Oracle Spatial for PostGIS Users – Understand, Isolate and Migrate

Simon Greener,
The SpatialDB Advisor
simon@spatialdbadvisor.com
http://www.spatialdbadvisor.com
Introduction

- Oracle Spatial and PostGIS are two of the most mature implementations of a spatial type system for their relevant host databases.
- PostgreSQL/PostGIS is increasing in strength, EnterpriseDB is aiming to convert businesses from Oracle to PostgreSQL but....
  - You rarely see, on a customer's servers, only ONE DB product;
  - Learn to live together: Not Either/Or but Both/And!
- This talk provides an understanding of:
  - Oracle Locator/Spatial concepts and components;
  - Relevant standards in common;
    - Metadata structures;
    - Type system.
  - Tolerance model
  - Programmatic and framework issues.
  - Helping you understand each, and know how to migrate between or minimise solutions that can be deployed on both databases.
Oracle Releases

- Oracle's first version of “Spatial” was released with 8iR1 (8.1.5) back in 1999 (10 years ago).
  - OpenGIS SFS for SQL was released in 1999;
  - No initial support for OGC/SQL/MM object type:
    - “Singly inherited”?
    - Timing of releases?
  - Had 6 major releases since some with, without major spatial releases:
    - 9i Releases 1 (9iR1) and 2 (9iR2)
    - 10g Releases 1 (10gR1) and 2 (10gR2)
    - 11g Releases 1 (11gR1) and 2 (11gR2)
## Oracle Spatial Functionality Releases...

<table>
<thead>
<tr>
<th>Release</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>8i</td>
<td>Basic SDO_Geometry type &amp; Quad Tree indexing</td>
</tr>
<tr>
<td>9iR1</td>
<td>Geodetic, Linear Referencing System, RTree Spatial index, Spatial Aggregate functions, Partitioned Indexes</td>
</tr>
<tr>
<td>9iR2</td>
<td>Various function additions and changes eg SDO_AGGR_UNION, SDO_AGGR_MBR</td>
</tr>
<tr>
<td>10gR1</td>
<td>Annotation Point GeoRaster, Network Data Model, Geocoding, Topology, Spatial Analysis and Mining (Spatial correlation, colocation, clustering, prospecting, binning) Various function additions and changes.</td>
</tr>
<tr>
<td>10gR2</td>
<td>EPSG SRS, WKB in/out, Various function additions and changes.</td>
</tr>
<tr>
<td>11gR1</td>
<td>TIN SOLIDS POINTCLOUDS, 3D RTree indexing and some 3D query operators.</td>
</tr>
</tbody>
</table>
| 11gR2   | SDO_AGGR_SET_UNION (cf STRM ST_Union) Various function additions and changes. KML in/out; GML in
Oranges and Lemons

- Oracle's “spatial” functionality is available in two versions: Locator and Spatial.
  
  - **Locator** is a free feature of Oracle Database available on all versions (XE, SE1, SE, and EE) and releases from 9iR1 that implements the basics of a vector type system that includes:
    
    - An object type (SDO_GEOMETRY) that describes and supports any type of geometry (whole earth geometry model for geodetic data introduced in 9iR1 – **PostGIS end of 2009**);
    - A spatial indexing capability (Quad Tree and RTree);
    - Spatial index aware operators for performing spatial queries;
    - Some geometry functions (not geoprocessing eg SDO_Union) and the SDO_AGGR_MBR spatial aggregate function;
    - Coordinate system support for explicit geometry transformations;
    - Spatial utility functions (eg Rectify_Geometry cf SQL Server 2008’s MakeValid)
Oranges and Lemons (Cont)

- **Spatial includes:**
  - All Spatial Functions e.g. SDO_Union and aggregates e.g. SDO_AGGR_UNION;
  - Linear Referencing System (*c.f. PostGIS LRS functions*);
  - GeoRaster Storage, Indexing and Querying (*cf WKT Raster beta*);
  - Network Data Model;
  - Topology Data Model (*c.f. PostGIS Topology beta implementation*);
  - Spatial Analysis and Mining (SAM) Functions;
  - Spatial Routing Engine (*c.f. PostGIS pgRouting*);
  - Geocoding Engine;
  - 3-D Geometry, Surface, and Point Cloud Storage; Index and Query;
  - Semantic Content Storage, Indexing and Querying (RDF/OWL Support).

- Cannot be purchased separately!
- Can only be deployed on Enterprise Edition (EE)!
Parallel Processing, Partitioning and Replication

- Oracle’s native spatial data type allows for:
  - Partitioning support for spatial indexes;
  - Parallel index builds for spatial R-tree indexes;
  - Parallel spatial queries;
  - Replication

- Some features available only with Enterprise Edition.
  - And so, $$$$$$$$$$$$$$$$$$$
Software that supports Oracle

- Oracle's focus, as always, is on sales and marketing.
- Technology Partners and Spatial Integrator Partners are all commercial businesses.
- But FOSS4G software also supports Oracle:
  - OGR, GDAL, FDO, uDig, GeoTools, Quantum GIS, GeoServer, Deegree, MapServer, MapGuide OS,...
Standards Bodies ...

