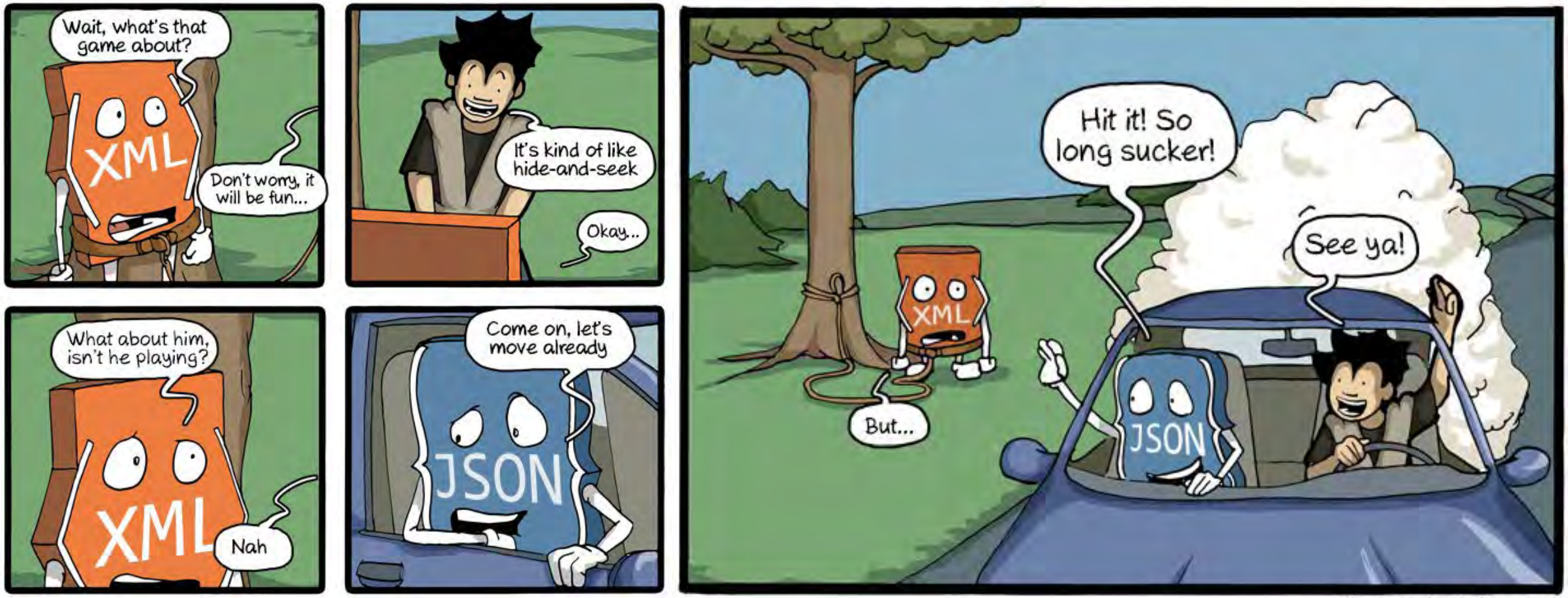


CommitStrip.com

GeoJSON and the Oracle Database

Albert Godfrind, Spatial Solutions Architect, Oracle

GeoJSON and Oracle

1. Why JSON ?
2. Why GeoJSON ?
3. Geometries, Features, Feature Collections
4. Publishing
5. Ingesting
6. Indexing and Querying
7. REST

JSON: The Fat-Free Alternative to XML

</>

```
<customer>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <street>2100 42nd Street</street>
    <city> New York </city>
    <state> NY </state>
    <postalCode> 10021 </postalCode>
    <isBusiness> false </isBusiness>
  </address>
  <phoneNumbers>
    <number type="home"> 212 555-1234 </number>
    <number type="cell"> 646 555-4567 </number>
  </phoneNumbers>
</customer>
```

