Using SQL and Spatial Types with the Geodatabase

Juliette Gutierrez, Shannon Shields

ESRI Redlands
Using SQL with Spatial Types with the Geodatabase: Agenda

- Introduce spatial types supported by Enterprise Geodatabases
- Discuss working with spatial types
- Demonstrate how to use spatial types with the Geodatabase
Intended Audience

• Beginning to Intermediate user of ArcGIS
  – including ArcSDE

• A general understanding of how SQL is used
  – SQL is a standard query language used to request information from a database.

• Helpful
  – Experience writing SQL statements
Spatial types are just another geometry storage option.
ArcGIS and Spatial Types

• Geodatabases provide a comprehensive GIS model
  – Objects with properties and behavior
  – Rich geoprocessing capability
  – Well defined schema for modeling spatial relationships
  – Data storage independence

• Spatial Types enhance the capabilities of an Enterprise Geodatabase
  – Direct SQL access to the geometry attribute
  – Well-defined, industry-standard storage model and API
Benefits of Using Spatial Types

• Efficiency
  – Spatial data and functions are stored in the database
  – Spatial operations are encapsulated within the type
  – Applications access native dbms type – no mapping layer

• Accessed using common API’s and SQL
  – C, C++, C#, Java, OLEDB, Oracle PL/SQL, T-SQL, etc.
  – SQL extensions enable spatial types and functions in DML and “SELECT” statement syntax
Spatial Types Supported by ArcSDE

- **IBM DB2**
  - `ST_GEOMETRY` (IBM with ESRI)

- **IBM Informix**
  - `ST_GEOMETRY` (Informix with ESRI)

- **Oracle**
  - `ST_GEOMETRY` (ESRI)
  - `SDO_GEOMETRY` (Oracle)

- **PostgreSQL**
  - `ST_GEOMETRY` (ESRI)
  - `GEOMETRY` (PostGIS by Refractions Research)
    - Version 8.3.0
    - New at ArcGIS 9.3

- **Microsoft SQL Server**
  - `GEOMETRY` (MS)
  - `GEOGRAPHY` (MS)
    - Version 2008
    - New at ArcGIS 9.3 *
ST_GEOMETRY

- OGC/ISO compliant
- Supports elevations and measures
- Spatial type stores geometry as compressed shape
  - Less data transfer over network
  - No conversion required in geodatabase
  - High performance
- Default storage option for Oracle and PostgreSQL (ArcGIS 9.3)
  - Geometry storage is selected through configuration keywords (DBTUNE)
- Storage option used by Informix and DB2
- SQL API syntax varies (slightly) by DBMS
OGC & ISO Spatial Type Specifications

- Simple Features
  - Points, lines, polygons, multipart
  - Stored as an attribute (column)
  - Well known interchange formats: WKT, WKB

- Spatial Reference
  - ASCII text definition
  - Based on POSC/EPSG model

- Metadata Tables
  - GEOMETRY_COLUMNS
  - SPATIAL_REF_SYS
Three Main Components of a Spatial Type

1. A data type for storing geometry
   - ST_GEOMETRY, SDO_GEOMETRY, GEOGRAPHY, etc.

2. A spatial index
   - For efficient data retrieval
   - Implementation specific

3. Functions (aka operators, predicates or methods)
   - Create new geometry: Constructor
   - Perform spatial operations: Relational
   - Return property of a geometry: Accessor
   - Transform from one geometry to another: Geometry
Working with Spatial Types

• Creating new feature classes
• Registration
• Inserting data with SQL
  – Constructor functions
• Using spatial types
  – Assessor functions
  – Relationship functions
  – Geometry functions
Creating a New Spatial Type Feature Class

- ArcGIS Desktop
- Geoprocessing tools
- ArcSDE admin commands
- SQL API
Registering 3rd party tables with ArcGIS

- Must register external tables with ArcSDE to see spatial content

- `sdelayer` command: Register table with ArcSDE

```
C:\>sdelayer -o register -l roads,shape -C ID, SDE -e l+n -t GEOGRAPHY -s world -i 5151 -u gisuser ...
```

