ORACLE

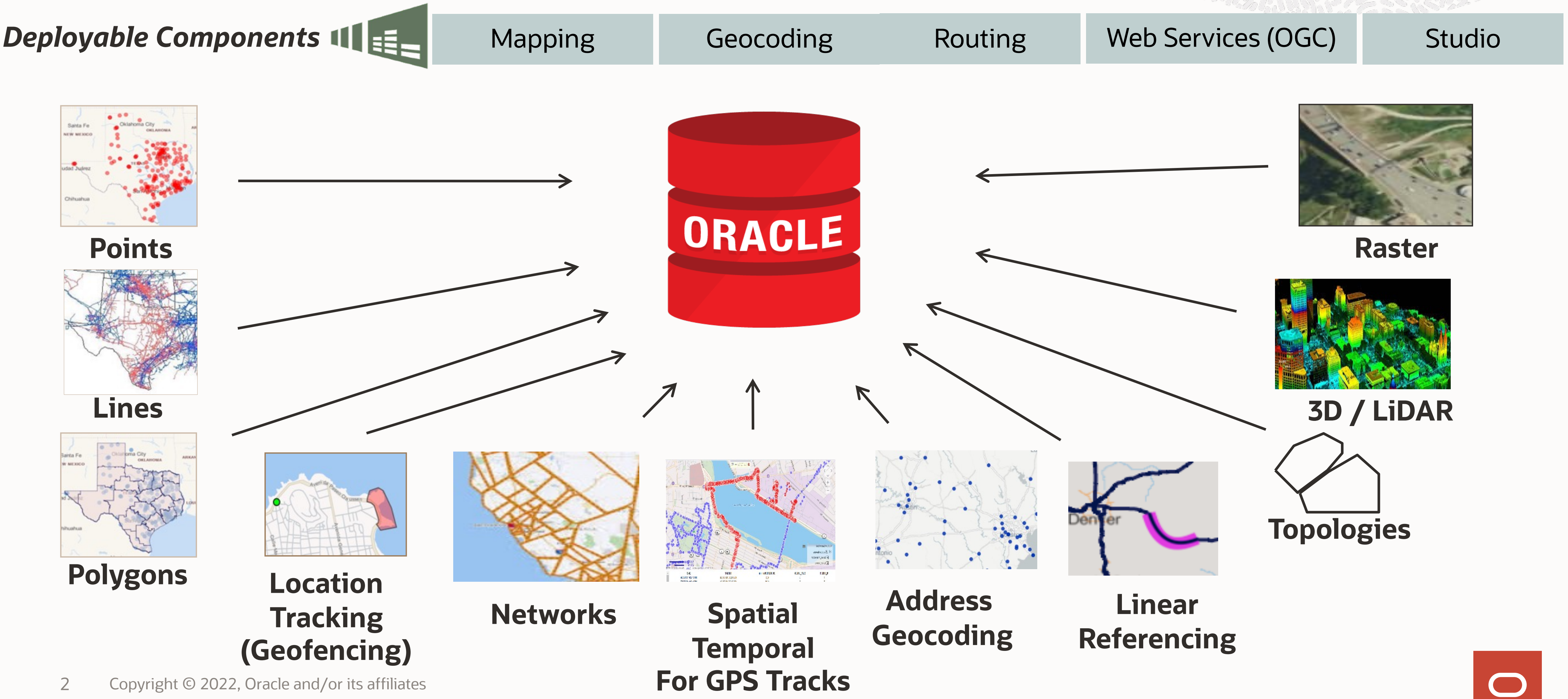
Optimize Oracle Spatial Performance

Best Practices, Tips and Tricks

Daniel Geringer
Spatial Solution Specialist
September 1, 2022

Oracle Spatial Features

Included in Every Oracle Database License



Oracle Spatial – Spatial Data and Models

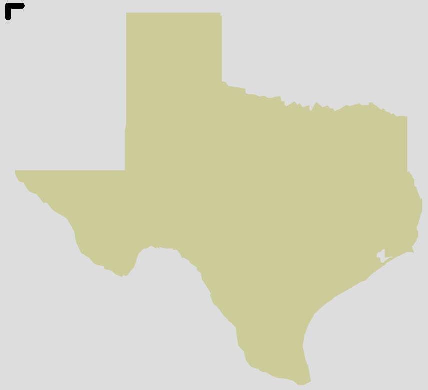
- Spatial data stored in database tables with same **security, high availability, manageability, data integrity, and scalability** as non-spatial data.

  - Vector data – Points, Lines, Polygons
  - Raster data – Digital Imagery and Gridded Data
  - GPS Tracking data – For Coinciding track analysis / GeoFence analysis
  - LIDAR Data – Point Cloud / LIDAR data
  - Network Model – Drive Time / Connectivity Analysis

- Transparent Data Encryption, Data Redaction, Active Data Guard, Replication, Parallel Query, and more