{ }

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "Address": "2100 42nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021",
    "isBusiness" : false
  },
  "phoneNumbers": [
    {"type": "home", "number": "212 555-1234"},
    {"type": "cell", "number": "646 555-4567"}
  ]
}
```

JSON and XML – So Close, Yet so Different

JSON is Like XML Because …

- Both JSON and XML are **self describing** (human readable)
- Both JSON and XML are **hierarchical** (values within values)
- Both JSON and XML can be **parsed** and used by lots of programming languages
- Both JSON and XML can be **fetched** with an XMLHttpRequest

JSON is Unlike XML Because …

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays
- JSON is **schema-less**

Why JSON is Better than XML …

XML has to be parsed with an XML parser.
JSON can be parsed by a standard JavaScript function

Using XML

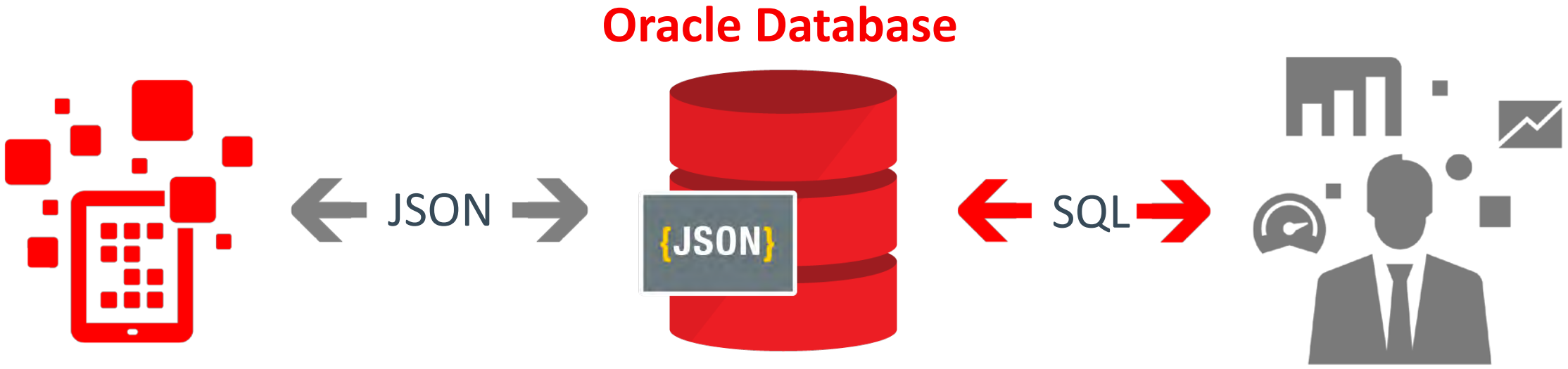
- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables
- Use XMLSCHEMA

Using JSON

- Fetch a JSON string
- JSON.Parse the JSON string into a JavaScript object.
- No schema
- Parse-less access in database

JSON Support in Oracle Database 12c

Flexible Application Development + Powerful SQL Analytics



Data accessed via RESTful service or native API's

Data persisted in database In JSON

Data analyzed via SQL

Storing and Querying JSON

Application developers: Store JSON using RESTful API

```
PUT /my_database/my_schema/customers HTTP/1.0
Content-Type: application/json
Body:
{
 "firstName": "John",
 “lastName”: "Smith",
 "age": 25,
 "address": {
       "streetAddress": "21 2nd Street",
       "city": "New York",
       "state": "NY",
       "postalCode": "10021“,
       "isBusiness" : false  },
       "phoneNumbers": [
       {"type": "home",
        "number": "212 555-1234“ },
       {"type": "fax",
        "number": "646 555-4567“ }  ]
}
```

Analytical tools and business users: Query JSON using SQL

```
select
  c.document.firstName,
  c.document.lastName,
  c.document.address.city
from customers c;”
```

```
firstName    lastName     address.city
-----------  -----------  -------------
“John”       “Smith”      “New York”
```

GeoJSON: JSON for geometries