- We look to those standards bodies that are defining applicable standards to control/support design, use and uptake of spatial databases:
  - Open Geospatial Consortium (OGC) Inc
  - International Standards Organisation (ISO)
  - W3C Consortium (XML/SVG…)
- Help to “level the playing fields” for open source projects.
- Oracle participates actively on technical committees eg authoring/editing of SQL/MM standards by Dr John Herring.
OpenGIS Standards (Latest)

<table>
<thead>
<tr>
<th>OpenGIS Document Title</th>
<th>Version</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenGIS Implementation Specification for Geographic Information - Simple Feature Access (ISO 19125)</td>
<td>1.2</td>
<td>IS</td>
</tr>
<tr>
<td><em>Part 1: Common Architecture</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supplies the common feature model for use by applications that will use the Simple Features data stores and access interfaces.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Part 2: SQL option</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provides a standard SQL implementation of the abstract model in Part 1 that supports storage, retrieval, query and update of features. Includes Normalised, Binary and “SQL with Geometry Types” 1 (Says nothing about physical storage format) implementation options</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IS - Implementation Specification
DIS - Deprecated Implementation Specification
SAP - Specification Application Profile

ISO Standards (Latest)

<table>
<thead>
<tr>
<th>ISO Document Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO 19107, Geographic information @ Spatial schema</td>
</tr>
<tr>
<td>ISO 19111, Geographic information @ Spatial referencing by coordinates (Implemented in the EPSG collection of geodetic systems)</td>
</tr>
</tbody>
</table>
OGC Standards Compliance

- Both original SDO_* and ST_* implementations have been submitted to standards bodies.

---

**Oracle Corporation**

<table>
<thead>
<tr>
<th>Product Name</th>
<th>OGC Spec</th>
<th>Type</th>
<th>Contact</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Application Server MapViewer, 10g Release 2 (10.1.3)</td>
<td>WMS 1.1.1 (server compliant)</td>
<td>Server and Client</td>
<td>Ravada, Siva</td>
<td>2005-07-26</td>
</tr>
<tr>
<td>Oracle Locator 11g, Release 1 11.1.0.7</td>
<td>SFS(TF) 1.1 (compliant)</td>
<td>Server</td>
<td>Ravada, Siva</td>
<td>2009-09-14</td>
</tr>
<tr>
<td>Oracle Locator, 10g Release 1 (10.1.0.4)</td>
<td>SFS(TF) 1.1 (compliant)</td>
<td>Server and Client</td>
<td>Ravada, Siva</td>
<td>2005-07-26</td>
</tr>
<tr>
<td>Oracle Locator, 10g Release 2 (10.2.0.1)</td>
<td>SFS(TF) 1.1 (server compliant)</td>
<td>Server and Client</td>
<td>Ravada, Siva</td>
<td>2005-11-01</td>
</tr>
<tr>
<td>Oracle Spatial, 10g Release 1 (10.1.0.4)</td>
<td>SFS(TF) 1.1 (server compliant)</td>
<td>Server and Client</td>
<td>Ravada, Siva</td>
<td>2005-07-26</td>
</tr>
<tr>
<td>Oracle Spatial, 10g Release 2 (10.2.0.1)</td>
<td>SFS(TF) 1.1 (server compliant)</td>
<td>Server and Client</td>
<td>Ravada, Siva</td>
<td>2005-11-01</td>
</tr>
<tr>
<td>Oracle Spatial, 11g Release 1 11.1.0.7</td>
<td>WFS 1.0.0 (compliant), WFS(T) 1.0.0 (compliant), SFS(TF) 1.1 (compliant)</td>
<td>Server</td>
<td>Ravada, Siva</td>
<td>2009-09-14</td>
</tr>
<tr>
<td>Oracle Spatial, 9i Release 2 (9.2.0)</td>
<td>SFS(RG) 1.1 (server compliant)</td>
<td>Server and Client</td>
<td>Ravada, Siva</td>
<td>2002-09-30</td>
</tr>
<tr>
<td>Oracle Spatial, release 9i (0.0.1)</td>
<td>SFS(RG) 1.1 (server compliant)</td>
<td>Server and Client</td>
<td>Ravada, Siva</td>
<td>2002-09-30</td>
</tr>
<tr>
<td>Oracle Spatial 8.1.7</td>
<td>SFS(RG) 1.1 (server compliant)</td>
<td>Server and Client</td>
<td>Ravada, Siva</td>
<td>2000-10-24</td>
</tr>
<tr>
<td>Oracle Spatial 8.1.0</td>
<td>SFS(RG) 1.1 (server compliant)</td>
<td>Server and Client</td>
<td>Ravada, Siva</td>
<td>1999-03-17</td>
</tr>
</tbody>
</table>

**Refractions Research Inc**

<table>
<thead>
<tr>
<th>Product Name</th>
<th>OGC Spec</th>
<th>Type</th>
<th>Contact</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostGIS / PostgreSQL 1.1.3 / 8.1.3</td>
<td>SFS 1.1.0 (compliant), SFS(TF) 1.1 (compliant)</td>
<td>Server</td>
<td>Lounsbury, Jeff</td>
<td>2006-08-03</td>
</tr>
</tbody>
</table>
Prefixes and Naming ...

• “ST/ST_” Prefix....
  – Seems to be universally accepted in PostGIS, QSL Server 2008, Oracle SQL/MM type, Informix...
  – OGC SFS 1.2 does not mention it.
  – ISO/TC 211 N 2393 (19125-2), “7.2.2.2 Language constructs” says:

  “Note: Class names in SQL/MM carry a "ST_" prefix. This is optional and implementations may chose to drop this prefix as has been done in various places in this standard.”

