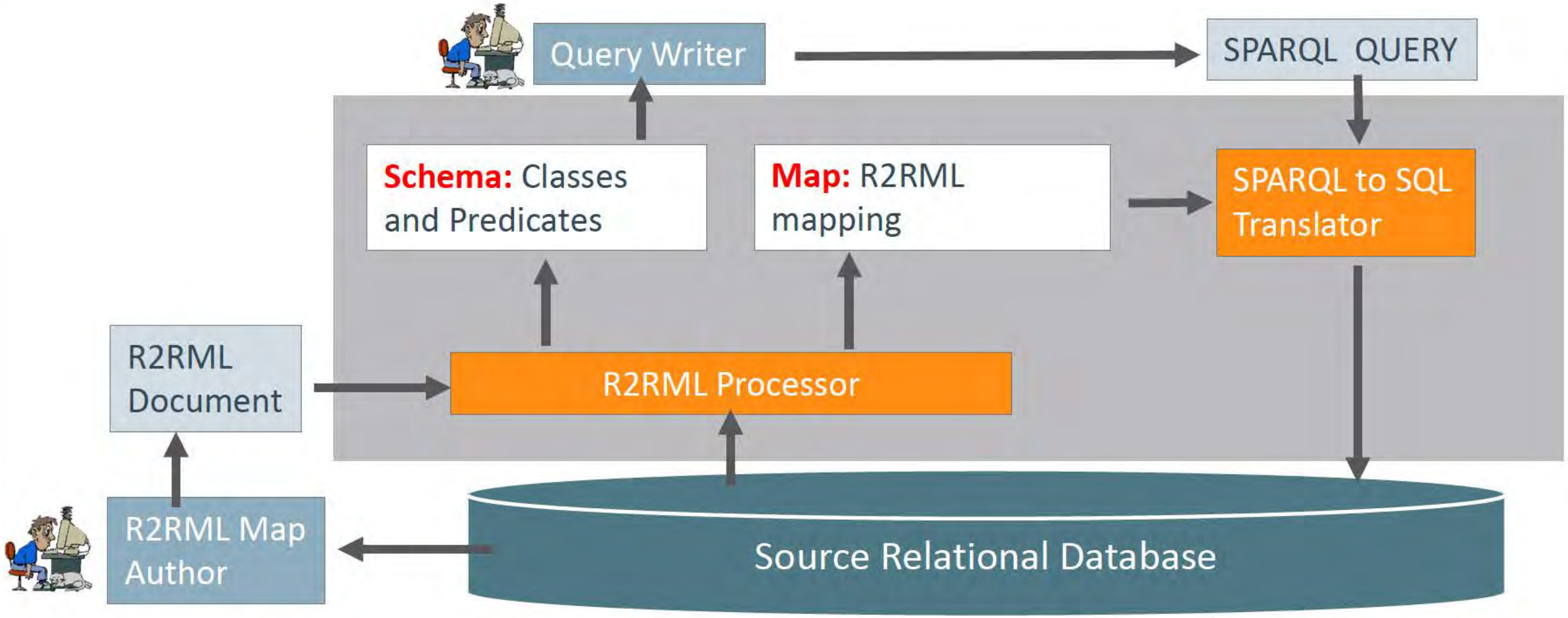
- Direct Mapping
  - Automatically generates a mapping based on an input relational schema
- R2RML (RDB to RDF Mapping Language)
  - Language for expressing customized mappings

ORACLE

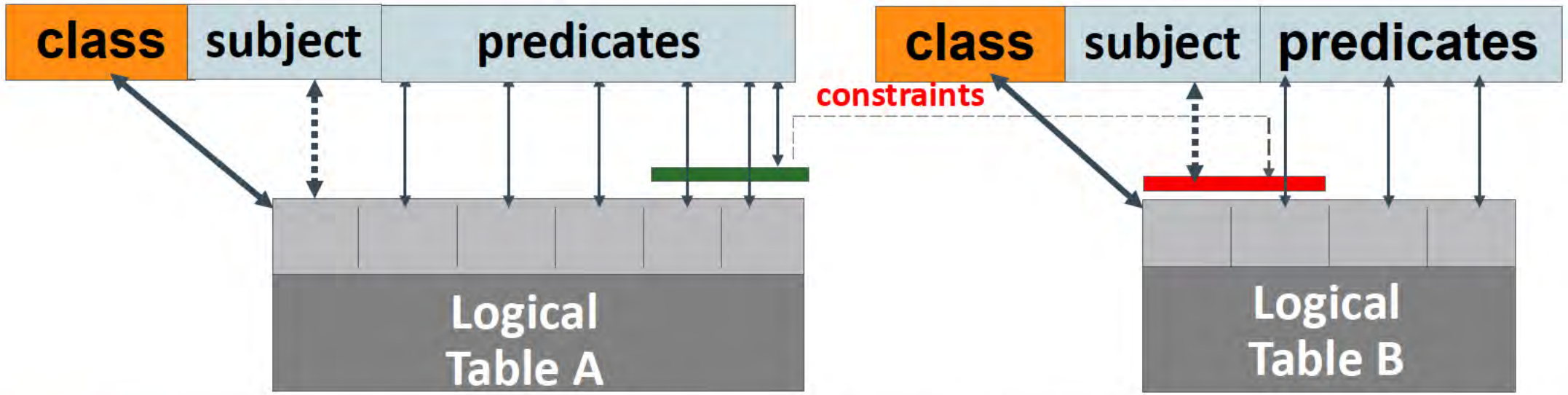
Using Direct Mapping: Overall Flow



Using R2RML: Overall Flow



R2RML Basics: Mapping

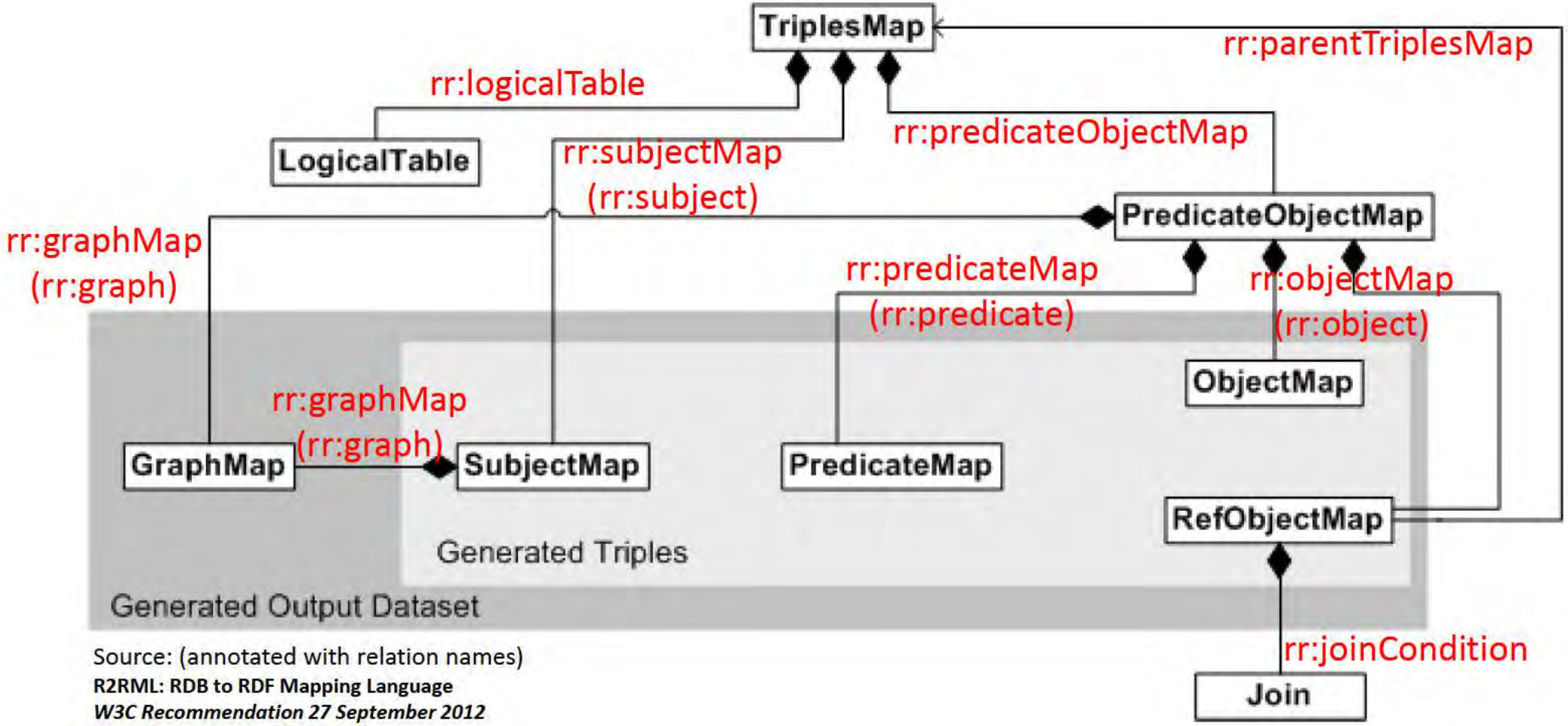


- **TriplesMap**: Row of a Logical Table (Table / View / SQL query) ➔ triples
  - SubjectMap: Primary key value of a Row ➔ subject (+ *static* class + GraphMap)