```
select json_value(
 '{
    "type": "Point",
    "coordinates": [125.6, 10.1]
  }', '$'
  returning sdo_geometry
)
from dual;
```

```
SDO_GEOMETRY(2001, 4326,
SDO_POINT_TYPE(125.6, 10.1,
NULL), NULL, NULL)
```

- Extend JSON support in the database with Spatial operations
- JSON_VALUE() supports GeoJSON and SDO_GEOMETRY
- SDO_GEOMETRY constructors extended to take JSON as input
- Support spatial index and spatial queries on JSON documents
- **Coordinates are in WGS84 (4326)**

GeoJSON Geometry Encoding

- Type of geometry
  - Point, LineString, Polygon
  - MultiPoint, MultiLineString, MultiPolygon

```
{
  "type": "Point",
  "coordinates": [-73.943849, 40.6698]
}
```

- Coordinates
  - Each point is an array that contains the X and Y values
  - Lines, polygons: multiple nested arrays of coordinates

Examples of GeoJSON **Geometries**

- Point

```
{
  "type": "Point",
  "coordinates": [-73.943849, 40.6698]
}
```

- Line

```
{
  "type": "LineString",
  "coordinates": [
    [-73.839157, 42.695122],
    ...
    [-73.846558, 42.700333]
  ]
}
```

- Polygon

```
{
  "type": "Polygon",
  "coordinates": [
    [
      [-105.032997, 39.129829],
      ...
      [-105.032997, 39.129829]
    ]
  ]
}
```

Examples of GeoJSON **Geometries**

- Polygon with Hole

```
{
  "type": "Polygon",
  "coordinates": [
    [
      [-105.052597, 39.791199],
      ...
      [-105.052597, 39.791199]
    ],
    [
      [-104.933578, 39.698139],
      ...
      [-104.933578, 39.698139]
    ]
  ]
}
```

Outer Ring

Inner Ring

- Multi Polygon

```
{
  "type": "MultiPolygon",
  "coordinates": [
    [
      [
        [-104.884201, 39.7402], ..
        [-104.884201, 39.7402]
      ]
    ],
    [
      [
        [-104.933578, 39.698139], .
        [-104.933578,39.698139]
      ]
    ]
  ]
}
```

First Polygon

Second Polygon

GeoJSON **Feature**: Geometry with Properties

A type: “Feature”

A collection of properties

A geometry

```
{
  "type": "Feature",

  "properties": {
    "CITY": "Jersey City",
    "STATE_ABRV": "NJ",
    "POP90": 228537.0,
    "RANK90": 67.0
  },

  "geometry": {
    "type": "Point",
    "coordinates": [ -74.064962, 40.7113 ]
  }
}
```

GeoJSON **Feature Collection**: A Collection of Features

```
{
  "type": "FeatureCollection",
  "features": [

    { "type": "Feature", "properties": { ... }, "geometry": {...} },
    ...
    { "type": "Feature", "properties": { ... }, "geometry": {...} }

  ]
}
```

So, how does this all work ?

1 Create a Table

```
CREATE TABLE us_cities (
  id NUMBER PRIMARY KEY,
  city VARCHAR2(42),
  state_abrv CHAR(2),
  location VARCHAR2(4000)
  CONSTRAINT json_check CHECK (location IS JSON)
);
```

2 Populate

```
INSERT INTO us_cities (id, city, state_abrv, location)
VALUES (1,'New York','NY','{"type":"Point","coordinates":[-73.943849, 40.6698]}');
```

3 Create Index

```
CREATE INDEX us_cities_sx
on us_cities (JSON_VALUE(location, '$' RETURNING SDO_GEOMETRY)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

4 Query

```
SELECT id, JSON_VALUE(location, '$' RETURNING SDO_GEOMETRY)
FROM us_cities where id=1;
```

From **SDO_GEOMETRY** to **GeoJSON**

Generating GeoJSON **Geometries**

- Using the SDO_GEOMETRY method GET_GEOJSON()
  - Remember to use a table alias !

```
select id, c.location.get_geojson() as json from us_cities c;
```