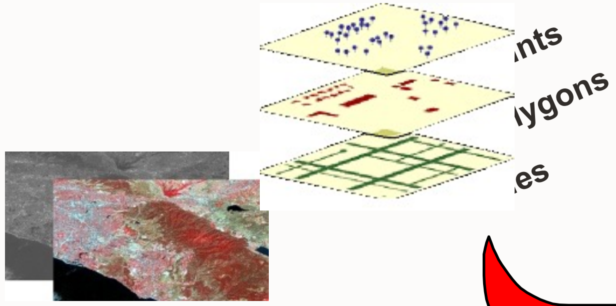
Vector Data

- Points, Lines, Polygons
- Geometry stored in ordinary database tables
- Ordinary data modeling concepts
  - Normalized tables, 1-1 relationships
  - Denormalized tables not recommended
- Geometry Validation
- Spatial indexing
- Spatial queries – most of the time, spatial predicate is most selective

STATE_NAME	CAPITAL	GEOMETRY
CALIFORNIA	Sacramento	
TEXAS	Austin	

In-Database Spatial Capabilities

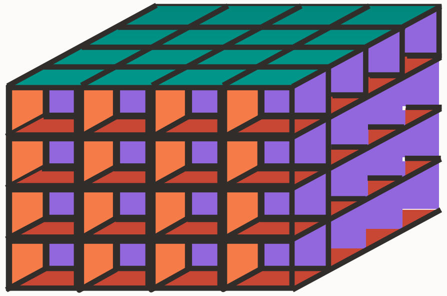
Spatial Data Types



Spatial Data
Stored in the Database
(vector,raster,Lidar)

Spatially Enabled Database

Spatial Indexing



Fast Access to
Spatial Data

Spatial Analysis Through SQL

```
SELECT a.customer_name, a.phone_number
FROM policy_holders a
WHERE sdo_within_distance ( a.geom, hurricane_path_geom,
       'distance = 10 unit = mile') = 'TRUE';
```

Vector Data – Table Partitioning

- Generally recommended when table size exceeds 50 million rows, but can be effective for much smaller tables too.
- **Temporal partitioning** is very common:
  - For manageability – make it easy to bulk add new data quickly, and age older data out, with exchange partition and drop partition
  - For performance –
    - Enables searching within a specified time period "only".
    - If not partitioned, spatial computation applied first across all times, and then time predicate.
- **Feature type partitioning** can be very effective too:
  - For example, FEATURE_TYPE = transformer, substation, manhole, utility pole, etc..
  - Without partitioning, spatial applied to all features, then feature_type applied. This is not optimal.
  - Partitioning enables spatially searching just the feature_types of interest.

Vector Data – Table partitioning strategies

- Range, Hash, List, Interval, and Reference partitioning strategies
- Composite (generate subpartitions for each partition)
  - Range-Range
  - Range-Hash
  - Range-List
  - List-Range
  - List-Hash
  - Hash-Hash
  - Hash-List
  - Hash-Range
- Local Partitioned Spatial Indexes are very effective

Vector Data – Geometry Validation Is Important

- Open Geospatial Consortium (OGC) – standard geometry validation
- Extremely common for data sets to contain invalid geometries
- Common issues –
  - Repeated consecutive points in a line or polygon
  - Self crossing polygons
- Invalid geometries may result in incorrect results
- Use built in validation routines to identify invalid geometries (validate_geometry_with_context)
- Use built in routines to fix invalid geometries (rectify_geometry)

Fastest Way To Validate Geometries – With Parallel Query

- Similar output to SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT
- You control the parallel degree

```
CREATE TABLE validation_results PARALLEL 16 NOLOGGING AS
SELECT sdo_rowid, status
FROM (SELECT rowid sdo_rowid,
             sdo_geom.validate_geometry_with_context(geom, tolerance) status
      FROM roads)
WHERE status <> 'TRUE';
```

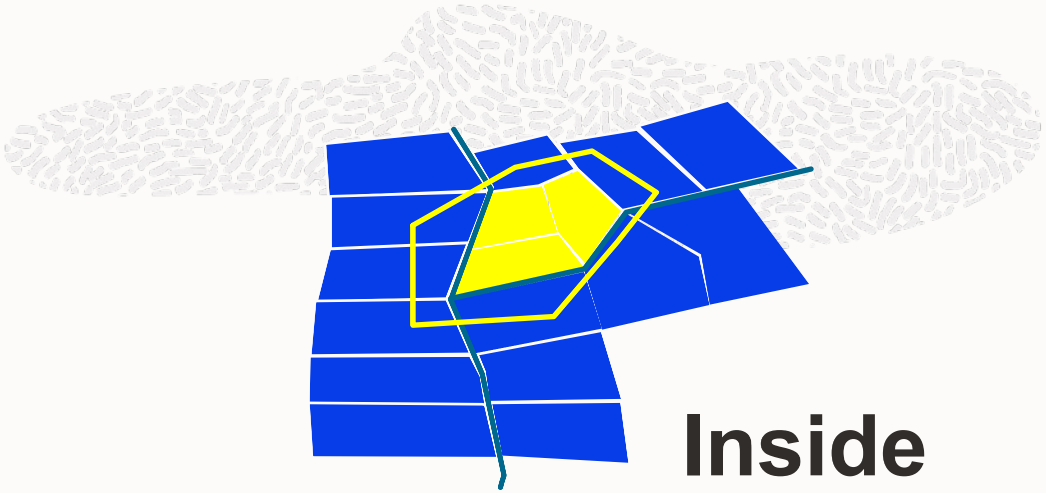

Geodetic Tolerance – Now support smaller than 0.05

Why is this important?