POMap

- PredicateMap: Names of columns and constraints ➔ predicates (incl. rdf:type)
- ObjectMap: Values in columns or foreign keys ➔ objects (+ class)
- GraphMap: Put this triple in a named graph

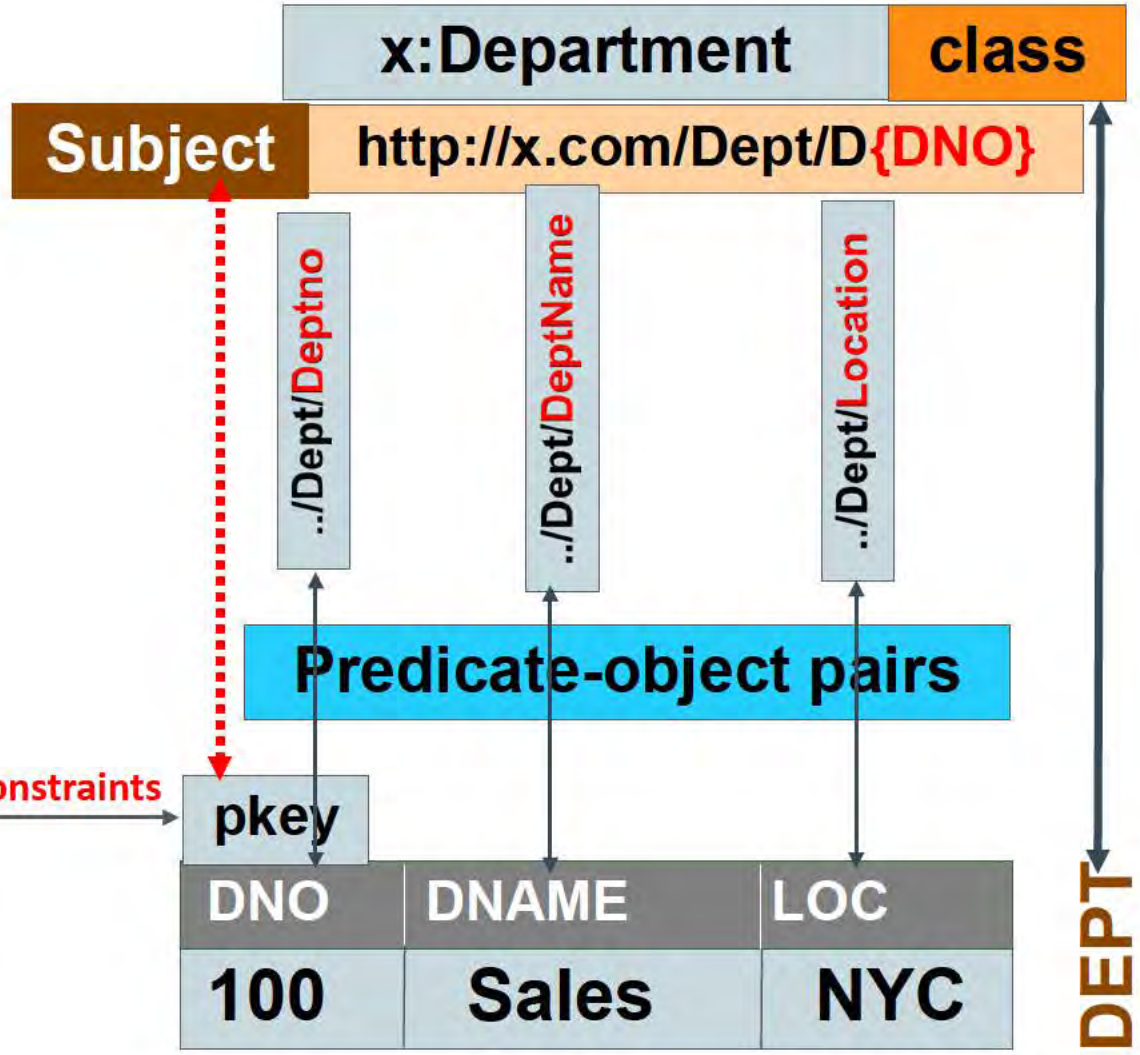
Vocabulary: R2RML Classes and Relations



Source: (annotated with relation names)
R2RML: RDB to RDF Mapping Language
W3C Recommendation 27 September 2012
http://www.w3.org/TR/r2rml/

Ex: Mapping EMP and DEPT Tables to RDF

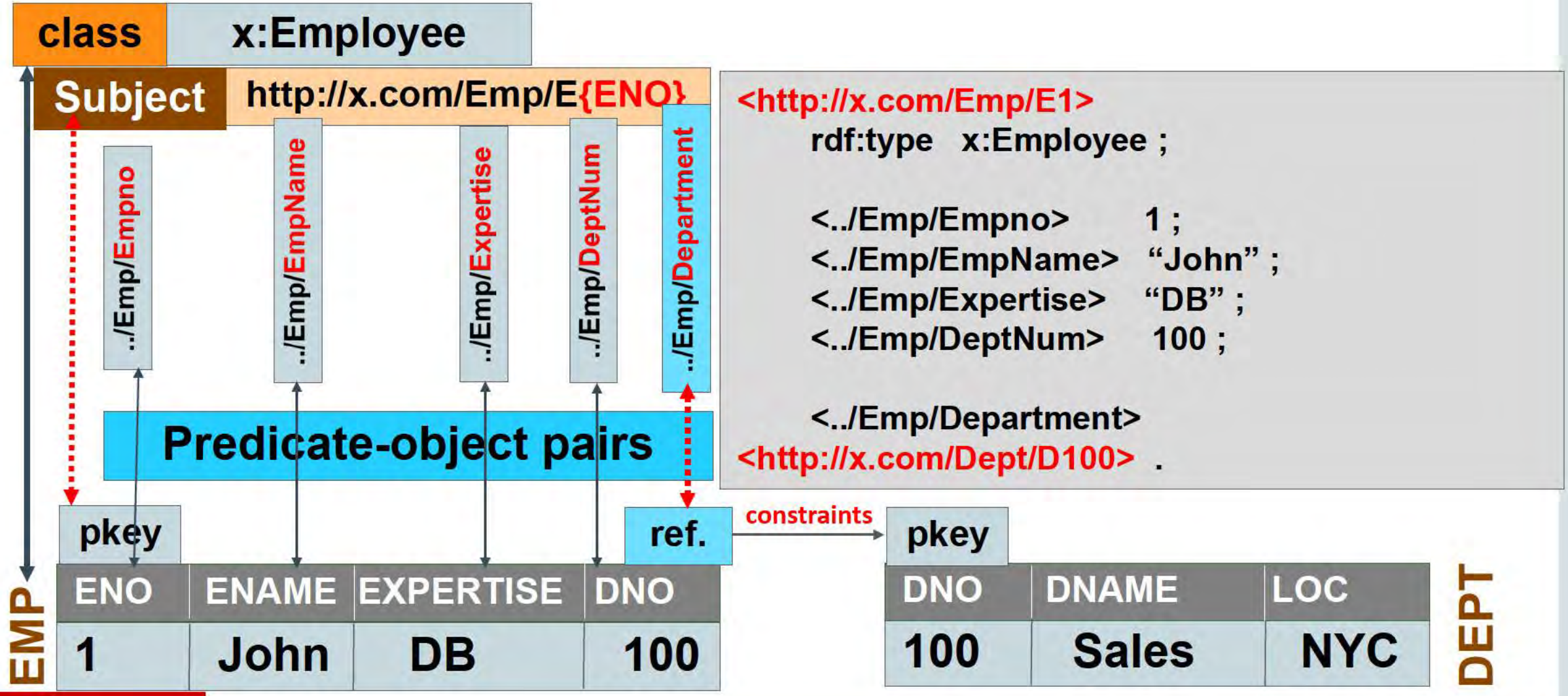
```
<http://x.com/Dept/D100>
      rdf:type   x:Department ;

      <../Dept/Deptno>        100 ;
      <../Dept/DeptName>   "Sales" ;
      <../Dept/Location>     "NYC"
.
```