- Using the SDO_UTIL.TO_GEOJSON() function

```
select id, sdo_util.to_geojson(location) as json from us_cities;
```

```
ID    JSON
----- --------------------------------------------------
    1 { "type": "Point", "coordinates": [-73.943849, 40.6698] }
    2 { "type": "Point", "coordinates": [-118.411201, 34.112101] }
...
  195 { "type": "Point", "coordinates": [-118.751299, 34.26305] }

195 rows selected.
```

Generating GeoJSON **Features**

```
select json_object(
  'type'          value 'Feature',
  'properties'    value json_object (
    'CITY'        value city,
    'STATE_ABRV'  value state_abrv,
    'POP90'       value pop90,
    'RANK90'      value rank90
  ),
  'geometry'      value c.location.get_geojson() format json
)
from us_cities c;
```

```
{"type":"Feature", "properties":{"CITY":"New York", "STATE_ABRV":"NY",
"POP90":7322564, "RANK90":1}, "geometry":{ "type": "Point", "coordinates":
[-73.943849, 40.6698] }}
```

Generating GeoJSON **Feature Collection**

```
select json_object (
  'type' value 'FeatureCollection',
  'features' value
    json_arrayagg(
      json_object(
        'type'           value 'Feature',
        'properties'     value json_object (
          'CITY'         value city,
          'STATE_ABRV'   value state_abrv,
          'POP90'        value pop90,
          'RANK90'       value rank90
        ),
        'geometry'       value c.location.get_geojson() format json
      )
    )
  )
from us_cities c;
```

From **GeoJSON** to **SDO_GEOMETRY**

Reading GeoJSON **Geometries**

- Using the **JSON_VALUE()** function
  - Note that this does not let you choose a SRID.

```
select id, json_value(location_g, '$' returning sdo_geometry)
from us_cities;
```

- Using the SDO_UTIL.FROM_GEOJSON() function
  - No SRID specified: will use 4326

```
select id, sdo_util.from_geojson(location_g) from us_cities;
```

  - With an explicit SRID

```
select id, sdo_util.from_geojson(location_g,null,8265) from us_cities;
```

  -

Reading GeoJSON **Features**

- Use the **dot notation** to extract properties
- Get all properties in JSON

```
select c.location_f.properties from us_cities c;
```

```
{"CITY":"Simi Valley","STATE_ABRV":"CA","POP90":100217,"RANK90":195}
...
```

- Get selected properties

```
select c.location_f.properties.CITY, c.location_f.properties.STATE_ABRV
from us_cities c;
```

```
Simi Valley       CA
...
```

Reading GeoJSON **Features**

- Use **JSON_VALUE** to extract the geometry

```
select id, json_value(location_f, '$.geometry' returning sdo_geometry)
from us_cities;
```

```
select id, json_value(c.location_f.geometry, '$' returning sdo_geometry)
from us_cities c;
```

- Or use SDO_UTIL.FROM_GEOJSON() with dot notation

```
select id, sdo_util.from_geojson(c.location_f.geometry)
from us_cities c;
```

Reading from a GeoJSON **Feature Collection**

- Use **JSON_TABLE** to split the collection in individual features

```
select f.id, f.city, f.state_abrv, f.pop90, f.rank90,
       sdo_util.from_geojson(location) location
from json_documents d,
     json_table(
       json_document,'$.features[*]'
       columns (
         id for ordinality,
         city       varchar2(40) path '$.properties.CITY',
         state_abrv varchar2(2)  path '$.properties.STATE_ABRV',
         pop90      number       path '$.properties.POP90',
         rank90     number       path '$.properties.RANK90',
         location   format json  path '$.geometry'
       )
     ) f;
```

```
Name                 Null?    Type
-------------------- -------- ------
ID                   NOT NULL NUMBER
JSON_DOCUMENT                 CLOB
```