  – ISO/IEC 13249 “3.2.2 Notations provided in Part 3” says:

  “This part of ISO/IEC 13249 uses the prefix 'ST_' for user-defined type, attribute and SQL-invoked routine names.”
• Oracle's standard search operators that use spatial indexes are of the following form:
  – SDO_<predicate> eg
    • SDO_ANYINTERACT (ie ST_Intersects)
    • SDO_CONTAINS
    • SDO_COVEREDBY
    • SDO_COVERS
    • SDO_EQUAL
    • SDO_FILTER (Primary Filter)
    • SDO_INSIDE
    • SDO_NN
    • SDO_ON
    • SDO_OVERLAPBDYDISJOINT
    • SDO_OVERLAPBDYINTERSECT
    • SDO_OVERLAPS
    • SDO_RELATE (generic wrapper not 9matrix)
    • SDO_TOUCH
Metadata.....
ISO/TC 211 6.2 Architecture — SQL implementation using Geometry Types, 6.2.1 Overview:

“This standard defines a schema for the management of feature table, Geometry, and Spatial Reference System information in an SQL-implementation with a Geometry Type extension.”
Geometry Columns – The Standard

- Eg OCG (1.2):

```sql
CREATE TABLE GEOMETRY_COLUMNS (  
    F_TABLE_CATALOG CHARACTER VARYING NOT NULL,  
    F_TABLE_SCHEMA CHARACTER VARYING NOT NULL,  
    F_TABLE_NAME CHARACTER VARYING NOT NULL,  
    F_GEOMETRY_COLUMN CHARACTER VARYING NOT NULL,  
    G_TABLE_CATALOG CHARACTER VARYING NOT NULL,  
    G_TABLE_SCHEMA CHARACTER VARYING NOT NULL,  
    G_TABLE_NAME CHARACTER VARYING NOT NULL,  
    STORAGE_TYPE INTEGER,  
    GEOMETRY_TYPE INTEGER,  
    COORD_DIMENSION INTEGER,  
    MAX_PPR INTEGER,  
    SRID INTEGER NOT NULL  
    REFERENCES SPATIAL_REF_SYS,  
    CONSTRAINT GC_PK PRIMARY KEY  
    (F_TABLE_CATALOG, F_TABLE_SCHEMA,  
    F_TABLE_NAME, F_GEOMETRY_COLUMN)
)
```

- For the GEOMETRY_TYPE column, the “use of a non-leaf Geometry class name from the Geometry Object Model for a geometry column implies that domain of the column corresponds to instances of the class and all of its subclasses” [OGC 06-104r3, 7.1.3.3 Field description, Page 29]
### Geometry_Coloums - PostGIS

- **CREATE TABLE** geometry_columns
  
  ```sql
  CREATE TABLE geometry_columns
  (f_table_catalog character varying(256) NOT NULL,
f_table_schema character varying(256) NOT NULL,
f_table_name character varying(256) NOT NULL,
f_geometry_column character varying(256) NOT NULL,
coord_dimension integer NOT NULL,
srid integer NOT NULL,
"type" character varying(30) NOT NULL,
CONSTRAINT geometry_columns_pk PRIMARY KEY
  (f_table_catalog,f_table_schema,
f_table_name, f_geometry_column))
  ```

- **Notes:**
  - Doesn't bother with G_* columns
  - Geometry Type column is named “type” and is a character field not integer.
  - PostGIS's Management Functions for this table eg AddGeometryColumns does not insert “super-type” into “type” when mixed geometry types appear in table as per standard. So, MultiPolygon does not include “Polygon” as it is required to do.
Geometry_Columns - Oracle

- CREATE TABLE MDSYS.OGIS GEOMETRY_COLUMNS ( 
  F_TABLE_SCHEMA    VARCHAR2(64),
  F_TABLE_NAME      VARCHAR2(64),
  F_GEOMETRY_COLUMN VARCHAR2(64),
  G_TABLE_SCHEMA    VARCHAR2(64),
  G_TABLE_NAME      VARCHAR2(64),
  STORAGE_TYPE      NUMBER,
  GEOMETRY_TYPE     NUMBER,
  COORD_DIMENSION   NUMBER,
  MAX_PPR           NUMBER,
  SRID              NUMBER,
  CONSTRAINT FK_SRID FOREIGN KEY (SRID) REFERENCES 
                      MDSYS.OGIS_SPATIAL_REFERENCE_SYSTEMS (SRID) 
)

- There is no global GEOMETRY_COLUMNS view only Oracle-specific USER_GEOMETRY_COLUMNS and ALL_GEOMETRY_COLUMNS public views based on MDSYS.OGC_GEOMETRY_COLUMNS table.

- The MAX_PPR and G_TABLE_SCHEMA/G_TABLE_NAME columns are no longer of any use as Oracle’s implementation of the Normalised model has long been dropped.

  – Note: Oracle does not have concept of a CATALOG so F_TABLE_CATALOG was never supported.

- STORAGE_TYPE should always be NULL = geometry types implementation (OGC SFS SQL 1.2)

- Geometry_Type column is declared as a Number/Integer
PostGIS Management Functions....

In Oracle there are no equivalent Management Functions for metadata management to these in PostGIS (not that these are hard to write):

- **AddGeometryColumn**
  - Adds a geometry column to an existing table.

- **DropGeometryColumn**
  - Removes a geometry column from a spatial table.

- **DropGeometryTable**
  - Drops a table and GEOMETRY_COLUMNS reference.

- **Populate_Geometry_Columns**
  - Ensures geometry column metadata exists in GEOMETRY_COLUMNS and table has appropriate spatial constraints (not requirement of standard).

