Best Practices for using SQL

Thomas Brown

Kevin Watt

Please!

turn OFF cell phones and refrain from using flash photography

Our assumptions

- Prerequisites: Experience writing SQL
- Our expectations…
  - You have an understanding of the geodatabase
  - You understand spatial relationships
  - You have experience with SQL concepts
- Our objective…
  - Introduce the value of using spatial types
  - Provide insight for understanding SQL performance
  - Highlight new functionality with st_geometry for Oracle

Today's agenda

- Efficient SQL with ArcObjects
- Working with multi-version views
  - Creating, editing and obtaining row_id values
- Working with spatial types
  - What is a spatial type
  - Accessing spatial data
  - Working with relational operators
- Think like an *Optimizer*
  - *Selectivity and cardinality*
  - When is the spatial index used
  - Spatial clustering
- Writing a complex spatial query

Efficient SQL with ArcObjects

Efficient SQL with ArcObjects

- As ArcObject developers you are responsible for writing efficient SQL
  - Fetch only the necessary subfields
  - Join the appropriate tables
  - Define efficient where clauses
  - Set correct postfix clauses

Efficient SQL with ArcObjects

- Example: table join with a complicated where clause

```
IQueryDef queryDef = featureWorkspace.CreateQueryDef();
queryDef.Tables = "ElectricStation, CircuitSource, CableJunction";

queryDef.SubFields = "ElectricStation.StationName,
  ElectricStation.StationAbbreviation, CableJunction.ObjectID";
queryDef.WhereClause = "((ElectricStation.CircuitID LIKE 'A-235%' " +
    "AND SUBSTR(ElectricStation.CircuitID,5+1,1) not between '0' and '9')" +
    "OR (ElectricStation.CircuitID = 'A-235'))" +
    "AND ElectricStation.StationTypeCode = 'GEN'" +
    "AND CircuitSource.ObjectID = ElectricStation.CircuitSourceObjectID" +
    "AND ElectricStation.ObjectID = CircuitSource.ElectricStationObjectID" +
    "AND CableJunction.DeviceObjectID = ElectricStation.ObjectID" +
    "AND CableJunction.DeviceType = 'ENG'" +
    "OR  INSTR(UPPER(CircuitSource.PhaseDesignation),'123') > 0)";

ICursor cursor = queryDef.Evaluate();
IRow row = cursor.NextRow();
```

Efficient SQL with ArcObjects

- Watch out for DBMS operators which might require ***Full table scans***
  - SUBSTR()
  - INSTR()
  - UPPER()
- Utilize function based indexes when applicable
- Sometimes efficient SQL means executing multiple single table statements and generate the final result set within the client application

Efficient SQL with ArcObjects

- IWorkspace.ExecuteSQL
  - Execute arbitrary SQL statements
  - DDL or DML statements (but can not return a result set)
  - Stored procedures

```
IWorkSpace.ExecuteSQL 'BEGIN <stored_procedure_name> ("<arguments>"); END;'

//Within your procedure

  err_num := SQLCODE;
  err_msg := SUBSTR(sqlerrm, 1, 100);
  INSERT INTO <temporary_table> VALUES (err_num, err_msg);
  COMMIT;

//Use ArcObjects to check the return code

  IQueryDef queryDef = featureWorkspace.CreateQueryDef();
  queryDef.Tables = "<temporary_table>"
  queryDef.SubFields = "err_num, err_msg"

  IRow row = cursor.NextRow();
```

Working with multi-versioned views

Working with multi-versioned views

- For enterprise applications which require SQL access to versioned tables
  - Ability to access any version
  - View derives a result set based on a version query
  - Procedures provided with SDE installation

- SDE administration command for creating the view

```
sdetable –o create_mv_view –T <view_name> -t <table_name>
  [-i <service>] [-s <server_name>] [-D <database>]
  –u <DB_User_name> [-p <DB_User_password>] [-N] [-q]

sdetable –o create_mv_view –T parcels_mv –t parcels –i 5151
  –s alex –u tomb -N
```

Working with multi-versioned views

- DBMS procedure for setting the version for the view to reference

```
//Oracle

 SQL> exec sde.version_util.set_current_version ('tomb.PROPOSED_SUBDIVISION');

 SQL> SELECT owner, parcel_id FROM parcel_mv
      WHERE st_envintersects(shape, 5,5,10,10) = 1;

//SQL*Server

 exec sde.set_current_version ('tomb.PROPOSED_SUBDIVISION')
   or
 exec dbo.set_current_version ('tomb.PROPOSED_SUBDIVISION')

//DB2

 call setcurrentversion ('tomb.PROPOSED_SUBDIVISION')
```