Indexing and **Querying**

Indexing GeoJSON Geometries

- Setup spatial metadata

```
insert into user_sdo_geom_metadata values (
  'US_CITIES', 'JSON_VALUE(location_g, ''$'' returning sdo_geometry)',
  sdo_dim_array (
    sdo_dim_element ('Long',-180,180,0.05),
    sdo_dim_element ('Lat',-90,90,0.05)
  ),
  4326
);
```

- The index

```
create index us_cities_sx_g
on us_cities (JSON_VALUE(location_g, '$' returning sdo_geometry))
indextype is mdsys.spatial_index_v2;
```

Indexing GeoJSON Features

- Setup spatial metadata

```
insert into user_sdo_geom_metadata values (
  'US_CITIES', 'JSON_VALUE(location_f, ''$.geometry'' returning sdo_geometry)',
  sdo_dim_array (
    sdo_dim_element ('Long',-180,180,0.05),
    sdo_dim_element ('Lat',-90,90,0.05)
  ),
  4326
);
```

- The index

```
create index us_cities_sx_f
on us_cities (JSON_VALUE(location_f, '$.geometry' returning sdo_geometry))
indextype is mdsys.spatial_index_v2;
```

Querying GeoJSON Geometries

```
select count(*)
from us_cities c, us_interstates i
where sdo_within_distance (
  JSON_VALUE(location_g, '$' returning sdo_geometry),
  i.geom,
  'distance=150 unit=km'
) = 'TRUE'
and i.interstate='I95';
```

```
select count(*)
from us_cities c, us_interstates i
where sdo_within_distance (
  i.geom,
  JSON_VALUE(location_g, '$' returning sdo_geometry),
  'distance=150 unit=km'
) = 'TRUE'
and c.city='Chicago';
```

Querying GeoJSON Features

```
select count(*)
from us_cities c, us_interstates i
where sdo_within_distance (
  JSON_VALUE(location_f, '$.geometry' returning sdo_geometry),
  i.geom,
  'distance=150 unit=km'
) = 'TRUE'
and i.interstate='I95';
```

```
select count(*)
from us_cities c, us_interstates i
where sdo_within_distance (
  i.geom,
  JSON_VALUE(location_f, '$.geometry' returning sdo_geometry),
  'distance=150 unit=km'
) = 'TRUE'
and c.city='Chicago';
```

SDO_GEOMETRY or **GeoJSON**?

- Determined by application design
- Can mix JSON and spatial predicates in same query
- Easy conversion between JSON and SDO_GEOMETRY

- Store in SDO_GEOMETRY

  … and make it look like GeoJSON
- Store in GeoJSON

  … and make it look like SDO_GEOMETRY

Oracle REST Data Services

Using Oracle Rest Data Services (ORDS)

- SDO_GEOMETRY automatically converted to GeoJSON geometry.

```
curl http://localhost:8080/ords/scott/us_cities/?q={"state_abrv":"NV"}
```