- **Probe_Geometry_Columns**
  - Scans all tables with PostGIS geometry constraints and adds them to the GEOMETRY_COLUMNS table if they are not there.

- **UpdateGeometrySRID**
  - Updates the SRID of all features in a geometry column, GEOMETRY_COLUMNS metadata and srid table constraint
No Oracle functions know of, or use, MDSYS.OGC_GEOMETRY_COLUMNS
Rather, all use Oracle-specific metadata tables, the most basic being:

CREATE TABLE mdsys.sdo_geom_metadata_table (  
  owner varchar2(32),  
  table_name varchar2(32),  
  column_name varchar2(32),  
  diminfo mdsys.sdo_dim_array,  
  srid number );  

- Needed mainly for creation of indexes.
- Populated by user or client software.

CREATE TYPE sdo_dim_array AS VARRAY(4) OF mdsys.sdo_dim_element;

- Has an sdo_dim_element for each dimension ie X, Y, Z or M

CREATE TYPE sdo_dim_element AS OBJECT (  
  sdo_dimname varchar2(32),  
  sdo_lb number,  
  sdo_ub number,  
  sdo_tolerance number );

- Holds range of all data in table/column for that dimension.
- Some GIS software use diminfo as an accurate extent of all data in table.
- Also, precision (see later) of the data in those ranges.
SDO_DIM_ARRAY - Example

- SELECT *
  FROM user_sdo_geom_metadata
  WHERE table_name = 'TAS_LOCALITY';
Geometry Columns (3)

- Oracle does not automatically synchronise GEOMETRY_COLUMNS as DML is executed against ****_SDO_GEOM_METADATA views.
- Manual DML executed against actual OGC_GEOMETRY_COLUMNS table or views generates errors.
- One approach is to build public view called GEOMETRY_COLUMNS over existing metadata (value-added within functions) as follows:
  
  ```sql
  CREATE VIEW GEOMETRY_COLUMNS AS
  SELECT asgm.owner        as F_TABLE_SCHEMA,
         asgm.table_name   as F_TABLE_NAME,
         asgm.column_name  as F_GEOMETRY_COLUMN,
         NULL              as STORAGE_TYPE,
         Get_Geometry_Type(asgm.owner,
                           asgm.table_name,
                           asgm.column_name) as GEOMETRY_TYPE,
         (SELECT count(*)
          FROM TABLE(asgm.diminfo)
         )                 as COORD_DIMENSION,
         asgm.SRID         as SRID
  FROM ALL_SDO_GEOM_METADATA asgm;
  
  (Note: I have implemented the function Get_Geometry_Type() that returns the correct OGC Geometry_Type -- see my website for details.)

  CREATE PUBLIC SYNONYM geometry_columns FOR codesys.geometry_columns;
  ```
Spatial Reference Systems

- **OGC:**
  - `CREATE TABLE SPATIAL_REF_SYS (SRID INTEGER NOT NULL PRIMARY KEY, AUTH_NAME VARCHAR (256), AUTH_SRID INTEGER, SRTEXT VARCHAR (2048))`

- **Oracle:**
  - `CREATE TABLE MDSYS.OGIS_SPATIAL_REFERENCE_SYSTEMS (SRID NUMBER, AUTH_NAME VARCHAR2 (100), AUTH_SRID NUMBER, SRTEXT VARCHAR2 (1000), SRNUM NUMBER, CONSTRAINT PK_SRID PRIMARY KEY (SRID))`
  - This table is NOT POPULATED and,
  - There is no global view called SPATIAL_REF_SYS based on it.
Spatial Reference Systems

• **Oracle does provide the following table:**
  ```sql
  CREATE TABLE MDSYS.SDO_CS_SRS (
    SRID INTEGER NOT NULL PRIMARY KEY,
    AUTH_NAME VARCHAR2(256),
    AUTH_SRID INTEGER,
    WKTEXT VARCHAR2(2046),
    CS_NAME VARCHAR2(80),
    CS_BOUNDS MDSYS.SDO_GEOMETRY
  )
  ```

• And associated tables such as:
  - SDO_DATUMS, SDO_ELLIPSOIDS, SDOCOORD_AXES,
    SDO_COORD_OPS. etc.

• Oracle's SRS tables are populated by default.
  - Since 10g Oracle's SRS is based on EPSG.

• There is no global view called SPATIAL_REF_SYS defined on this or the previous table.

• Oracle does not automatically synchronise OGC_SPATIAL_REFERENCE_SYSTEMS as DML is executed against mdsys.SDO_CS_SRS and other tables.
We can, however, create our own SPATIAL_REF_SYS view in Oracle as follows:

- `CREATE VIEW SPATIAL_REF_SYS
  AS
  SELECT SRID,
          AUTH_NAME,
          AUTH_SRID,
          WKTEXT AS SRTEXT
  FROM MDSYS.SDO_CS_SRS;`

One could create a global synonym for this view as follows:

- `CREATE PUBLIC SYNONYM spatial_ref_sys
  FOR codesys.spatial_ref_sys;`
- `CREATE PUBLIC SYNONYM spatial_reference_systems
  FOR codësys.spatial_ref_sys;`
• Oracle does not support this aspect of SQL92 standard
  – Needed for some open source software eg ogr
  – Can get a basic implementation from the SourceForge project “Oracle Information Schema” (Lewis Cunningham) at http://sourceforge.net/projects/ora-info-schema/
• This, plus active GEOMETRY_COLUMNS and SPATIAL_REF_SYS objects makes ogr tools like ogrinfo & ogr2ogr work with ODBC driver (don't need compiled OCI version)!
Storage Format and API...
Database Storage Formats...