EMP

pkey			ref.
ENO	ENAME	EXPERTISE	DNO
1	John	DB	100

Ex: Mapping EMP and DEPT Tables to RDF



Schema for Generated RDF data

- **Classes**
  - x:Department
  - x:Employee

- **Properties**
  - <../Dept/Deptno>
  - <../Dept/DeptName>
  - <../Dept/Location>
  - <../Emp/Empno>
  - <../Emp/EmpName>
  - <../Emp/Expertise>
  - <../Emp/DeptNum>
  - <../Emp/Department>

ORACLE

R2RML mapping

set of TriplesMaps

TriplesMap: … **rr:logicalTable: …**	**PredicateObjectMap**	**PredicateObjectMap**	**PredicateObjectMap**
SubjectMap: [rr:class …] [rr:graphMap …]	PredicateMap:	PredicateMap:	PredicateMap: I
	GraphMap:		GraphMap:
	ObjectMap:	ObjectMap:	ObjectMap: rr:parentTriplesMap rr:joinCondition(s)

TriplesMap: … **rr:logicalTable: …**	**PredicateObjectMap**
SubjectMap: [rr:class …] [rr:graphMap …]	PredicateMap:
	ObjectMap:

R2RML Mapping for EMP (w/ multi-ObjectMap)

Empno NUMBER	Ename Varchar	W_phone NUMBER	C_phone Varchar	H_phone Varchar	W_addr Varchar	H_addr Varchar	DeptNo NUMBER
1	JOHN	1112223333	1113334444	2225556666	NYC	NJ	100

Tmap → <#EmpTM>

```
rr:logicalTable [
  rr:tableName "EMP"
].
```

Smap → []

```
rr:template
  "http://ex.org/E/{EMPNO}" ;
rr:class ex:Employee .
```

POmap → []

```
rr:predicate em:phone ;
rr:objectMap
  [ rr:column "W_PHONE" ]
, [ rr:column "C_PHONE" ]
, [ rr:column "H_PHONE" ] .
```

POmap → []

```
rr:predicate em:dept ;
rr:objectMap [ rr:parentTriplesMap <#DeptTM> ;
  rr:joinCondition [ rr:child "DEPTNO" ; rr:parent "DEPTNO" ]].
```

RefObjectMap

ORACLE

R2RML Mapping: using “R2RML Views”

```
<#DeptTableView> rr:sqlQuery """
SELECT DEPTNO, DNAME, LOC
 , (SELECT COUNT(*) FROM EMP WHERE EMP.DEPTNO=DEPT.DEPTNO) AS STAFF
FROM DEPT """.

<#TriplesMap1>
  rr:logicalTable <#DeptTableView>;

  rr:subjectMap [ rr:template "http://data.example.com/department/{DEPTNO}";
                  rr:class ex:Department; ];

  rr:predicateObjectMap [ rr:predicate ex:name; rr:objectMap [ rr:column "DNAME" ]; ];
  rr:predicateObjectMap [ rr:predicate ex:location; rr:objectMap [ rr:column "LOC" ]; ];
  rr:predicateObjectMap [ rr:predicate ex:staff; rr:objectMap [ rr:column "STAFF" ]; ].
```

ORACLE

R2RML Mapping: Translating type codes to IRIs

R2RML Mapping

```
<#TriplesMap1>
  rr:logicalTable [ rr:sqlQuery """
SELECT *, (CASE JOB WHEN 'CLERK' THEN 'general-office'
                    WHEN 'NIGHTGUARD' THEN 'security'
                    WHEN 'ENGINEER' THEN 'engineering'
            END) ROLE
FROM EMP
""" ];
  rr:subjectMap [ rr:template "http://data.example.com/employee/{EMPNO}"; ];
  rr:predicateObjectMap [
        rr:predicate ex:role;
        rr:objectMap [ rr:template "http://data.example.com/roles/{ROLE}" ]; ].
```

Generated RDF triple

```
<http://data.example.com/employee/7369>  ex:role  <http://data.example.com/roles/general-office> .
```

ORACLE

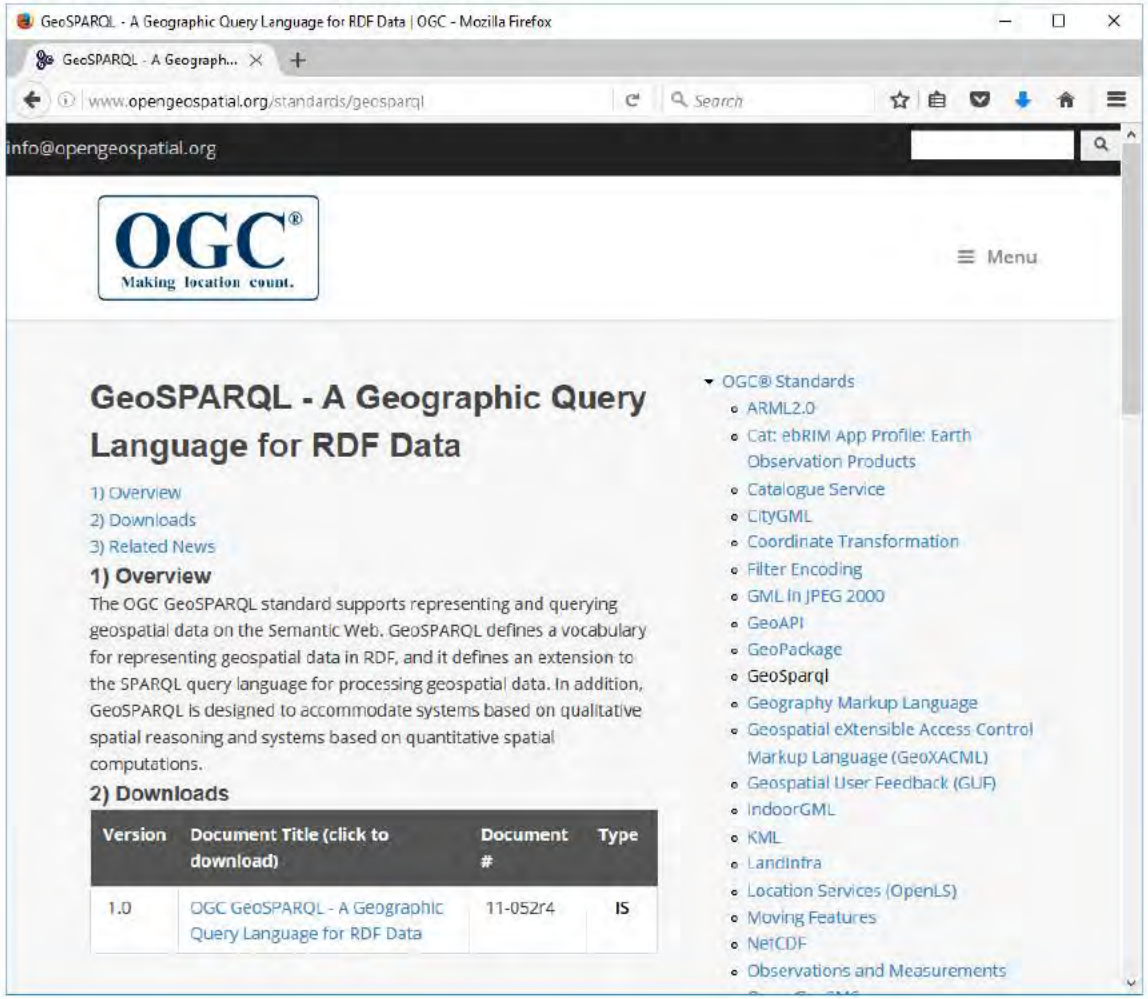
Agenda

1. Introduction and Motivation
2. Review of RDF/OWL/SPARQL
3. RDB2RDF Standard
4. GeoSPARQL Standard
5. Adding GeoSPARQL Support to RDB2RDF
6. Demo

ORACLE

OGC GeoSPARQL Standard

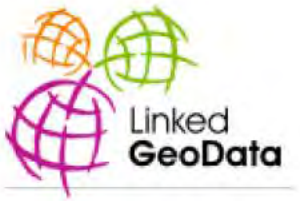
- Open Geospatial Consortium Standard
- Published in 2012
- Modular standard that defines several components:
  - RDF Vocabulary
  - Datatypes for geometry literals
  - SPARQL extension functions
  - Entailment rules



Why GeoSPARQL?

Linked Geospatial Data

- Many Linked Open Data (LOD) datasets have geospatial components
- Some GIS providers want to publish LOD
- Barriers to integration
  - Vendor-specific geometry support
  - Different vocabularies
    - W3C Basic Geo, GML XMLLiteral, Vendor-specific
  - Different spatial reference systems
    - WGS84 Lat-Long, British National Grid



Semantic GIS

- GIS applications with semantically complex thematic aspects
  - Logical reasoning to classify features
    - Land cover type, suitable farm land, etc.
  - Complex Geometries
    - Polygons and Multi-Polygons with 1000's of points
  - Complex Spatial Operations
    - Union, Intersection, Buffers, etc.