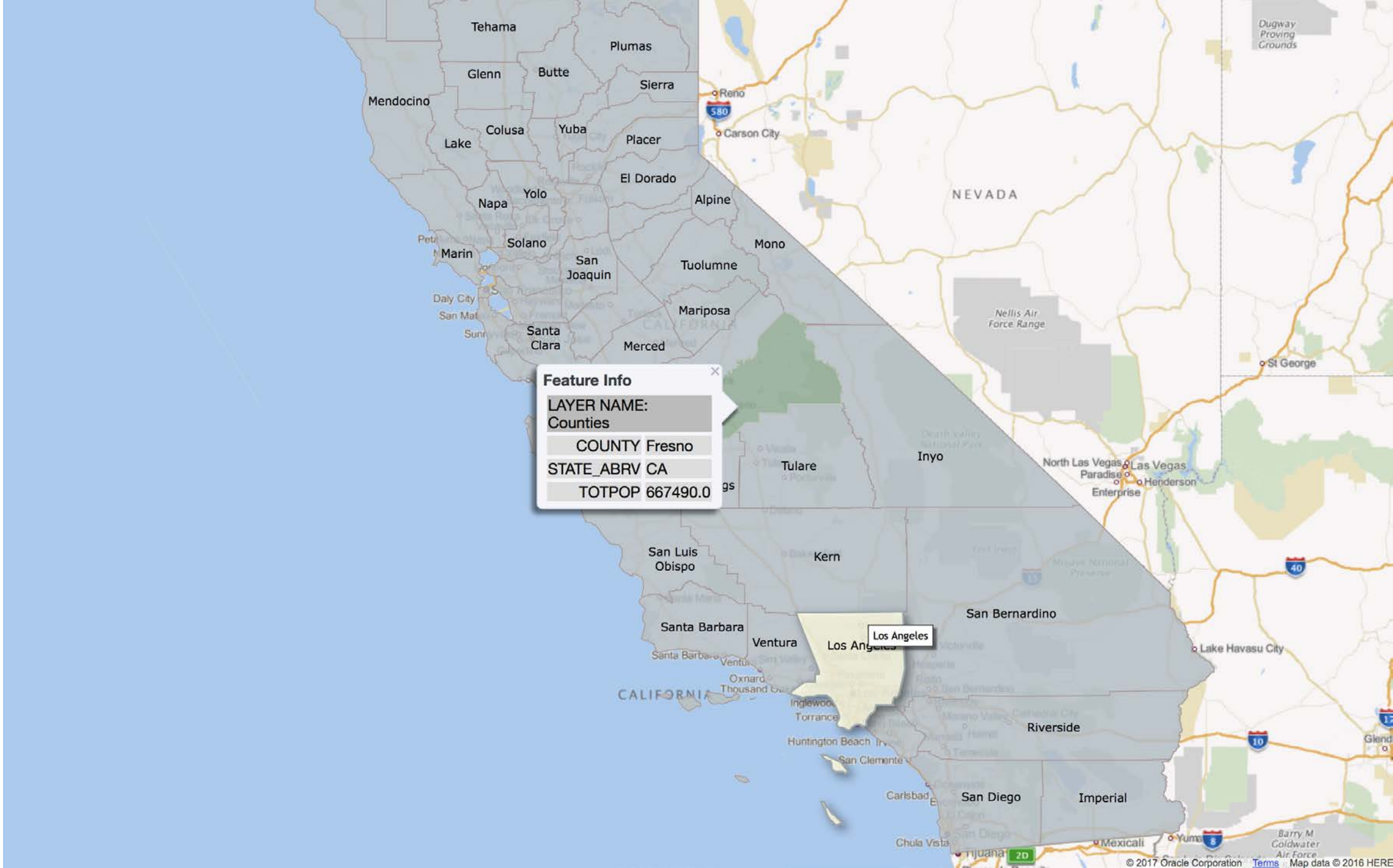
```
{
  "count": 2, "hasMore": false,
  "items": [
    {
      "id": 63,
      "city": "Las Vegas",
      "pop90": 258295,
      "rank90": 63,
      "state_abrv": "NV",
      "location": {"type": "Point", "coordinates": [-115.222799, 36.20575]},
      "links": [{"href":"http://localhost:8080/ords/scott/us_cities/63","rel": "self"}]
    },...
  ],
  "offset": 0, "limit": 25, "links": [...]
}
```