- ArcGIS Desktop
  - Register with Geodatabase
  - Register as Versioned
  - Grant privileges
Inserting New Features with SQL: Constructor Functions

- Used to build a geometry
  - Input:
    - Well known representation of geometry
    - Spatial reference identifier
  - Returns:
    - A geometry
- Examples
  - ST_Geometry, ST_Point, ST_LineFromWKB, ...

```sql
INSERT INTO parcels (objectid, shape) VALUES 
(r118.nextval, SDE.ST_Geometry('polygon((3 3, 4 6, 5 3, 3 3))',8))
```
Creating data with the SQL API

Registering spatial table with ArcSDE

```sql
CREATE TABLE demo (
    objectid NUMBER(10) NOT NULL,
    name VARCHAR2(100),
    shape VARCHAR2(100)
);
Table created.
```
Accessor Functions

- Used to get a specific property of a geometry
  - Input:
    - A geometry
  - Returns:
    - The geometry property
- Examples
  - ST_Area
  - ST_SRID
  - ST_AsText
  - ST_AsBinary...
Relational Functions

• Tests the spatial relationship between two geometries
  – Input:
    • Two geometries to be compared
  – Returns:
    • True/False (true if the condition applies)

• Examples
  – ST_Contains
  – ST_Disjoint
  – ST_Within...

```sql
SELECT b.building_name, b.shape
FROM buildings b, parcels p
WHERE p.zipcode = 92373
AND SDE.ST_Contains(p.shape, b.shape) = 1
```
ST_Disjoint / ST_Intersects

ST_Disjoint(g1 ST_Geometry, g2 ST_Geometry) : Integer
ST_Intersects(g1 ST_Geometry, g2 ST_Geometry) : Integer

Note: ST_Intersects = NOT ST_Disjoint
Color differences must be pronounced or they will not be apparent to the audience. Use green versus yellow or orange or white. Avoid red which does not look good or show up well on projected slides. Avoid blue when it needs to pop out at you because the slides are mostly blue already.

Thomas Dunn, 7/28/2008
ST_Within / ST_Contains

ST_Within(g1 ST_Geometry, g2 ST_Geometry) : Integer
ST_Contains(g2 ST_Geometry, g1 ST_Geometry) : Integer
Geometry Functions

• Builds a new geometry from one or more existing geometries
  – Input:
    • One or more geometries
    • Sometimes additional arguments
  – Returns:
    • A geometry
• Examples
  – ST_Buffer, ST_ConvexHull, ST_Difference…

```sql
SELECT b.building_name, b.shape
FROM buildings b, parcels p
WHERE p.apn = '450-5982-139'
AND ST_Intersects(b.shape,ST_Buffer(p.shape, 1000))= 1
```
ST_Buffer

ST_Buffer(g1 ST_Geometry, dist double precision) : ST_Geometry
ST_Convexhull

ST_Convexhull(g1 ST_Geometry) : ST_Geometry
Demo:
How to use Accessor, Relational and Geometry functions
Using Spatial Types with the Geodatabase

- Using SQL within ArcGIS Desktop
- Editing ArcGIS feature classes with SQL
  - Non-versioned editing
  - Versioned editing and multiversioned views
- Writing efficient SQL
Demo: Definition query
Using SQL to Edit Spatial Data in a Geodatabase

- A Geodatabase contains a rich set of functionality
  - More than just feature classes

See ArcGIS Desktop Help topic: “An overview of working with ST_Geometry storage using SQL”
Editing with SQL

- Edit simple features only: Points, lines, polygons; single or multipart

- Using SQL to access the geodatabase bypasses geodatabase functionality

- Do not update records using SQL after the data has been versioned unless you use multiversioned views

- You are using SQL directly, so remember to COMMIT changes
Editing Non-versioned Feature Classes

• Edits applied directly to business table
  – No delta tables

• Requires a unique identifier (objectid) when inserting
  – Obtained from classes sequence or procedure

• Can leverage DBMS functionality
  – Unique indexes, constraints, referential integrity, default values, triggers, etc.
Editing Versioned Feature Classes