Find parcels with an **area** of at least 3 sq. miles that **touch** a local feeder road and are **inside** an area of suitable farm land.

ORACLE

Key Qualities of the GeoSPARQL Standard

- Provides a common target for implementers & users
  - Representation and query
- Works within SPARQL's extensibility framework
- Simple enough for general users
  - Keep the common case simple (WGS 84 point data)
- Capable enough for GIS professionals
  - Multiple SRS's, complex geometries, complex operators
- Leverages the wealth of existing spatial standards

International Organization for Standardization

ISO 19107 – Spatial Schema
ISO 13249 – SQL/MM

OGC Making Location Count

Simple Features
Well Known Text (WKT)
GML

From SPARQL to GeoSPARQL

SPARQL Query

RDF Data

```
:res1 rdf:type     :House .
:res1 :baths       "2.5"^^xsd:decimal .
:res1 :bedrooms    "3"^^xsd:decimal .

:res2 rdf:type     :Condo .
:res2 :baths       "2"^^xsd:decimal .
:res2 :bedrooms    "2"^^xsd:decimal .

:res3 rdf:type     :House
:res3 :baths       "1.5"^^xsd:decimal .
:res3 :bedrooms    "3"^^xsd:decimal .
```

SPARQL Query

```
SELECT ?r ?ba ?br
WHERE { ?r rdf:type :House .
        ?r :baths ?ba .
        ?r :bedrooms ?br }
```

Result Bindings

```
?r    | ?ba   | ?br
===================
:res1 | "2.5" | "3"
:res3 | "1.5" | "3"
```

ORACLE

SPARQL Query

RDF Data

```
:res1 rdf:type     :House .
:res1 :baths       "2.5"^^xsd:decimal .
:res1 :bedrooms    "3"^^xsd:decimal .

:res2 rdf:type     :Condo .
:res2 :baths       "2"^^xsd:decimal .
:res2 :bedrooms    "2"^^xsd:decimal .

:res3 rdf:type     :House
:res3 :baths       "1.5"^^xsd:decimal .
:res3 :bedrooms    "3"^^xsd:decimal .
```

SPARQL Query

```
SELECT ?r ?ba ?br
WHERE { ?r rdf:type :House .
        ?r :baths ?ba .
        ?r :bedrooms ?br
        FILTER (?ba > 2) }
```

Result Bindings

```
?r    | ?ba   | ?br
===================
:res1 | "2.5" | "3"
```

Spatial SPARQL QUERY

Spatial RDF Data

```
:res1   rdf:type          :House .
:res1   :baths            "2.5"^^xsd:decimal .
:res1   :bedrooms         "3"^^xsd:decimal .
:res1   ogc:hasGeometry   :geom1 .
:geom1  ogc:asWKT         "POINT(-122.25 37.46)"^^ogc:wktLiteral .

:res3   rdf:type          :House
:res3   :baths            "1.5"^^xsd:decimal .
:res3   :bedrooms         "3"^^xsd:decimal .
:res3   ogc:hasGeometry   :geom3 .
:geom3  ogc:asWKT         "POINT(-122.24 37.47)"^^ogc:wktLiteral .
```

This is what GeoSPARQL standardizes

Vocabulary & Datatypes

Find houses within a search polygon

GeoSPARQL Query

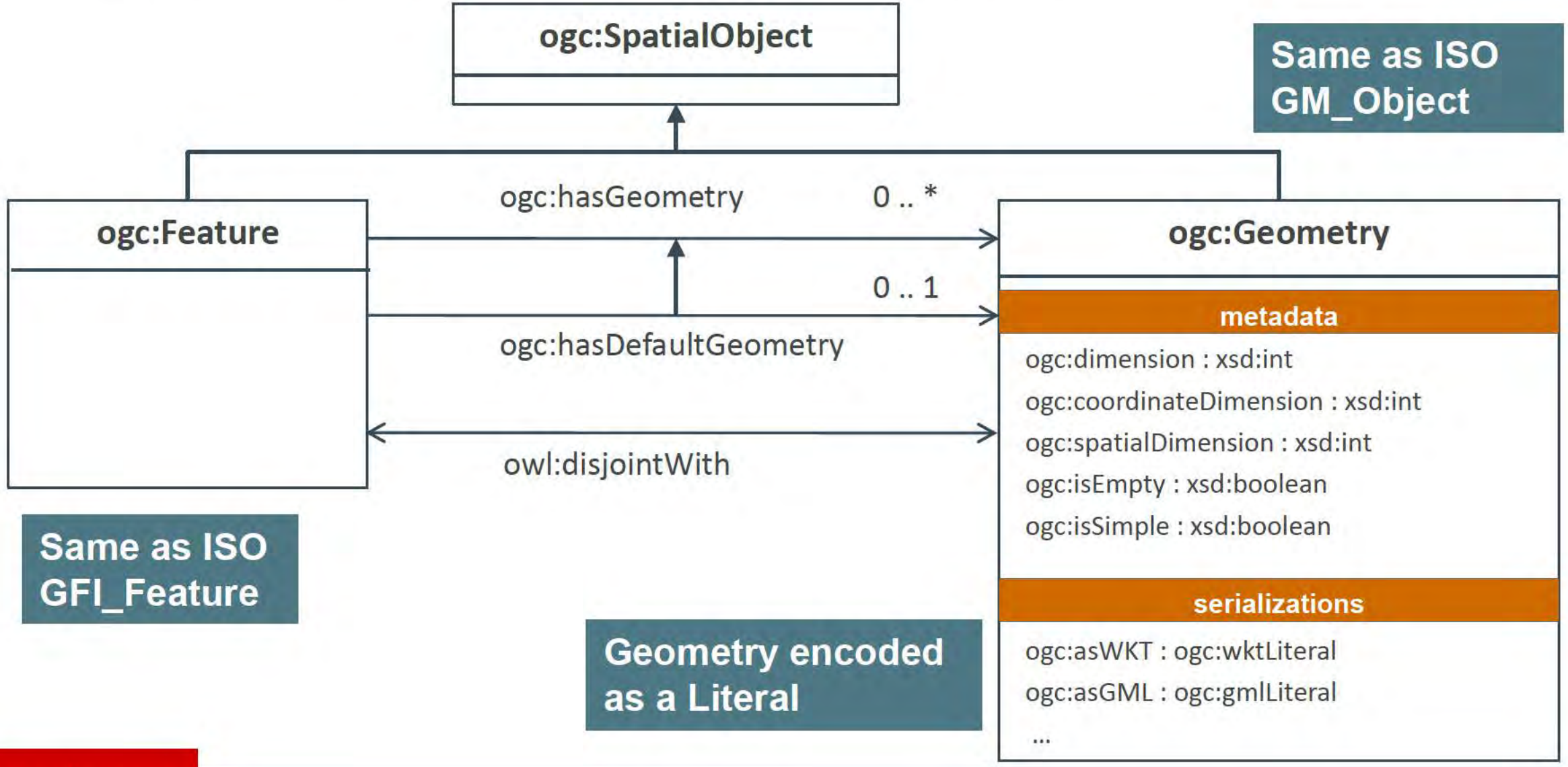
```
SELECT ?r ?ba ?br
WHERE { ?r rdf:type :House . ?r :baths ?ba . ?r :bedrooms ?br .
        ?r ogc:hasGeometry ?g . ?g ogc:asWKT ?wkt
        FILTER(ogcf:sfWithin(?wkt, "POLYGON(…)"^^ogc:wktLiteral))
}
```

Extension Functions

ORACLE

GeoSPARQL Vocabulary

GeoSPARQL Vocabulary: Basic Classes and Relations



Details of ogc:wktLiteral

All RDF Literals of type ogc:wktLiteral shall consist of an optional IRI identifying the spatial reference system followed by Simple Features Well Known Text (WKT) describing a geometric value [ISO 19125-1].

```
"<http://www.opengis.net/def/crs/OGC/1.3/CRS84>
 POINT(-122.4192 37.7793)"^^ogc:wktLiteral
```

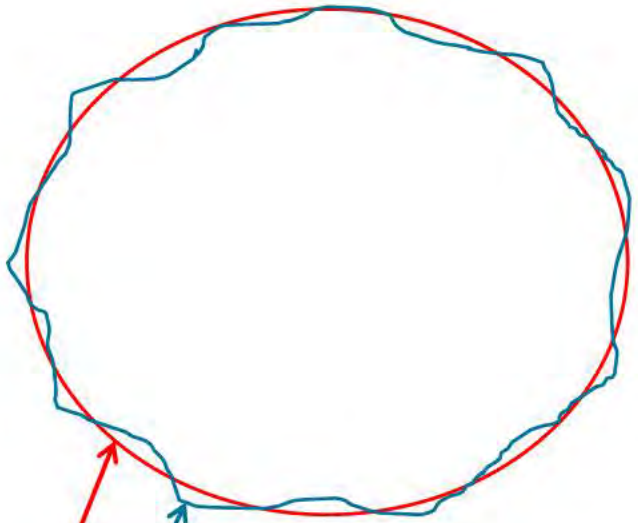
WGS84 longitude – latitude is the default CRS

```
"POINT(-122.4192 37.7793)"^^ogc:wktLiteral
```

European Petroleum Survey Group (EPSG) maintains a set of CRS identifiers.