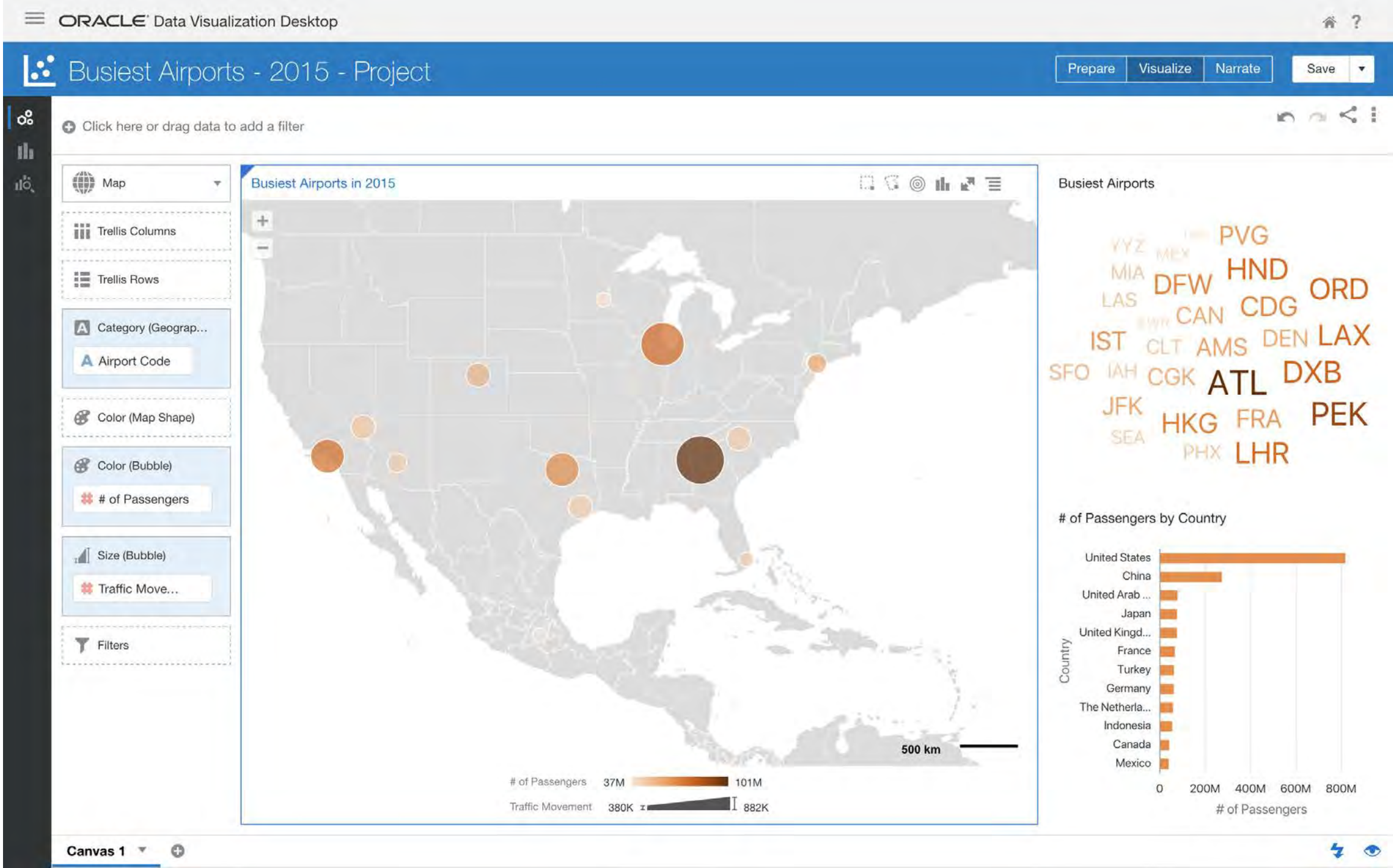
Updating using Oracle Rest Data Services (ORDS)

- GeoJSON automatically converted to SDO_GEOMETRY geometry.

```
curl -X PUT -H "Content-Type:application/json"
   http://localhost:8080/ords/scott/us_cities/196
   -d '{"id":196,"city":"Bismarck","state_abrv":"ND",
   "pop90":72417,"rank90":196,
   "location":{"type":"Point","coordinates":[-100.74869, 46.7666667]}}'
```