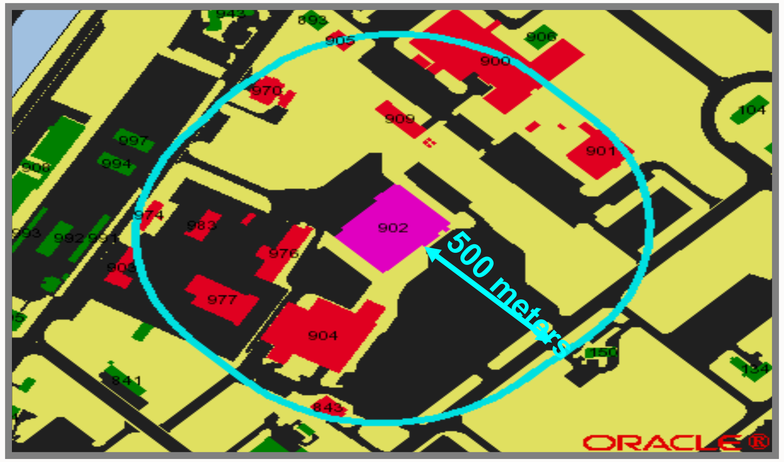
- Geometries may be invalid at 0.05 tolerance, which requires fixing them
- Before you fix, you can try a smaller tolerance than 0.05 (5 centimeters), for example, 0.005 (5 millimeters)
- Just a tolerance change may address many 13356 (repeated duplicate vertices) and 13349 (self intersection polygon) errors.
- Tolerance should be consistent across all spatial layers you plan to compare

Vector Data – Spatial Operators

- Full range of spatial operators
  - Topological Operators
    - Inside Contains
    - Touch Disjoint
    - Covers Covered By
    - Equal Overlaps
  - Distance Operators
    - Within Distance
    - Nearest Neighbor



Inside



Within Distance

Spatial Vector Acceleration

SPATIAL_VECTOR_ACCELERATION

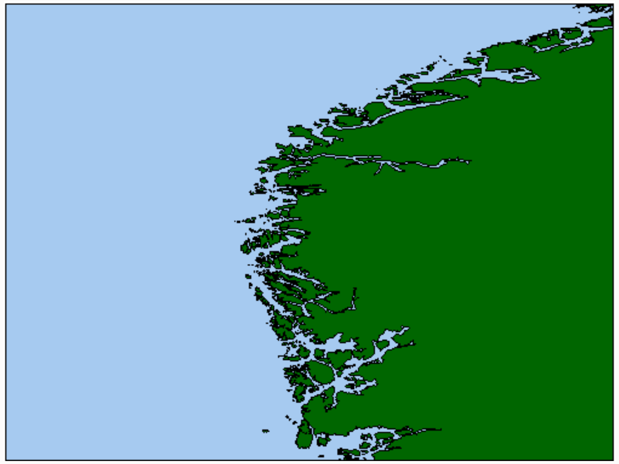
Very Important Initialization Parameter

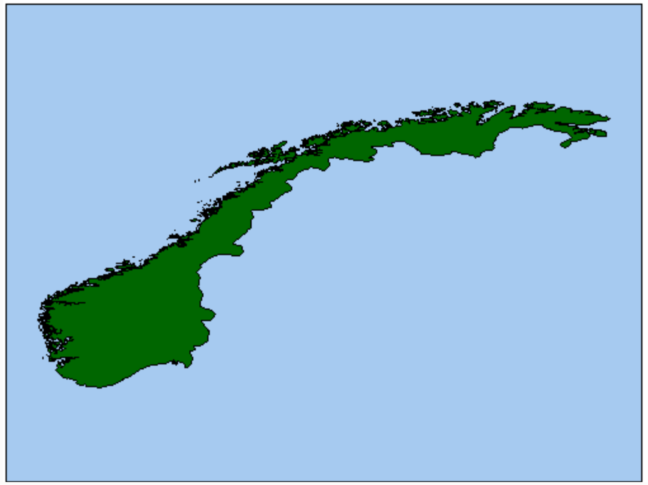
- **Faster algorithms** for spatial operations and functions **(100's of times faster)**
- Recommended for any application with mission critical spatial query performance requirements.
- Oracle Initialization Parameter – Make sure it is set to TRUE
  - ALTER SYSTEM SET SPATIAL_VECTOR_ACCELERATION = TRUE
  - ALTER SESSION SET SPATIAL_VECTOR_ACCELERATION = TRUE
- All users benefit!

SPATIAL_VECTOR_ACCELERATION

Oracle Initialization Parameter

- Spatial operators
  - Performance optimizations for **"high vertex count"** query window (2nd argument of spatial operator).
  - Relation masks hundreds of times faster (i.e. COVEREDBY, COVERS, TOUCH, etc.)
    - Time Zone Polygon Example
    - Very detailed coastline
    - 343,395 vertices
    - Hundreds of times faster
    - 300x faster for this test





Oracle Support Note – Doc ID 2514624.1

What Is the Latest Spatial Patch Bundle for 12c and Higher Databases?