ORACLE

More About Coordinate Reference Systems



A CRS is based on a specific datum.

Because Earth has a very irregular shape, different datums and corresponding CRSs provide more accurate fits for different locations.

Local CRSs are used for high precision applications.

The same location will have different coordinates in each CRS.

Geoid

close approximation of Earth's shape (mean sea level)

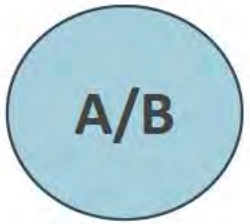
Reference Ellipsoid / Datum

mathematically defined approximate surface

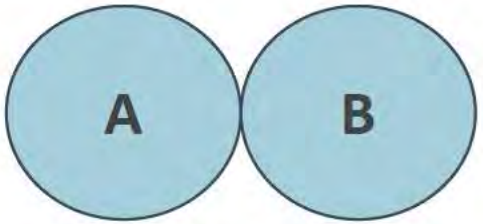
GeoSPARQL Default

WGS84 – Global (GIS, most web data, etc.)
ETRS89 – Europe
NAD83 – North America
GDA94 – Australia
OSGB36 – UK

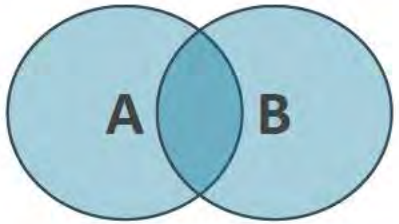
Topological Relations between ogc:SpatialObject



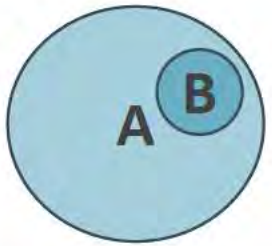
ogc:sfEquals



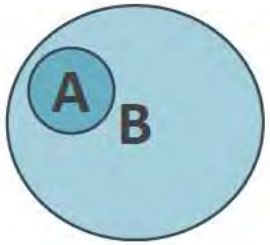
ogc:sfTouches



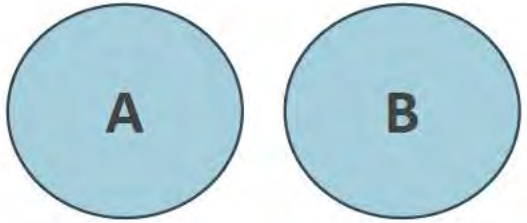
ogc:sfOverlaps



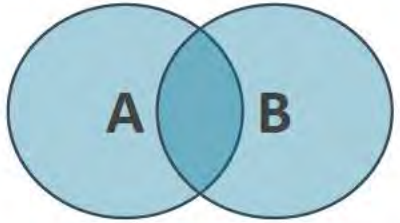
ogc:sfContains



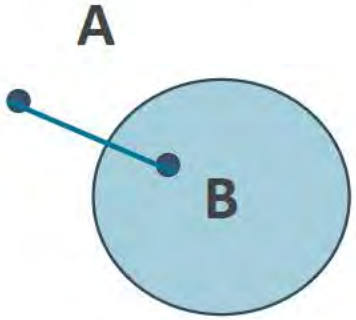
ogc:sfWithin



ogc:sfDisjoint



ogc:sfIntersects



ogc:sfCrosses

- rdfs:domain/range ogc:SpatialObject
- Assumes Simple Features Relation Family
- Also support Egenhofer and RCC8

Example Data

Schema Information

```
:City              rdfs:subClassOf     ogc:Feature .
:Park              rdfs:subClassOf     ogc:Feature .
:exactGeometry     rdfs:subPropertyOf  ogc:hasGeometry .
```

Non-spatial Properties

```
:SanFrancisco      rdf:type            :City .
:UnionSquarePark   rdf:type            :Park .
:UnionSquarePark   :commissioned       "1847-01-01"^^xsd:date .
```

Spatial Properties

```
:UnionSquarePark   :exactGeometry      :geo1 .
:geo1              ogc:asWKT           "Polygon((…))"^^ogc:wktLiteral .

:SanFrancisco      :exactGeometry      :geo2 .
:geo2              ogc:asWKT           "Polygon((…))"^^ogc:wktLiteral .

:UnionSquarePark   ogc:sfWithin        :SanFrancisco .
```

GeoSPARQL Query Functions

GeoSPARQL Query Functions

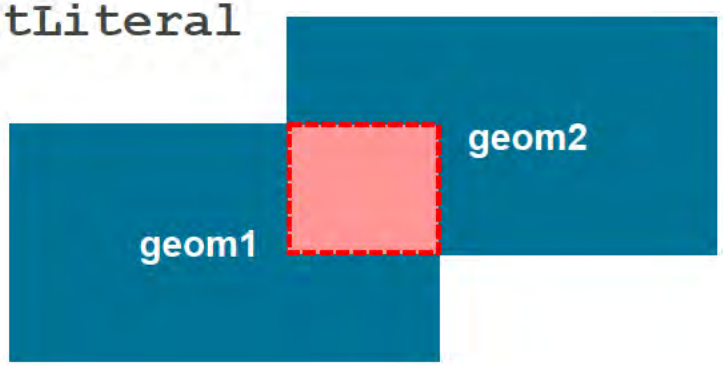
- `ogcf:distance(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral,`
  `units: xsd:anyURI): xsd:double`

- `ogcf:buffer(geom: ogc:wktLiteral,`
  `radius: xsd:double,`
  `units: xsd:anyURI): ogc:wktLiteral`

- `ogcf:convexHull(geom: ogc:wktLiteral): ogc:wktLiteral`

GeoSPARQL Query Functions

- `ogcf:intersection(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral): ogc:wktLiteral`

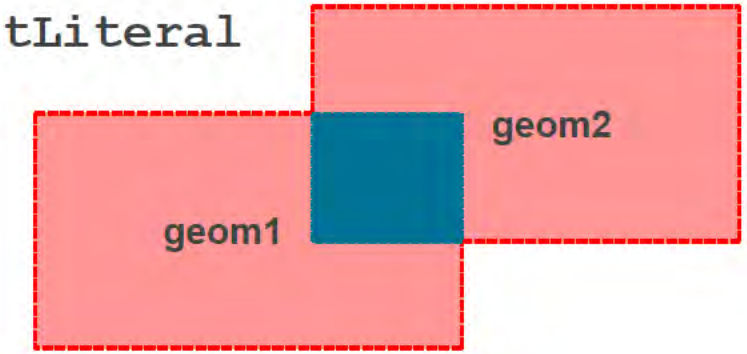


- `ogcf:union(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral): ogc:wktLiteral`

GeoSPARQL Query Functions

- `ogcf:difference(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral): ogc:wktLiteral`

- `ogcf:symDifference(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral): ogc:wktLiteral`



GeoSPARQL Query Functions

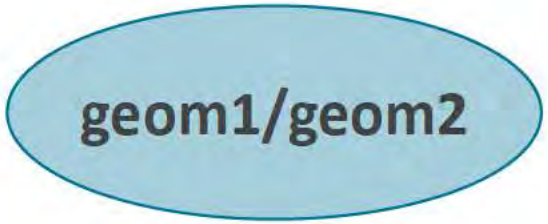
- `ogcf:envelope(geom: ogc:wktLiteral): ogc:wktLiteral`

- `ogcf:boundary(geom1: ogc:wktLiteral): ogc:wktLiteral`

- `ogcf:getSRID(geom: ogc:wktLiteral): xsd:anyURI`

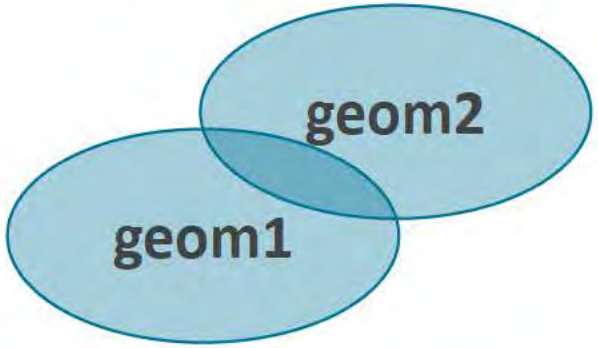
GeoSPARQL Topological Query Functions

- `ogcf:sfEquals(geom1: ogc:wktLiteral, geom2: ogc:wktLiteral): xsd:boolean`



- `ogcf:sfDisjoint(geom1: ogc:wktLiteral, geom2: ogc:wktLiteral): xsd:boolean`

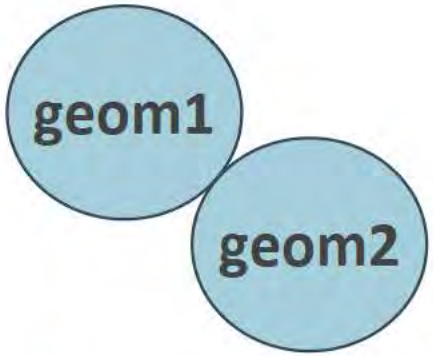
- `ogcf:sfIntersects(geom1: ogc:wktLiteral, geom2: ogc:wktLiteral): xsd:boolean`