• Use SQL to access multiversioned views
  – View derives a result set based on a version query
  – Created on a single versioned table and contains all columns
  – Ability to access or edit a specific version
  – Stored procedures installed with ArcSDE geodatabase

• Requires a defined workflow

• More information available in ArcGIS Desktop Help
  – Search for 'multiversioned views'
Working with multiversioned views

- **Create a multiversioned view**
  - sdetable -o create_mv_view
- **Create a version in which to do your edits**
- **From SQL, for each edit session**
  - Select a specific version
  - Start an edit session
  - Perform your edits through the multiversioned view
  - Stop the edit session
- **Use ArcMap to reconcile/post to its parent version**
**Multiversioned Views: Set Current Version**

- DBMS stored procedure sets multiversioned view to use a specific version

<table>
<thead>
<tr>
<th>Database</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Oracle</strong></td>
<td><code>exec sde.version_util.set_current_version ('joe.PROPOSED_SUBDIVISION');</code></td>
</tr>
<tr>
<td><strong>SQL Server</strong></td>
<td><code>exec dbo.set_current_version ('joe.PROPOSED_SUBDIVISION')</code></td>
</tr>
<tr>
<td><strong>DB2</strong></td>
<td><code>call setcurrentversion ('joe.PROPOSED_SUBDIVISION')</code></td>
</tr>
</tbody>
</table>
Multiversioned Views: Edit Version

- DBMS procedures for starting and stopping an edit session on a multiversioned view

**Informix**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EXECUTE FUNCTION sde.edit_version('joe.PROPOSED_SUBDIVISION', 1)</code></td>
<td>Start edit session</td>
</tr>
<tr>
<td><code>EXECUTE FUNCTION sde.edit_version('joe.PROPOSED_SUBDIVISION', 2)</code></td>
<td>Close edit session</td>
</tr>
<tr>
<td><code>--perform edit</code></td>
<td></td>
</tr>
</tbody>
</table>

**PostgreSQL**

<table>
<thead>
<tr>
<th>SQL Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SELECT sde.sde_edit_version('joe.PROPOSED_SUBDIVISION', 1)</code></td>
<td>--perform edit</td>
</tr>
<tr>
<td><code>SELECT sde.sde_edit_version('joe.PROPOSED_SUBDIVISION', 2)</code></td>
<td></td>
</tr>
</tbody>
</table>
Multiversioned Views: Obtaining unique identifiers

• Must populate SDE-maintained ObjectID field with a valid value

**Oracle**

```sql
SELECT registration_id FROM sde.table_registry
WHERE owner = 'JOE' AND table_name = 'PARCELS';

SELECT sde.version_user_ddl.next_row_id('JOE', 114)
FROM dual;
```

**SQL Server**

```sql
SELECT registration_id FROM sde.sde_table_registry
WHERE owner = 'JOE' AND table_name = 'PARCELS'

DECLARE @id AS INTEGER
DECLARE @num_ids AS INTEGER
exec joe.i114_get_ids 2, 1, @id OUTPUT, @num_ids OUTPUT
```
Demo: Editing using a multiversioned view
Writing Efficient SQL

• Do not access more data than you need
  – Fetch only the necessary columns
  – Join only necessary tables

• Define efficient where clauses
  – Avoid DBMS operators which might force full table scans
    • SUBSTR(), INSTR(), UPPER()
  – Make sure appropriate indexes exist on where clause columns

• Prototype your SQL

• Know your execution path/plan
Using SQL with Spatial Types with the Geodatabase: Summary

- Introduced spatial types supported by Enterprise Geodatabases
- Discussed working with spatial types
- Demonstrated how to use spatial types with the Geodatabase
Related Presentations

• Technical workshops / briefings:
  – Working with ESRI’s Spatial Type for Oracle
    • Wed August 6, 12:00pm, room 4
  – Administration for SQL Server – Advanced
    • Tue August 5, 3:15pm, room 4
    • Thu August 7, 10:15am, room 4
  – Administration for PostgreSQL
    • Tue August 5, 12:00pm, room 4
    • Thu August 7, 3:15pm, room 6C
  – Administration for IBM Databases – An Introduction
    • Tue August 5, 12:00pm, room 4
Questions?

Please complete the session survey.