- DBRU inclusion of Spatial patches needed by most
- Live document maintained by Oracle Support
  - DBRU specific recommended Spatial patches to apply
  - Updated when a new DBRU is released
  - Updated when Spatial patches to apply are superseded
- 19.14 DBRU and beyond:
  - DBRU alone includes Spatial updates needed by most
  - Doc ID 2514624.1 DBRU specific Spatial patches are targeted for inclusion in the next DBRU. Depending on the cutoff tome, possibly the subsequent DBRU.
- 19.13 DBRU and earlier, key to apply Spatial patches in Doc 2514624.1

Spatial Data Organization on Disk

Strategies To Optimize Performance

Spatial Data Organization On Disk

To optimize spatial query performance

- Spatial Indexes are organized (by default)
  - Spatial indexes are stored in a secondary table, managed by Oracle (MDRT$ table)
  - Spatial indexes (store geometry MBRs), along with rowid pointers back to geometries in the base table.
  - Spatial indexes cluster MBRs close to each other in the same database blocks
- Spatial Data is not organized (by default)
  - While geometry MBRs are clustered in the same database block, associated base table geometries are usually scattered
  - At query time, scattered geometries can result in many database block gets
  - Solution is to order by a linear key

Spatial Data Organization on Disk

Two strategies

- For point only data
  - Use Oracle built in feature – **Attribute Clustering**
- For lines and polygons
  - Order by linear key with Oracle Spatial function (sdo_util.linear_key)
- Both strategies discussed in the next few slides

Spatial Data – Organization on Disk

For Point Only Data – Use Attribute Clustering

- Interleaved attribute clustering –
  - Not spatial specific
  - Must store point as two NUMBER columns, not as SDO_GEOMETRY
  - Can create a function based spatial index
  - Can cluster time,x,y too
- Assume your point data is longitude/latitude:
  - Just append the following clause to the CREATE TABLE statement
  - CLUSTERING BY INTERLEAVED ORDER (longitude, latitude) YES ON LOAD;
  - Full example on next slide

For Point Only Data – Use Attribute Clustering (Not Spatial Specific)

Example

```
CREATE TABLE track_table (user_id     NUMBER,
                          capture_time  DATE,
                          longitude     NUMBER,
                          latitude      NUMBER,
                          date_as_number NUMBER) NOCOMPRESS NOLOGGING
                          CLUSTERING BY INTERLEAVED ORDER (capture_time, longitude,
latitude) YES ON LOAD;
```

```
--Attribute clustering only available for direct path insert operations, for example
from staging table or external table
INSERT /*+ APPEND PARALLEL (8) */ INTO TRACK_TABLE
SELECT user_id, capture_time, longitude, latitude,
       capture_time - to_date('01-01-2019', 'MM-DD-YYYY')
FROM external_staging_table;
```

Spatial Data – Organization on Disk

For Line and Polygon Data – Use Spatial Clustering (sdo_util.linear_key)

- Interleaved Attribute Clustering not for lines or polygons or tables with SDO_GEOMETRY columns
- For lines and polygons, user Spatial functions sdo_util.linear_key instead
- sdo_util.linear_key –
  - Based on gridding a coordinate system
  - Every cell in the grid has a unique key
  - Give a point as input, function returns the unique key associated with the cell the point falls in
  - For lines and polygons, choose input point (for example, first point or center point)
  - On insert, ordering by linear key will optimally cluster line and polygon spatial data on disk
- Example on next slide

Spatial Data – Organized in a Tablespace

For Line and Polygon Data – Use Spatial Clustering (sdo_util.linear_key)

```
CREATE TABLE ship_tracks_ordered (col1 NUMBER, col2 NUMBER, geom SDO_GEOMETRY, id
       NUMBER);
```

```
INSERT /*+ APPEND PARALLEL (6) */ INTO ship_tracks_ordered NOLOGGING
WITH part1 AS ( select col1, col2,
                       geom,
                       sdo_geom.sdo_pointonsurface (geom,.005) first_point
                FROM ship_tracks_not_ordered
SELECT col1, col2,
       geom,
       row_number() OVER (ORDER BY sdo_util.linear_key (p1.first_point.sdo_point.x,
                                                        p1.first_point.sdo_point.y,
                                                        -180,-90,180,90,22)) id
FROM part1 p1;
```

NOTE - sdo_util.linear_key signature with x and y available in Oracle 19.13 and newer. Other signatures available before 19.13

- Any extent that covers all data can be used. For longitude/latitude use (-180,-90,180,90)
- For world Mercator use (-21000000,-75000000,21000000,240000000)

Spatial Function Based Indexes

For Tables With No SDO_GEOMETRY Column

Spatial Function Based Index

For tables with no SDO_GEOMETRY column

- For uniform geometries (same number of vertices in every row)
  - Point data (x1, y1)
  - Two point lines (x1, y1, x2, y2)
  - Box polygons (min_x, max_x, min_y, max_y)
- Steps (example in next few slides for track_table on previous slide):
  1. Create a function that returns an SDO_GEOMETRY
  2. Populate user_sdo_geom_metadata
  3. Create spatial function based index
  4. Run spatial queries

Spatial Function Based Index

STEP 1 – Create a function that returns an SDO_GEOMETRY

```
CREATE OR REPLACE FUNCTION get_geometry (lon NUMBER, lat NUMBER)
   RETURN sdo_geometry DETERMINISTIC PARALLEL_ENABLE AS
BEGIN
   IF lon IS NULL OR lat IS NULL
   THEN
      RETURN NULL;
   ELSE
      RETURN sdo_geometry(2001,4326, sdo_point_type(lon,lat,null),null,null);
   END IF;
END;
```