• Should we care what storage format is used by a database vendor or type manufacturer?
  – While often useful, it is, frankly, irrelevant.
  – Chris Date and Hugh Darwen wrote in their book “Foundation for Future Database Systems: The Third Manifesto”:
    “What we are saying is that, in the relational world, a domain is a data type, system- or user-defined, whose values are manipulable solely by means of the operators defined for the type in question (and whose internal representation can be arbitrarily complex but is hidden from the user).” [Emphasis added by myself]
  – No one really worries about how a number is stored (ie IEEE) within a database as long as we can create, modify, delete and access the data via appropriate languages and standards to a desired precision.
Spatial Database Storage Formats...

- For those that think storage format matters, PostGIS uses “extended” WKB and Oracle uses openly accessible numbers and arrays (SQL/3 components).
- WKT and WKB are provided primarily as interchange and not storage formats.

“The Well-known Binary Representation for Geometry is obtained by serializing a geometric object as a sequence of numeric types drawn from the set {Unsigned Integer, Double} and then serializing each numeric type as a sequence of bytes using one of two well defined, standard, binary representations for numeric types...”
Standards: Orientation & Organisation

- OGC/SQLMM standards also define things like orientation of vertices in a polygon:
  - Anti-clockwise for all outer-shells
  - Clockwise for all inner-shells
- And polygon inversion/exversion and bowties
Oracle's Original UDT Implementation ...

SQL> desc mdsys.sdo_geometry
Name                                      Null?    Type
----------------------------------------- -------- -------------------------
SDO_GTYPE                                          NUMBER
SDO_SRID                                           NUMBER
SDO_POINT                                          MDSYS.SDO_POINT_TYPE
SDO_ELEM_INFO                                      MDSYS.SDO_ELEM_INFO_ARRAY
SDO_ORDINATES                                      MDSYS.SDO_ORDINATE_ARRAY

METHOD
-------
MEMBER FUNCTION GET_GTYPE RETURNS NUMBER

METHOD
-------
MEMBER FUNCTION GET_DIMS RETURNS NUMBER

... etc ...

METHOD
-------
MEMBER FUNCTION ST_COORDDIM RETURNS NUMBER

METHOD
-------
FINAL CONSTRUCTOR FUNCTION SDO_GEOMETRY RETURNS SELF AS RESULT
Argument Name                  Type                    In/Out Default?
------------------------------ ----------------------- ------ --------
WKT                            CLOB                    IN
SRID                           NUMBER                  IN     DEFAULT

... etc ...

• There is no inheritance (no SDO_Polygon, SDO_Point etc)
• Very limited number of methods
• Most “processing” done in PL/SQL packages: SDO_GEOM and SDO_UTIL.
Note: Inheritance
Note: Class names in SQL/MM carry a "ST_" prefix. This is optional and implementations may choose to drop this prefix.
CREATE TYPE ST_Geometry
AS (
    ST_PrivateDimension SMALLINT DEFAULT -1,
    ST_PrivateCoordinateDimension SMALLINT DEFAULT 2,
    ST_PrivateIs3D SMALLINT DEFAULT 0,
    ST_PrivateIsMeasured SMALLINT DEFAULT 0
)
NOT INSTANTIABLE
NOT FINAL

METHOD ST_Dimension()
    RETURNS SMALLINT
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
    RETURNS NULL ON NULL INPUT

CREATE TYPE ST_Point
UNDER ST_Geometry AS
(
    ST_PrivateX DOUBLE PRECISION DEFAULT NULL,
    ST_PrivateY DOUBLE PRECISION DEFAULT NULL,
    ST_PrivateZ DOUBLE PRECISION DEFAULT NULL,
    ST_PrivateM DOUBLE PRECISION DEFAULT NULL
)
INSTANTIABLE
NOT FINAL

METHOD ST_X()
    RETURNS DOUBLE PRECISION
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
    RETURNS NULL ON NULL INPUT

"Standard does not prescribe a particular ADT mechanism, but specifies the behaviour of the ADT through a specification of interfaces that must be supported"
What's in a name

- **UDT** – User Data Type
- **ADT** – Abstract Data Type
- Both refer to a data type that extends the SQL type system.
  - Both can define table column types
    - Stored values are instances of the ADT/UDT.
    - SQL functions may be declared to manipulate ADT/UDT values.
- Difference between implementations is important where you want to inherit from the geometry object as required by "ISO Geometry Object Model"
  - ADTs allow sub-typing, UDTs do not.
  - UDTs generally use existing data types for storage, ADTs can create new storage formats.

(Concrete examples soon...)
Oracle's SQL/MM ADT Implementation

CREATE OR REPLACE TYPE ST_GEOMETRY AS OBJECT (  
    GEOM SDO_GEOMETRY,
    ...  
    MEMBER FUNCTION ST_CoordDim RETURN SMALLINT,
    MEMBER FUNCTION ST_IsValid RETURN INTEGER,
    ...  
    STATIC FUNCTION FROM_WKT(wkt CLOB) RETURN
    ST_GEOMETRY,
    ...
    MEMBER FUNCTION ST_Envelope RETURN ST_Geometry,
    MEMBER FUNCTION ST_GeometryType RETURN VARCHAR2,
    MEMBER FUNCTION ST_Buffer(d NUMBER) RETURN
    ST_Geometry,
    MEMBER FUNCTION ST_Intersects(g2 ST_Geometry)
    RETURN Integer,
    MEMBER FUNCTION ST_Intersection(g2 ST_Geometry)
    RETURN ST_Geometry,
    MEMBER FUNCTION ST_Union(g2 ST_Geometry) RETURN
    ST_Geometry ) NOT FINAL

create or replace TYPE ST_LINESTRING
UNDER ST_CURVE (  
    CONSTRUCTOR FUNCTION
    ST_LINESTRING(apointarray ST_Point_Array)
    RETURN SELF AS RESULT,
    ...
    OVERRIDING MEMBER FUNCTION ST_IsSimple
    RETURN Integer
    ...