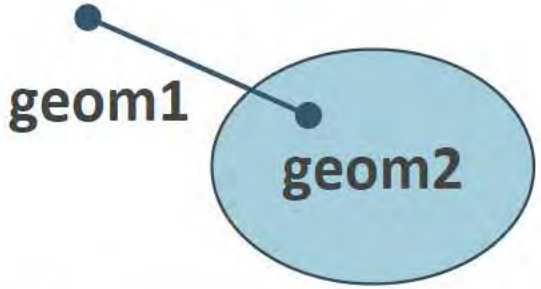
Assumes Simple Features Relation Family

GeoSPARQL Topological Query Functions

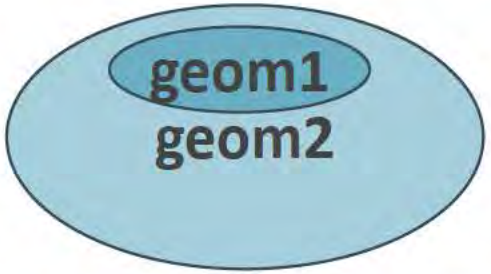
- `ogcf:sfTouches(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral): xsd:boolean`



- `ogcf:sfCrosses(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral): xsd:boolean`



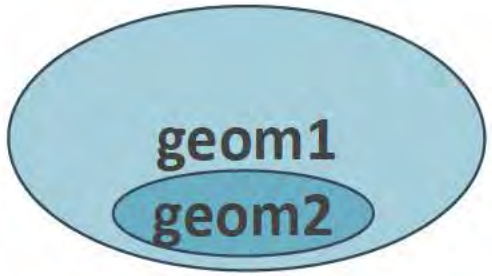
- `ogcf:sfWithin(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral): xsd:boolean`



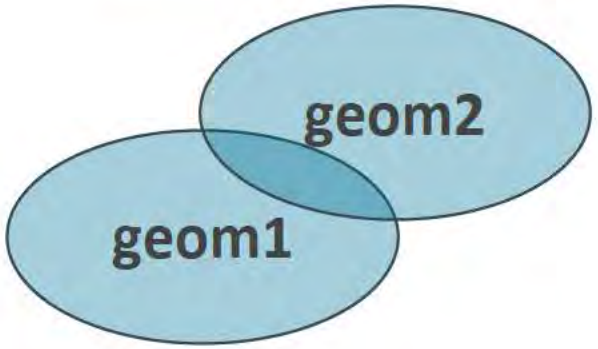
Assumes Simple Features Relation Family

GeoSPARQL Topological Query Functions

- `ogcf:sfContains(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral): xsd:boolean`



- `ogcf:sfOverlaps(geom1: ogc:wktLiteral,`
  `geom2: ogc:wktLiteral): xsd:boolean`

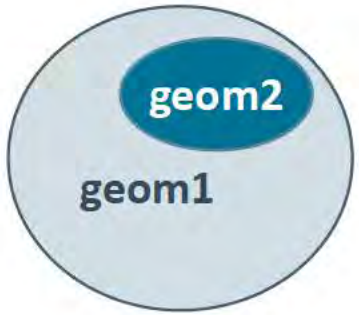


Assumes Simple Features Relation Family

GeoSPARQL Topological Query Functions

```
– ogcf:relate(geom1: ogc:wktLiteral,
              geom2: ogc:wktLiteral,
              patternMatrix: xsd:string): xsd:boolean
```

DE-9IM Intersection Matrix				
		geom2		
		Interior	Boundary	Exterior
geom1	Interior	T	T	T
	Boundary	F	F	T
	Exterior	F	F	T



ogc:sfContains

patternMatrix: TTTFFTFFT

Example Query 1

Linked GeoData

Find historic places within 10 KM of San Francisco, CA city center

```
PREFIX geovocab: <http://geovocab.org/geometry#>
PREFIX uom: <http://xmlns.oracle.com/rdf/geo/uom/>
SELECT ?s ?l ?wkt
WHERE {
 ?s rdfs:label ?l .
 ?s geovocab:geometry ?geom .
 ?geom ogc:asWKT ?wkt .
 FILTER(
  ogcf:distance(
     ?wkt,
     "POINT(-122.416667 37.783333)"^^ogc:wktLiteral,
     uom:KM) <= 10)
}
```

ORACLE

Example Query 2

Find Monuments within a query window

```
PREFIX geovocab: <http://geovocab.org/geometry#>
PREFIX lgd: <http://linkedgeodata.org/ontology/>
SELECT ?s ?l ?wkt
WHERE {
 ?s rdf:type lgd:Monument .
 ?s rdfs:label ?l .
 ?s geovocab:geometry ?geom .
 ?geom ogc:asWKT ?wkt .
 FILTER(
  ogcf:sfWithin(
    ?wkt,
    "POLYGON((-122.313056 37.554167,
              -122.026389 36.971944,
              -119.766667 36.75,
              -120.994444 37.661389,
              -122.313056 37.554167))"^^ogc:wktLiteral))
 }
```

Example Query 3

Find all land parcels that are within the intersection of :City1 and :District1

```
PREFIX : <http://my.com/appSchema#>
PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
PREFIX ogcf: <http://www.opengis.net/def/geosparql/functions/>
PREFIX epsg: <http://www.opengis.net/def/crs/EPSG/0/>

SELECT  ?parcel
WHERE { ?parcel      rdf:type          :Residential .
        ?parcel      :exactGeometry    ?pGeo .
        ?pGeo        ogc:asWKT         ?pWKT .
        :District1   :exactGeometry    ?dGeo .
        ?dGeo        ogc:asWKT         ?dWKT .
        :City1       :extent           ?cGeo .
        ?cGeo        ogc:asWKT         ?cWKT .
        FILTER(ogcf:sfWithin(?pWKT, ogcf:intersection(?dWKT,?cWKT)))}
```

Summary

- GeoSPARQL Defines:
  - Basic vocabulary, Query functions, Entailment component
- Based on existing OGC/ISO standards
  - WKT, GML, Simple Features, ISO 19107
- Uses SPARQL's built-in extensibility framework
- Modular specification
  - Allows flexibility in implementations
  - Easy to extend

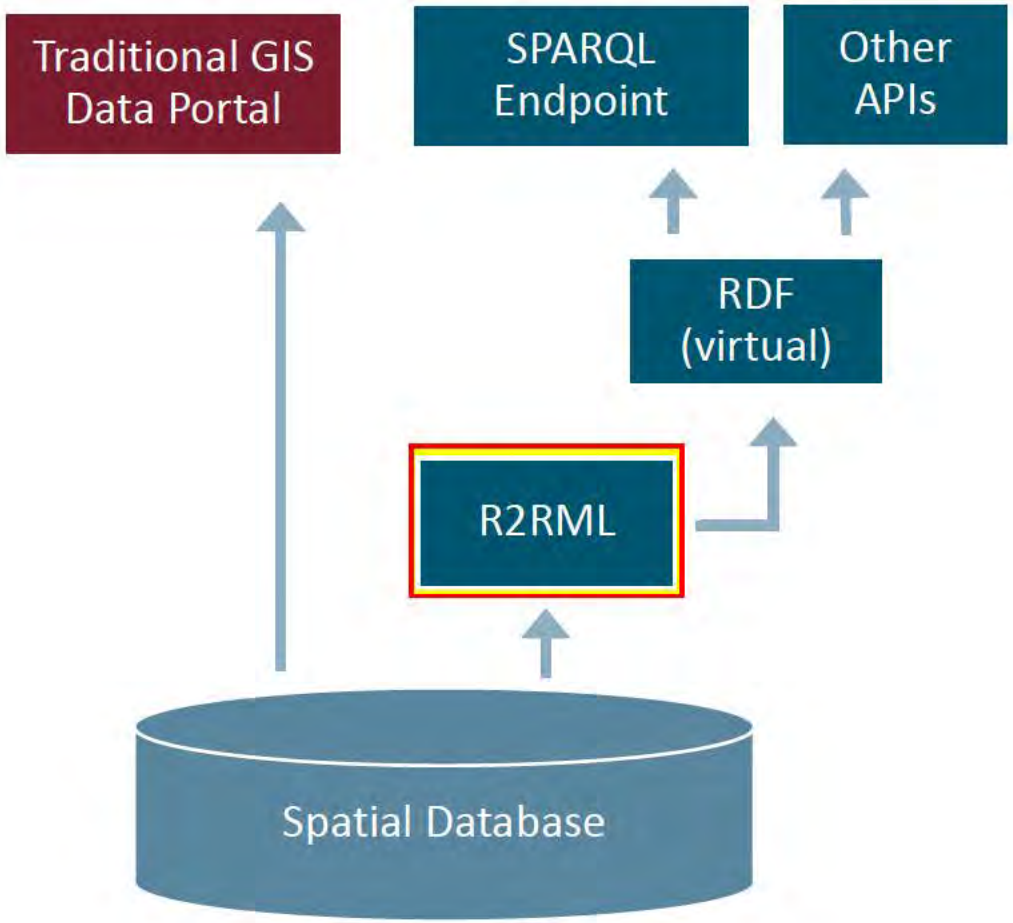
ORACLE

Agenda

1. Introduction and Motivation
2. Review of RDF/OWL/SPARQL
3. RDB2RDF Standard
4. GeoSPARQL Standard
5. Adding GeoSPARQL Support to RDB2RDF
6. Demo