****NOTE** Functions that return SDO_GEOMETRY (or any object) should be declared DETERMINISTIC for optimal query performance**

Spatial Function Based Index

STEP 2 – Populate user_sdo_geom_metadata

```
INSERT INTO user_sdo_geom_metadata VALUES (
        'TRACK_TABLE', 'SCOTT.GET_GEOMETRY(LONGITUDE,LATITUDE)',
        sdo_dim_array(sdo_dim_element('x',-180,180,.005),
                      sdo_dim_element('y',-90,90,.005)).
        4326);
```

- **NOTE** For user_sdo_geom_metadata entry:
  - Specify function name instead of a column name
  - OWNER.FUNCTION_NAME must be specified
  - Function parameters must match table column names

Spatial Function Based Index

STEP 3 – Create spatial function based index

```
CREATE INDEX track_table_sidx ON track_table (get_geometry(longitude,latitude))
INDEXTYPE IS mdsys.spatial_index_V2 PARAMETERS('layer_gtype=point
        cbtree_index=true')
```

- **NOTE**
  - rtree and cbtree spatial index supported. For rtree, omit cbtree_index=true
  - cbtree spatial index will be discussed more in an upcoming slide
  - cbtree spatial index requires mdsys.spatial_index_V2 for local spatial indexes on partitioned tables
  - Specify layer_gtype=point during create index to optimize query performance against point only layers

Spatial Function Based Index

STEP 4 – Try a spatial query

```
SELECT count(*)
FROM track_table
WHERE sdo_anyinteract (get_geometry(longitude,latitude),
                      sdo_geometry(2003,4326,null,sdo_elem_info_array(1,1003,3),
                                   sdo_ordinate_array(-75,35,-74,36)))='TRUE';
```

- **NOTE**
  - Normally, a geometry column is specified as first parameter of a spatial operator
  - Instead, specify the function used to create the spatial function based index

CBTREE – Point Only Spatial Index

Optimized For Streaming Point Data

CBTREE – Point Only Spatial Index

- Optimized for ingesting streamed point data with spatial index enabled
- CBTREE spatial index:
  - Designed to handle concurrent DML from multiple sessions (i.e. connection pool)
  - Much faster spatial index creation
- No spatial functionality compromised
- Specify cbtree_index=true

```
CREATE INDEX point_sidx ON cities (geometry)
INDEXTYPE IS mdsys.spatial_index_v2
PARAMETERS('layer_gtype=point cbtree_index=true');
```

Parallel Query and Spatial

US Rail Application

Parallel Query and Spatial Operators

US Rail Application

- Requirement
  - GPS locations for each train collected throughout the day
  - Each location has other attributes (time, speed, and more)
  - GPS locations have a degree of error, so they don't always fall on a track.
  - Bulk nearest neighbor queries to find closest track, and project reported train positions onto tracks
- This information is used for:
  - Tracking trains
  - Analysis for maintenance, ensure engineers are within parameters, etc.

Parallel Query and Spatial Operators

What we tested

- 45,158,800 GPS train positions.
- For each train position:
  - Find the closest track to the train (with SDO_NN)
  - Then calculate the position on the track closest to the train

Parallel Query and Spatial Operators

US Rail Application

```
CREATE TABLE results PARALLEL 72 NOLOGGING AS
SELECT a.locomotive_id, sdo_lrs.find_measure (b.track_geom, a.locomotive_pos)
FROM locomotives a, tracks b
WHERE sdo_nn (b.track_geom, a.locomotive_pos, 'sdo_num_res=1') = 'TRUE';
```

Parallel Query and Spatial Operators

Exadata Results

- Exadata Half RAC:
  - 34.75 hours serially vs. 44.1 minutes in parallel
  - Linear Scalability - 48 database cores - 47x faster
- X9-2 even faster with newer generation chips – **Easily exceed 100x faster**

Spatial Clustering

For Trend Analysis

Server Side Parallel Enabled Clustering

- **Trend Analysis** - Telematics clustering (this is really powerful)
  - GPS points collected in the billions
  - Cluster points to generate much more manageable datasets for analytics
  - Identify patterns or trends associated with clustered data.
  - Clusters at a particular time of day tend to be near a particular type of store or restaurant.