CREATE OR REPLACE TYPE ST_CURVE
UNDER ST_GEOMETRY (  
    OVERRIDING MEMBER FUNCTION ST_Dimension
    RETURN Integer,
    MEMBER FUNCTION ST_NumPoints RETURN INTEGER,
    MEMBER FUNCTION ST_PointN(aposition INTEGER)
    RETURN ST_Point,
    MEMBER FUNCTION ST_IsClosed RETURN Integer,
    MEMBER FUNCTION ST_MidPointRep RETURN
    ST_Point,
    MEMBER FUNCTION ST_IsRing RETURN Integer,
    OVERRIDING MEMBER FUNCTION ST_IsSimple
    RETURN Integer,
    MEMBER FUNCTION ST_EndPoint RETURN ST_Point,
    ...  
)

NOT FINAL
Indexing...

- ST_* search functions like ST_Intersects are NOT indexed in Oracle.
  - Only underlying SDO_Geometry object.
- So...

  ```sql
  SELECT *
  FROM <table> a
  WHERE a.geometry.ST_Intersects(<search geometry>) = 1;
  - Will not use Rtree index.
- But...

  ```sql
  SELECT *
  FROM <table> a
  WHERE SDO_Filter(a.geometry.geom,<search_geometry>) = 'TRUE'
    AND a.geometry.ST_Intersects(<search geometry>) = 1;
  - Will use index and be efficient.
Precision Model...
Precision Model

• An important aspect of Oracle Spatial for PostGIS users is in understanding Oracle's precision model.

• There is a lot written about Oracle's precision model that is wrong. For example:
  – I come from the ESRI and Oracle world. Both ArcSDE and Oracle Spatial have user-defined spatial tolerance for each spatially enabled layer. This ensures that coordinates are exact, down to the last decimal (or integer for ArcSDE).

• That Oracle Spatial has a spatial tolerance associated with each sdo_geometry column in a table (which is not a layer) is correct.

• Strictly speaking, as the Oracle documentation points out, a tolerance is not the same as coordinate precision!
Many think Oracle's tolerance describes the precision of an actual ordinate.

- That is if the tolerance is 0.05, an ordinate 123.45678 should actually be 123.5.

However, the Oracle documentation describes tolerance as:

"Tolerance reflects the distance that two points can be apart and still be considered the same (for example, to accommodate rounding errors)."

- This is different from an exact number of digits in an ordinate.

A tolerance of 0.05 means 5cm between two vertices:

- If the distance between the ordinates is less than that the vertices are considered to be equal.
- So, if the actual distance between geometries is 0.846049894.
  - An SDO_TOLERANCE value of 0.005 will cause the Oracle SDO_Distance function to return a distance of 0.846049894.
  - While an SDO_TOLERANCE value of 0.5 will return 0.0.

(Oracle's documentation tells users to set tolerances to be half the actual real world tolerance: so, 0.05 means 0.1m. For those who know how rounding is traditionally done in the C language, this is why tolerances are specified in this way.)
Precision Model - Reality

• You can store anything in the number that make up an ordinate of a geometry!

```sql
SELECT mdsys.OGC_LineStringFromText('LINESTRING(1.123456789 1.3445837283728232, 2.4322323534 2.232303998398)',NULL).Get_WKT() |
    as geom
FROM dual a;
```

GEOM
-------------------------------------------
LINESTRING (1.123456789 1.3445837283728233, 2.4322323534 2.232303998398)

• Oracle has no automatic mechanism for applying the tolerance stored in USER_SDO_GEOM_METADATA during transactions such that the ordinates are rounded to a stated precision.

• It is up to your client application or your own programming of triggers to ensure that ordinate precision remains exact: some do, some don't.
Having said all that, in my programming of Oracle (see my free PL/SQL packages) I actually take the second view in how I handle the comparison of co-ordinates.

- I prefer to round precisely because when I view the data in textual form (ST_AsText etc) I want to see that it is stored to a stated ordinate (numeric) precision.
- So, in my packages, I have programmed a function called Tolerance (with wrapper called ST_SnapToGrid) which will round the ordinates to the stated precision.

In the following, you will note that I can construct a geometry with any number of digits but you have to write a function yourself to round them to your data's actual precision (in this case 1cm):

```sql
SELECT ST_Geom.ST_SnapToGrid(a.geom,0.005).GET_WKT() as geom
FROM (SELECT mdsys.OGC_LineStringFromText('LINESTRING(1.12345 1.3445,2.43534 2.03998398)',NULL) as geom
FROM dual) a;
```

GEOM
---------------------------------------
LINESTRING (1.12 1.34, 2.44 2.04)

To do this is PostGIS you need to use use ST_SnapToGrid():

```sql
SELECT ST_AsText(ST_SnapToGrid(a.geom,0.05,0.05)) as geom
FROM (SELECT ST_GeomFromText('LINESTRING(1.12345 1.3445,2.43534 2.03998398)',0) as geom
FROM dual) a;
```

geom
text
------------------------
LINESTRING(1.1 1.35, 2.45 2.05)
Programming...
I do all my programming of Oracle using PL/SQL and the standard SDO_Geometry data type.