ORACLE

R2RML mapping of Relational Table with Spatial Data



Relational Data

for table with a single GeoSpatial column

```
LocGeom (
LOC            NOT NULL     VARCHAR2(20)
GEOM_MAIN                   SDO_GEOMETRY
)
```

Primary key (LOC)

LOC	GEOM_MAIN
NEW YORK	SDO_GEOMETRY('POINT(-74.005897521973 40.712699890137)',8307)
DALLAS	SDO_GEOMETRY('POINT(-96.796669006348 32.775833129883)',8307)
CHICAGO	SDO_GEOMETRY('POINT(-87.684722900391 41.836944580078)',8307)
BOSTON	SDO_GEOMETRY('POINT(-71.063613891602 42.358055114746)',8307)

R2RML mapping

for table with a single GeoSpatial column

TriplesMap: LocGeom_IRI **rr:logicalTable: RDFUSER.LOCGEOM**	**PredicateObjectMap**	**PredicateObjectMap**
SubjectMap: "http://ex/LOC/{LOC}"	PredicateMap: ex:location	PredicateMap: ex: hasGeomMain_IRI
	ObjectMap: rr:column "LOC"	ObjectMap: rr:parentTriplesMap: **LocGeom_MAIN** rr:joinCondition: child.LOC = parent.LOC

TriplesMap: LocGeom_MAIN **rr:logicalTable: RDFUSER.LOCGEOM**	**PredicateObjectMap**
SubjectMap: "http://ex/GEOM_MAIN/{LOC}"	PredicateMap: ogc:asWKT
	ObjectMap: rr:column "GEOM_MAIN"

Content of RDF View using R2RML mapping

for table with a single GeoSpatial column

Spatial RDF Data

```
loc:NEW%20YORK          ex:location          "NEW YORK" ;
                        ex:hasGeomMain_IRI   geomMain:NEW%20YORK .
geomMain:NEW%20YORK     ogc:asWKT            "POINT(…)"^^ogc:wktLiteral .

loc:DALLAS              ex:location          "DALLAS" ;
                        ex:hasGeomMain_IRI   geomMain:DALLAS .
geomMain:DALLAS         ogc:asWKT            "POINT(…)"^^ogc:wktLiteral .

loc:CHICAGO             ex:location          "CHICAGO" ;
                        ex:hasGeomMain_IRI   geomMain:CHICAGO .
geomMain:CHICAGO        ogc:asWKT            "POINT(…)"^^ogc:wktLiteral .

loc:BOSTON              ex:location          "BOSTON" ;
                        ex:hasGeomMain_IRI   geomMain:BOSTON .
geomMain:BOSTON         ogc:asWKT            "POINT(…)"^^ogc:wktLiteral .
```

@prefix loc: <http:/ex/LOC/> . @ prefix geomMain <http://ex/GEOM_MAIN/>

ORACLE

GeoSPARQL query against RDF View

for table with a single GeoSpatial column

```
PREFIX ex:  <http://ex/>
PREFIX uom: <http://xmlns.oracle.com/rdf/geo/uom/>
select ?loc1 ?loc2 (ogcf:distance(?geom1,?geom2, uom:KM) as ?dist_km)
where {
 ?l1 ex:location ?loc1
   ; ex:hasGeomMain_IRI ?giri1 .
 ?giri1 ogc:asWKT ?geom1 .
 ?l2 ex:location ?loc2
   ; ex:hasGeomMain_IRI ?giri2 .
 ?giri2 ogc:asWKT ?geom2 .
FILTER(?loc1 > ?loc2)
}
```

Find distance (In KM) between each pair of locations

GeoSPARQL query against RDF View

for table with a single GeoSpatial column

```
PREFIX ex:  <http://ex/>
select ?loc ?geom_main
where {
 ?l ex:location ?loc
  ; ex:hasGeomMain_IRI ?giri_main .
 ?giri_main ogc:asWKT ?geom_main .
FILTER(ogcf:sfWithin(?geom_main,
"POLYGON((-75.0 43.0, -75.0 40.0, -70.0 40.0, -70.0 43.0, -75.0 43.0))"^^ogc:wktLiteral))
}
```

Find locations that are within a specified polygon

Relational Data

For table with multiple GeoSpatial columns

```
LocGeom (
LOC            NOT NULL    VARCHAR2(20)
GEOM_MAIN                  SDO_GEOMETRY
GEOM_SOUTH                 SDO_GEOMETRY
)
```

Primary key (LOC)

LOC	GEOM_MAIN	GEOM_SOUTH
NEW YORK	SDO_GEOMETRY('POINT(-74.005897521973 40.712699890137)',8307)	SDO_GEOMETRY('POINT(-74.247863 40.496377)',8307)
DALLAS	SDO_GEOMETRY('POINT(-96.796669006348 32.775833129883)',8307)	SDO_GEOMETRY('POINT(-96.949015 32.621286)',8307)
CHICAGO	SDO_GEOMETRY('POINT(-87.684722900391 41.836944580078)',8307)	SDO_GEOMETRY('POINT(-87.615434 41.645612)',8307)
BOSTON	SDO_GEOMETRY('POINT(-71.063613891602 42.358055114746)',8307)	SDO_GEOMETRY('POINT(-71.131317 42.229262)',8307)

R2RML mapping

for table with multiple GeoSpatial columns

TriplesMap: LocGeom_IRI **rr:logicalTable: RDFUSER.LOCGEOM**	**PredObjMap**	**PredObjMap**	**PredicateObjectMap**
SubjectMap: "http://ex/LOC/{LOC}"	PredicateMap: ex:location	PredicateMap: ex:geomMain_IRI	PredicateMap: ex:hasGeomSouth_IRI
	ObjectMap: ...	ObjectMap: ...	ObjectMap: rr:parentTriplesMap **LocGeom_SOUTH** rr:joinCondition: child.LOC = parent.LOC

New PredicateObjecMap

TriplesMap: LocGeom_MAIN **rr:logicalTable: RDFUSER.LOCGEOM**	**PredicateObjectMap**
...	...

TriplesMap: LocGeom_SOUTH **rr:logicalTable: RDFUSER.LOCGEOM**	**PredicateObjectMap**
SubjectMap: "http://ex/GEOM_SOUTH/{LOC}"	PredicateMap: ogc:asWKT
	ObjectMap: rr:column “GEOM_SOUTH”

New TriplesMap

Content of RDF View using R2RML mapping

for table with multiple GeoSpatial columns

Spatial RDF Data

```
loc:NEW%20YORK          …
                        ex:hasGeomSouth_IRI geomSouth:NEW%20YORK .
geomSouth:NEW%20YORK    ogc:asWKT           "POINT(…)"^^ogc:wktLiteral .

loc:DALLAS              …
                        ex:hasGeomSouth_IRI geomSouth:DALLAS .
geomSouth:DALLAS        ogc:asWKT           "POINT(…)"^^ogc:wktLiteral .

loc:CHICAGO             …
                        ex:hasGeomSouth_IRI geomSouth:CHICAGO .
geomSouth:CHICAGO       ogc:asWKT           "POINT(…)"^^ogc:wktLiteral .

loc:BOSTON              …
                        ex:hasGeomSouth_IRI geomMain:BOSTON .
geomSouth:BOSTON        ogc:asWKT           "POINT(…)"^^ogc:wktLiteral .
```