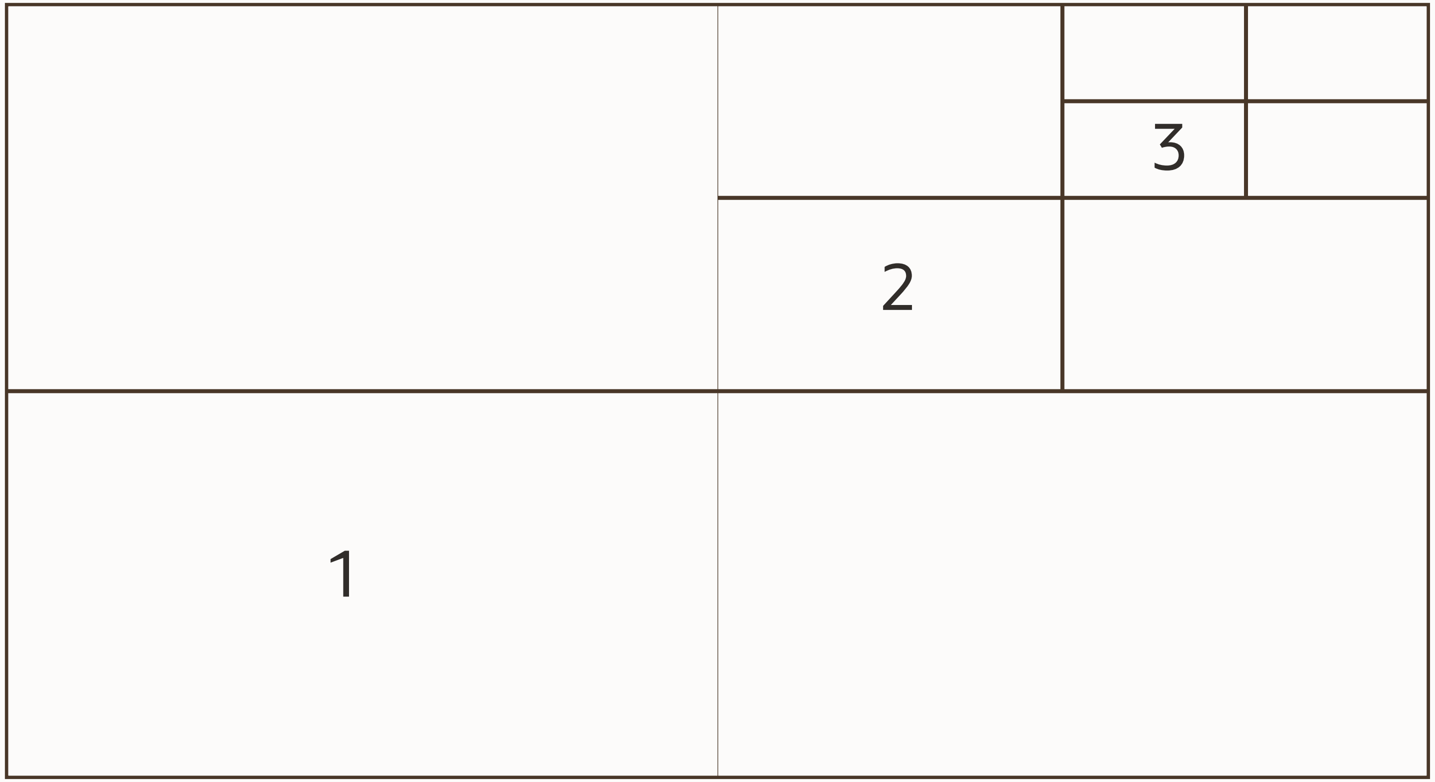
Server Side Parallel Enabled Clustering

- Cluster millions of rows in seconds (server side)
  - 1 million points into 62708 clusters in 0.86 seconds parallel 16 (subsecond performance)
  - Over 1 billion points (1,024,000,000) into 62708 clusters in 7 minutes parallel 16
- Returns cluster center and count
- Effective for Automatic Zoom In/Out Clustering in mapping applications
- Especially when too many rows to cluster client side
- Clustering results can be persisted (precomputed), especially when clustering millions or billions of records
- Clustering can be performed on the fly too… and also parallel enabled

Spatial Clustering – Also uses sdo_util.linear_key

Pick a cell size – Quad Tile Based

- Level is a parameter for sdo_util.linear_key
- Defines tile size for clustering
- Level 1 – 1/4 coord system
- Level 2 – 1/16 coord system
- Level 3 – 1/64 coord system
- etc. …