However, it is perfectly possible to minimise the effort required when switching between Oracle and PostGIS.

- For example, if we want the first vertex of a linestring geometry in Oracle (no native Oracle function):

  ```sql
  SELECT MDSYS.SDO_GEOMETRY(2001,NULL,
      SDO_POINT_TYPE(v.x,v.y,v.z),NULL,NULL) as first_point
  FROM TABLE(
      MDSYS.SDO_UTIL.GETVERTICES(
          MDSYS.SDO_GEOMETRY(2002,NULL,NULL,
              MDSYS.SDO_ELEM_INFO_ARRAY(1,2,1),
              MDSYS.SDO_ORDINATE_ARRAY(1,1,2,2)))) v
  WHERE rownum < 2;
  ```

- With PostGIS this is easy – use the ST_StartPoint function:

  ```sql
  SELECT ST_AsText(ST_StartPoint(ST_LineFromText('LINESTRING(1 1,2 2)'),28355));
  ```
How do we bring these two approaches together?

- Well, one way is to use Oracle's ST_Geometry implementation as it contains an ST_StartPoint method:

  ```sql
  SELECT MDSYS.OGC_AsText(mdsys.OGC_LinestringFromText('LINESTRING(1 1,2 2)',28355).ST_StartPoint())
  FROM DUAL;
  ```
  
or
  ```sql
  SELECT TREAT(MDSYS.ST_Linestring.From_WKT('LINESTRING(1 1,2 2)',28355) as MDSYS.ST_LineString).ST_StartPoint().Get_WKT()
  FROM dual;
  ```

- But what if the function doesn't exist in Oracle's SQL/MM implementation e.g. PostGIS's ST_RemovePoint?

  ```sql
  geometry ST_RemovePoint(geometry linestring, integer offset);
  ```

- Then I use PL/SQL to implement a function.
  - I use native Oracle methods to implement the function but
  - I include two overloaded methods:
    - One for the native SDO_Geometry type
    - The other using Oracle's ST_Geometry type
• CREATE OR REPLACE PACKAGE GEOM
  AUTHID CURRENT_USER
  AS
  ...
  Function SDO_RemovePoint(p_geometry IN MDSYS.SDO_Geometry,
                            p_position  IN Number)
    Return MDSYS.SDO_Geometry Deterministic;
  Function ST_RemovePoint(p_geometry IN MDSYS.ST_Geometry,
                          p_position  IN Number)
    Return MDSYS.ST_Geometry Deterministic;
  ...
END Network;

• CREATE OR REPLACE PACKAGE BODY GEOM
  AS
  ...
  Function ST_RemovePoint(p_geometry IN MDSYS.ST_Geometry,
                          p_position  IN Number)
    RETURN MDSYS.ST_Geometry
  Is
  Begin
    Return MDSYS.ST_Geometry.FROM_SDO_GEOM(
      SDO_RemovePoint( p_geometry.GET_SDO_GEOM(),
                       p_position ));
  End ST_RemovePoint;

• Where SDO_RemovePoint is the function that is written using native SDO_Geometry processing and methods.
Dot Notation...

- PostGIS is not implemented as an inheritable type system so one executes methods on a geometry object as follows:
  
  ```sql
  SELECT ST_Length(ST_LineFromText('LINESTRING(1 1,2 2)',28355));
  ```

- With Oracle, if you use the ST_* type system you have to use “dot” notation:

  ```sql
  SELECT mdsys.OGC_LineStringFromText('LINESTRING(1 1,2 2)',28355).ST_Length()
  FROM DUAL;
  ```

- But if you use the ordinary SDO_Geometry, while there are a limited set of methods for the type most processing is done using utility functions.

  ```sql
  SELECT mdsys.sdo_geom.Sdo_Length(mdsys.sdo_geometry('LINESTRING(1 1,2 2)',28355),0.05)
  FROM DUAL;
  ```
Hiding names....

- Don't like “mdsys.sdo_geom.sdo_length”? Then hide it:
  
  ```
  create function ST_Length( p_geometry  in sdo_geometry, 
  p_tolerance in number ) 
  return number DETERMINISTIC
  As
  Begin
  Return 
  mdsys.sdo_geom.sdo_length(p_geometry,p_tolerance);
  End ST_Length;
  ```

- Which you can use as follows:
  
  ```
  SELECT ST_Length(sdo_geometry('LINESTRING(1 1,2 2)',28355),0.05) 
  FROM DUAL;
  ```

- This “looks” a lot more like PostGIS
- Could be done for all Oracle packaged functions that are functionally the same.
ST_* Issue...

• Now, when one database implements things “properly” the other causes “problems”.

• For example, in Oracle the SQL/MM functions ST_GeometryN() and ST_NumGeometries() does not exist!

• In PostGIS one would like to write (but can't):

```sql
SELECT ST_GeometryN(m.mline, p.*) as Line
FROM (SELECT ST_MLineFromText('MULTILINESTRING((1 1,2 2),(3 3,4 4))', 28355) as mline
    ) m,
generate_series(1,ST_NumGeometries(m.mline),1) p;
```

• One can do this in Oracle because they have implemented an ST_Geometries method in ST_Geometries that returns an array of Geometries:

```sql
SELECT b.*
FROM TABLE(SELECT a.geom.ST_Geometries()
    FROM (SELECT mdsys.OGC_MultiLineStringFromText('MULTILINESTRING((1 1,2 2),(3 3,4 4))', 28355) as geom
    FROM dual) a
    ) b;
```

• This plays to Oracle's strengths but isn't an implementation of the SQL/MM standard.
Complain or....