@prefix loc: <http:/ex/LOC/> . @ prefix geomSouth <http://ex/GEOM_SOUTH/>

GeoSPARQL query against RDF View

for table with multiple GeoSpatial columns

```
PREFIX ex:  <http://ex/>
PREFIX uom: <http://xmlns.oracle.com/rdf/geo/uom/>
select ?loc (ogcf:distance(?geom_main,?geom_south, uom:KM) as ?dist_km)
where {
 ?l ex:location ?loc
  ; ex:hasGeomMain_IRI ?giri_main
  ; ex:hasGeomSouth_IRI ?giri_south .
 ?giri_main  ogc:asWKT ?geom_main .
 ?giri_south ogc:asWKT ?geom_south .
}
```

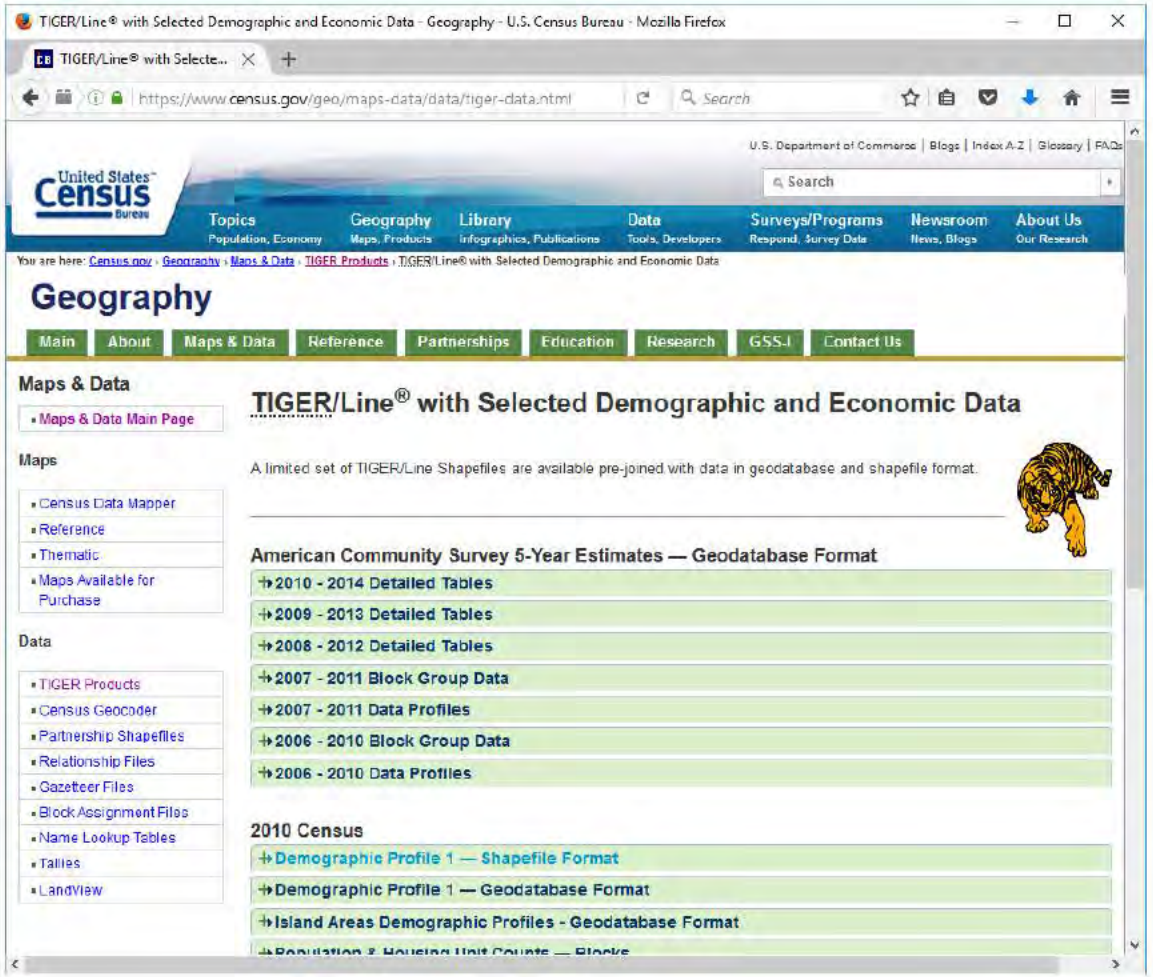
Find distance (In KM) between main and southernmost points of each city

Agenda

1. Introduction and Motivation
2. Review of RDF/OWL/SPARQL
3. RDB2RDF Standard
4. GeoSPARQL Standard
5. Adding GeoSPARQL Support to RDB2RDF
6. Demo

ORACLE

Exposing GIS Data as Linked Data



Starting Point:

County geometries with demographic and economic data downloaded from the US Census Bureau in ESRI shapefile format

Goal:

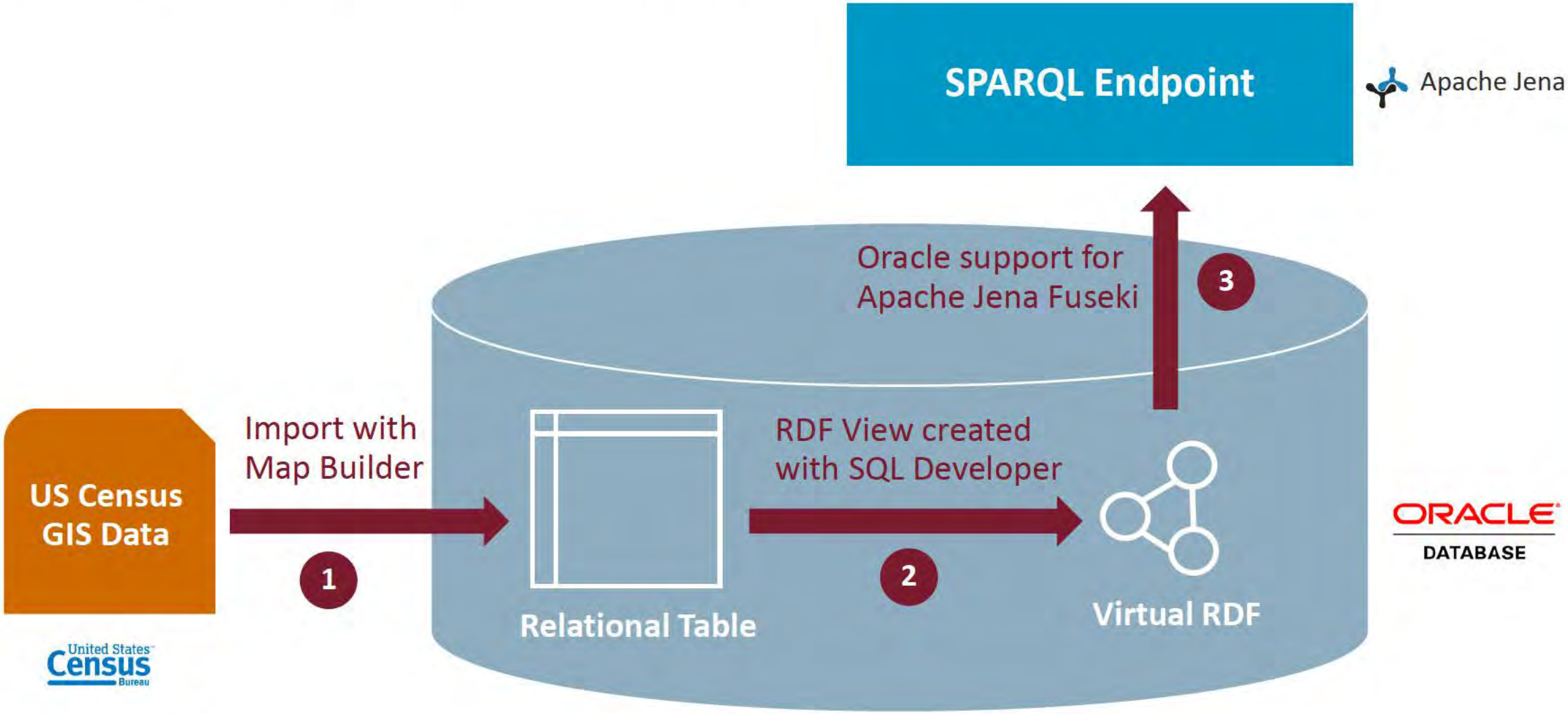
Create a SPARQL endpoint that exposes this data as Linked Data.

Exposing GIS Data as Linked Data

- Software Used
  - Big Data Lite 4.6 Virtual Machine Image
    - Oracle Database 12.1.0.2.0 Enterprise Edition with Spatial and Graph option (Plus patch 25114114)
  - SQL Developer 4.2 Early Adopter
  - Oracle Map Builder 12.2.1.2
  - Oracle Support for Apache Jena 2.11.2 and Fuseki 1.0.1

ORACLE

Exposing GIS Data as Linked Data



Integrated Cloud

Applications & Platform Services

ORACLE

ORACLE®