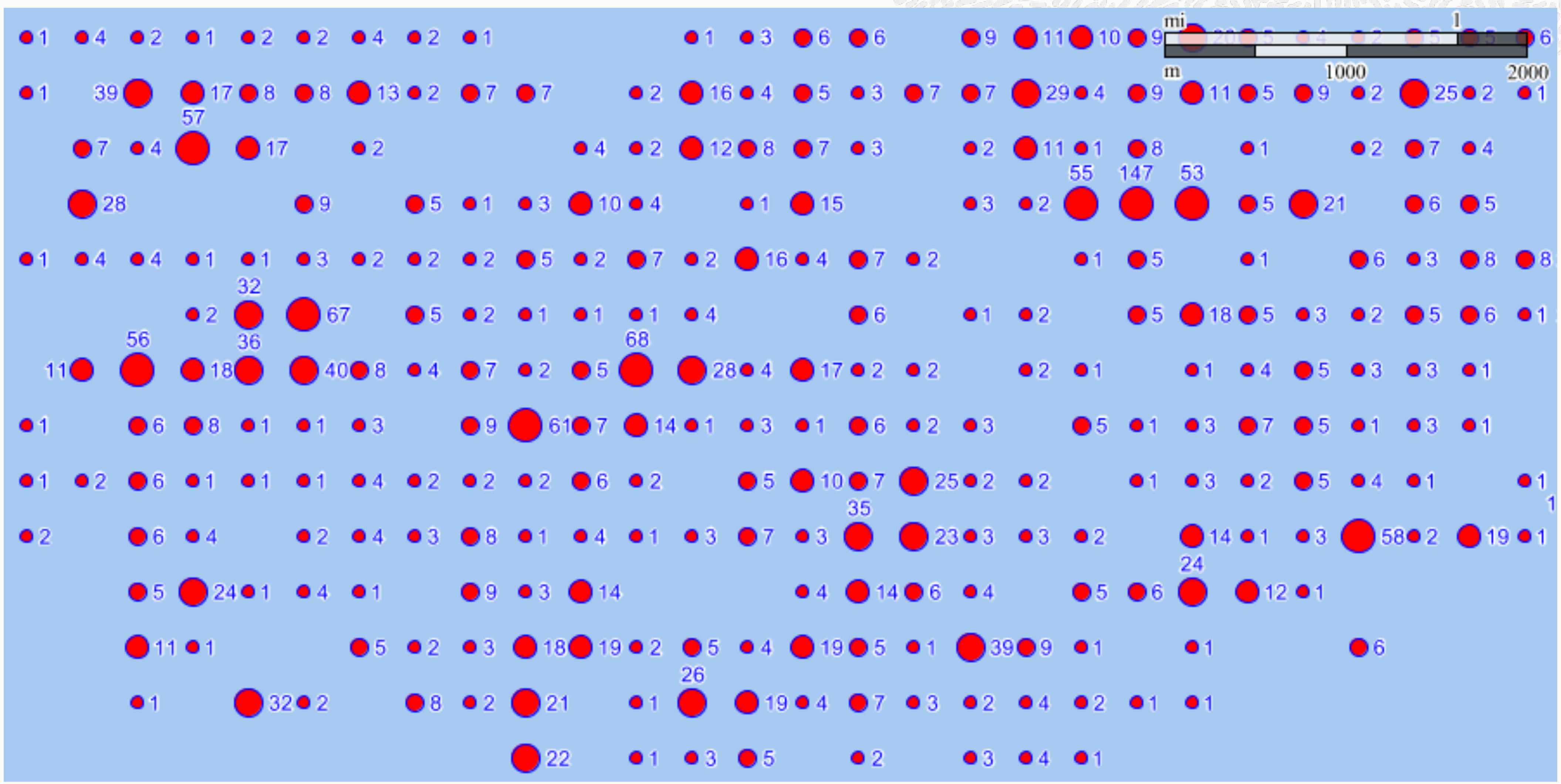


Spatial Clustering - Example

```
ALTER SESSION ENABLE PARALLEL DML;
CREATE TABLE results (cnt NUMBER, center SDO_GEOMETRY);
INSERT /*+ append parallel(16) */ INTO results NOLOGGING
SELECT count(*),
       sdo_util.linear_key_center (cell_id, -180, -90, 180, 90)
FROM ( SELECT sdo_util.linear_key (longitude, latitude, -180, -90, 180, 90, 15) as
       cell_id
       FROM one_billion_row_table a)
GROUP BY cell_id;
```

NOTE Use sdo_util.linear_key_boundary to see the cell geometry.
Signature is similar to sdo_util.linear_key_center.

Server Side Spatial Clustering – Street Network - Result



Spatial Clustering – GPS Data Example

- When clustering GPS positions of many users, a single user may report many positions in a cluster.
- This example ensures clusters count reflects the count of “distinct” users.

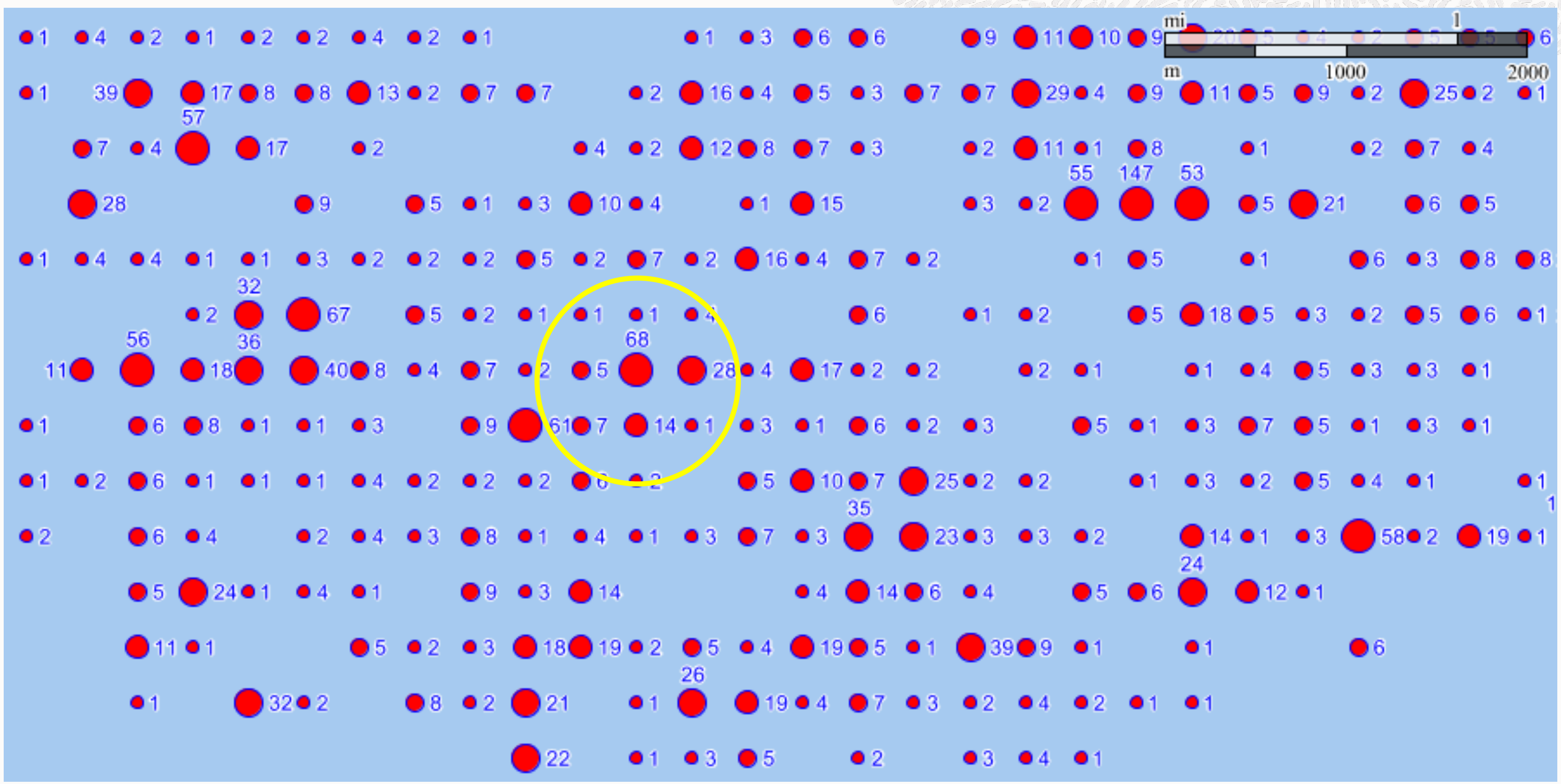
```
ALTER SESSION ENABLE PARALLEL DML;
CREATE TABLE results (cnt NUMBER, center SDO_GEOMETRY);

INSERT /*+ append parallel(16) */INTO results NOLOGGING
SELECT count(*), sdo_util.linear_key_center (cell_id, -180, -180, 180, 180)
FROM (SELECT cell_id, user_id, count(*)
     FROM (SELECT sdo_util.linear_key (longitude, latitude,-180,-90,180,90, 15) as
               cell_id, user_id
           FROM one_billion_row_table a)
     GROUP BY cell_id, user_id )
GROUP BY cell_id;
```