• To the lack of ST_GeometryN and OGC_MultiLineStringFromText we can:
  – Complain....
  – Or do something about it.

• Do the former, but implement the latter:

  ```sql
  create or replace function ST_GeometryN
  ( p_geometry in mdsys.ST_GeomCollection,
    p_num      in integer )
  return mdsys.st_geometry deterministic
  as
    v_geom mdsys.st_geometry;
  begin
    SELECT c.geom
    INTO v_geom
    FROM (SELECT rownum as rin,
      mdsys.ST_Geometry.From_SDO_Geom(g.geom)
      as geom
      FROM TABLE(SELECT p_geometry.ST_Geometries()
      FROM DUAL
    ) g
    WHERE rin = p_num;
    RETURN v_geom;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN NULL;
  end ST_GeometryN;
  ```
Complain (2)...

- **ST_NumGeometries:**
  Create Function `ST_NumGeometries`
  
  ```
  p_geometry in mdsys.ST_GeomCollection 
  Return Integer Deterministic 
  As 
  v_count integer; 
  BEGIN 
  SELECT count(*) 
  INTO v_count 
  FROM TABLE(SELECT p_geometry.ST_Geometries() FROM DUAL) g; 
  RETURN v_count; 
  EXCEPTION 
  WHEN NO_DATA_FOUND THEN 
  RETURN NULL; 
  End ST_NumGeometries;
  ```

- **Throw in some public synonyms:**
  ```
  create public synonym ST_LinestringFromText
  for mdsys.OGC_LinestringFromText;
  create public synonym ST_MultiLinestringFromText
  for mdsys.OGC_MultiLinestringFromText;
  ```

- **And it all starts to look just a bit... familiar!**
  ```
  SELECT ST_GeometryN(b.mline,n.column_value)
  FROM (SELECT ST_MultiLineStringFromText(
    'MULTILINESTRING((1 1,2 2),(3 3,4 4))',
    28355)
  as mline
  FROM dual ) b,
  TABLE(codesys.geom.generate_series(1,
    ST_NumGeometries(b.mline),1)) n;
  ```
Complain (3)

- Oracle's implementation of ST_Geometry is declared NOT FINAL so, theoretically, it would be possible to extend the type system as follows:

```sql
ALTER TYPE mdsys.ST_GeomCollection CASCADE
ADD MEMBER FUNCTION
ST_GeometryN ( p_geometry in mdsys.ST_GeomCollection,
                 p_num in integer )
  RETURN  mdsys.ST_Geometry DETERMINISTIC,
ADD MEMBER FUNCTION
ST_NumGeometries ( p_geometry in mdsys.ST_GeomCollection )
  RETURN Integer DETERMINISTIC;
```

- But one might meet support issues with Oracle.
Framework/Database issues....

- Programmatic problems often have nothing to do with the spatial data type.
- For example, one can, in a SELECT statement, in PostGIS you cannot call a function (generate_series) using the values from a table (m).
  
  ```sql
  SELECT ST_PointN(m.line,p.*) as point
  FROM (SELECT ST_LineFromText('LINESTRING(1 1,2 2)',28355) as mline
         ) m,
     generate_series(1,ST_NPoints(m.mline),1) p;
  ```

  As you get this error (what is called "Functional Row Expansion"):
  
  ERROR: function expression in FROM cannot refer to other relations of same query level

- Whereas, in Oracle, this is not a problem:

  ```sql
  SELECT a.geom.ST_PointN(g.COLUMN_VALUE)
  FROM (SELECT mdsys.OGC_LineStringFromText(    
    'LINESTRING(1 1,2 2)', 28355)
       as geom
       FROM dual
     ) a,
     TABLE(codesys.geom.generate_series(1,a.geom.ST_NumPoints(),1)) g
  ```
Issues (2)

• pg/PLSQL is like PL/SQL but it is not the same!
• Can't overload functions/procedures in Oracle as you can in PostgreSQL
  – PACKAGEd functions can be overloads
    • Only EnterpriseDB has packages!
• Casting is a part of life in PostGIS but you can only do it via the CAST() SQL function in Oracle.
• SELECT ... FROM DUAL;
• CHECK constraint limitations (can't do this in Oracle):
  – CHECK ( ST_Area(the_geom) > 10 )
• SQL Analytics, rownum, TABLE() ....
• Materialised Views, Schemas/Tablespaces...
• Redo and undo logs, nologging, direct path inserts...
  – The list is endless!
Open/Closed Source...

- Oracle may be closed source but your code can be open source...
  - I make my PL/SQL code available for free.
- **Lewis's INFORMATION_SCHEMA** on SourceForge is a good example.
- So, share it around!
Summary...

- To know how to port from one database to the other or support both in a production environment demands knowledge of each product.
- The rich set of tools any database provides offers much scope for improving portability: views, functions, synonyms etc.
- I have given you some methods for increasing portability of the spatial side of Oracle/PostGIS:
  - Synonyms, views, function wrappers, ST_* type etc;
- However, the majority of issues are not spatial
  - The spatial “design pattern” is pretty standard, it's just the names used that cause “problems”!
  - Major issues are endemic:
    - i.e., fundamentally a part of a database's architecture.
Questions...

- Thank you for being patient....

Any questions?