- Insert new rows or update existing rows


Feature Info
LAYER NAME: Counties
COUNTY Fresno
STATE_ABRV CA
TOTPOP 667490.0
Los Angeles
Tehama
Plumas
Glenn
Butte
Sierra
Mendocino
Colusa
Yuba
Placer
Lake
El Dorado
Yolo
Napa
Alpine
Solano
Mono
Marin
San Joaquin
Tuolumne
Mariposa
Santa Clara
Merced
Tulare
Inyo
Kern
San Luis Obispo
Santa Barbara
Ventura
San Bernardino
Riverside
San Diego
Imperial
NEVADA
CALIFORNIA
Reno
Carson City
St George
Las Vegas
Henderson
Lake Havasu City
Mexicali
Dugway Proving Grounds
Nellis Air Force Range
© 2017 Oracle Corporation
Terms
Map data © 2016 HERE


ORACLE Data Visualization Desktop
Busiest Airports - 2015 - Project
Prepare
Visualize
Narrate
Save
Click here or drag data to add a filter
Map
Trellis Columns
Trellis Rows
Category (Geograp...
Airport Code
Color (Map Shape)
Color (Bubble)
# of Passengers
Size (Bubble)
Traffic Move...
Filters
Busiest Airports in 2015
500 km
# of Passengers 37M 101M
Traffic Movement 380K 882K
Busiest Airports
PVG
HND
ORD
DFW
CDG
LAX
CAN
IST
AMS
DEN
DXB
ATL
PEK
HKG
FRA
LHR
# of Passengers by Country
United States
China
United Arab ...
Japan
United Kingd...
France
Turkey
Germany
The Netherla...
Indonesia
Canada
Mexico
Country
0
200M
400M
600M
800M
# of Passengers
Canvas 1

A Note about Versions …

18c ORACLE Database

Oracle 12c Release 2:

- JSON operators limited to VARCHAR2
- JSON_OBJECT, JSON_ARRAYAGG …
- Limits the size of features or feature collections

Oracle 18c:

- JSON operators can use **CLOB**
- **No limit** to feature or feature collection size.
- Can use **any** sort of shape (arcs, measures, …)

ORACLE

Other Geometry Serialization Formats …

- OGC Well-Known Text (WKT)

```
POINT (-73.943849 40.6698)
```

- OGC Well-Known Binary (WKB)

```
0000000001C0527C6805A2D730404455BC01A36E2F
```

- GML

```
<gml:Point srsName="EPSG:4326" xmlns:gml="http://www.opengis.net/gml"><gml:coordinates
decimal="." cs="," ts=" ">-73.943849,40.6698 </gml:coordinates></gml:Point>
```

- KML

```
<Point><extrude>0</extrude><tessellate>0</tessellate><altitudeMode>relativeToGround</alti
tudeMode><coordinates>-73.943849,40.6698 </coordinates></Point>
```

Engage with the Spatial and Graph SIG

Promotes interaction and communication to drive the market for spatial technology and data

Members connect and exchange knowledge via online communities and at conferences and events

- Talk with us at the Summit!
-

Morning Arrivals Tues & Wed 7:45-8:30 a.m. Registration Area	Receptions Tues & Wed evenings Spatial Table, lobby	**Birds of a Feather Lunch** **Wednesday** **12-1pm** **Auditorium**

- Join us online
  - tinyurl.com/oraclespatialcommunity
  - Search for “Oracle Spatial and Graph Community”
- Contact us:
  oraclespatialsig@gmail.com @oraspatialsig

ALBERT GODFRIND
Spatial Solutions Architect
Location and Geospatial Services

Oracle Corporation

Greenside
400 av. Roumanille - BP 309
06906 Sophia-Antipolis
France

phone +33 4 93.00.80.67
mobile +33 6 09.97.27.23
albert.godfrind@oracle.com
http://www.oracle.com/

Integrated Cloud

Applications & Platform Services


ORACLE®