Spatial Temporal Clustering – GPS Data Example

- When clustering GPS positions of many users, a single user may report many positions in a cluster
- This example ensures clusters count reflects the count of “distinct” users

```
ALTER SESSION ENABLE PARALLEL DML
CREATE TABLE results (day varchar2(100), hour_range_id NUMBER, cnt NUMBER, center SDO_GEOMETRY);
INSERT /*+ append parallel(16) */ INTO results NOLOGGING
SELECT day, hour_range_id, count(*) cnt,
       sdo_util.linear_key_center(cell_id, -180, -90, 180, 90, 15) center_geom
FROM (SELECT cell_id, user_id, day, hour_range_id, count(*)
      FROM (SELECT cell_id,
                   user_id,
                   day,
                   CASE WHEN hour_of_day >= 0 AND hour_of_day < 6 THEN 1
                        WHEN hour_of_day >= 6 AND hour_of_day < 10 THEN 2
                        WHEN hour_of_day >= 10 AND hour_of_day < 16 THEN 3
                        WHEN hour_of_day >= 16 AND hour_of_day < 20 THEN 4
                        WHEN hour_of_day >= 20 AND hour_of_day < 24 THEN 5
                   END hour_range_id
            FROM ( SELECT sdo_util.linear_key (lon, lat, -180,-90,180,90, 15) as cell_id, user_id,
                          to_char(reported_time, 'MONDDYYYY') day,
                          to_number(to_char(reported_time, 'HH24’)) hour_of_day
                   FROM one_billion_row_table a))
      GROUP BY cell_id, user_id, day, hour_range_id)
GROUP BY cell_id, day, hour_range_id;
```

Search For all Business within 2km of 8am cluster center with high count – Trend Analysis



Oracle Spatial – Spatial Data and Models (Summary)

- Spatial data stored in database tables with same **security, high availability, manageability, data integrity, and scalability** as non-spatial data.

  - Vector data – Points, Lines, Polygons
  - Raster data – Digital Imagery and Gridded Data
  - GPS Tracking data – For Coinciding track analysis / GeoFence analysis
  - LIDAR data – Point cloud / LIDAR data
  - Network Model – Drive Time / Connectivity Analysis

- Transparent Data Encryption, Data Redaction, Active Data Guard, Replication, Parallel Query, and more

Resources on Oracle Spatial Technologies

- Oracle Spatial technologies: https://www.oracle.com/database/spatial/
- Oracle LiveLabs: https://bit.ly/golivelabs-spatial
- Blog: https://blogs.oracle.com/oraclespatial/ , https://blogs.oracle.com/database/category/db-spatial
- Slack (Please join #spatial channel): https://bit.ly/Join-ANDOUC-Slack
- YouTube: https://bit.ly/Spatial-Graph-YouTube
- AskTOM video series: https://bit.ly/AskTOMSpatial
- LinkedIn: https://bit.ly/Spatial-Graph-LinkedIn
- Twitter: @SpatialHannes, @JeanIhm

Questions & Answers

Please enter your questions in the **Zoom Q&A box**

ORACLE