Working with multi-versioned views

- DBMS procedures for editing a versioned geodatabase and multi-versioned views

```
//Oracle

 SQL> exec sde.version_user_ddl.edit_version ('tomb.PROPOSED_SUBDIVISION', 1);

 SQL> UPDATE parcel_mv SET owner = 'Ethan Thomas'
      WHERE parcel_id = '322-2002-001' AND st_intersects(shape,st_geom) = 1;
 SQL> COMMIT;

 SQL> exec sde.version_user_ddl.edit_version ('tomb.PROPOSED_SUBDIVISION', 2);


//SQL*Server

 exec sde.edit_version ('tomb.PROPOSED_SUBDIVISION', 1)
 exec dbo.edit_version ('tomb.PROPOSED_SUBDIVISION', 2)
```

Working with multi-versioned views

- DBMS procedures for obtaining row_id values

```
//Oracle

  SQL> SELECT registration_id FROM sde.table_registry
       WHERE owner = 'TOMB' AND table_name = 'PARCELS';

  SQL> SELECT sde.version_user_ddl.next_row_id('TOMB', 114) FROM dual;


//SQL*Server

  SELECT registration_id FROM sde.sde_table_registry
  WHERE owner = 'TOMB' AND table_name = 'PARCELS'

  DECLARE @id AS INTEGER
  DECLARE @num_ids AS INTEGER
  exec sde.i114_get_ids 2, 1, @id OUTPUT, @num_ids OUTPUT
```

Working with multi-versioned views

- Do NOT…
  - Update the objectid (row_id) value
  - Modify geometries for classes participating in topologies or geometric networks
    - Will not create dirty areas or be validated
    - Will not maintain connectivity in the logical network
  - Update attributes which define geodatabase behavior
    - Enabled/Disabled attributes
    - Ancillary attributes
    - Weight attributes
    - Subtypes

Working with spatial types

Working with Spatial Types

It's about…

Spatial Types

- What is a "Spatial Type"
- Accessing Spatial Data
- Working with Relational Operators

It's not about…

Accessing geodatabase objects using SQL

Comparing Spatial Types

Geodatabase / ArcSDE geometry storage formats

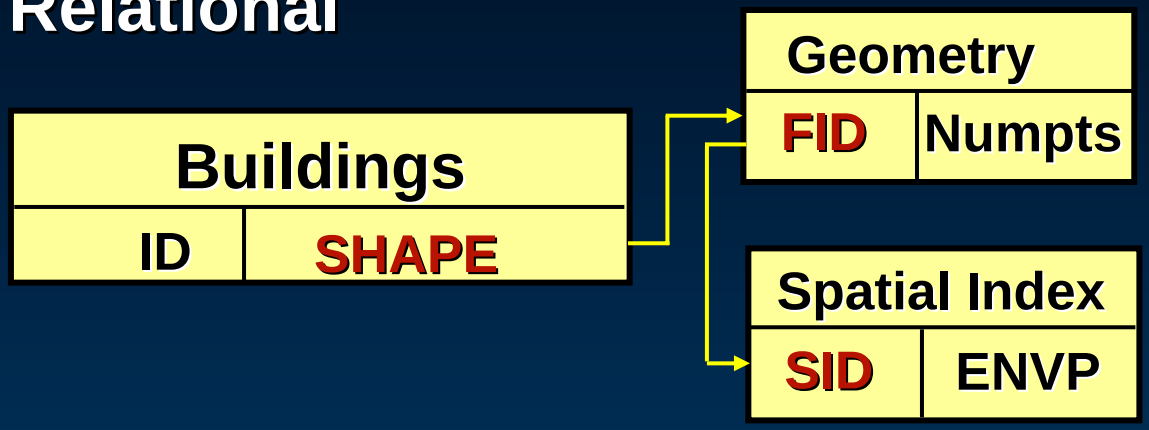
Relational or Object Relational

Storage formats vary by DBMS

– Oracle	
• SDEBinary (LONG RAW) and (LOB)	Relational / Relational
• ST_Geometry	Object Relational
• SDO_Geometry	Object Relational
– SQL Server	
• SDEBinary	Relational
– DB2	
• Spatial Extender	Object Relational
– Informix	
• Spatial DataBlade	Object Relational

Geometry storage formats

Relational




```
SELECT id,a.shape,f.fid
FROM Buildings a, F1 f, S1 s
WHERE s.gx <= :1 AND s.gx >= :2 AND s.gy <= :3 AND s.gy >= :4
        AND minx <= :5 AND miny <= :6 AND maxx >= :7 AND
        maxy >= :8 AND f.fid = s.sid AND a.shape = f.fid
```

Geometry storage formats

Object Relational

Buildings	
ID	SHAPE

```
SELECT b.id, b.shape
FROM Buildings b
WHERE ST_EnvIntersects (b.shape,10, 10, 50, 50) = 1;
```

Easier to work with in SQL than Relational

What is a Spatial Type…

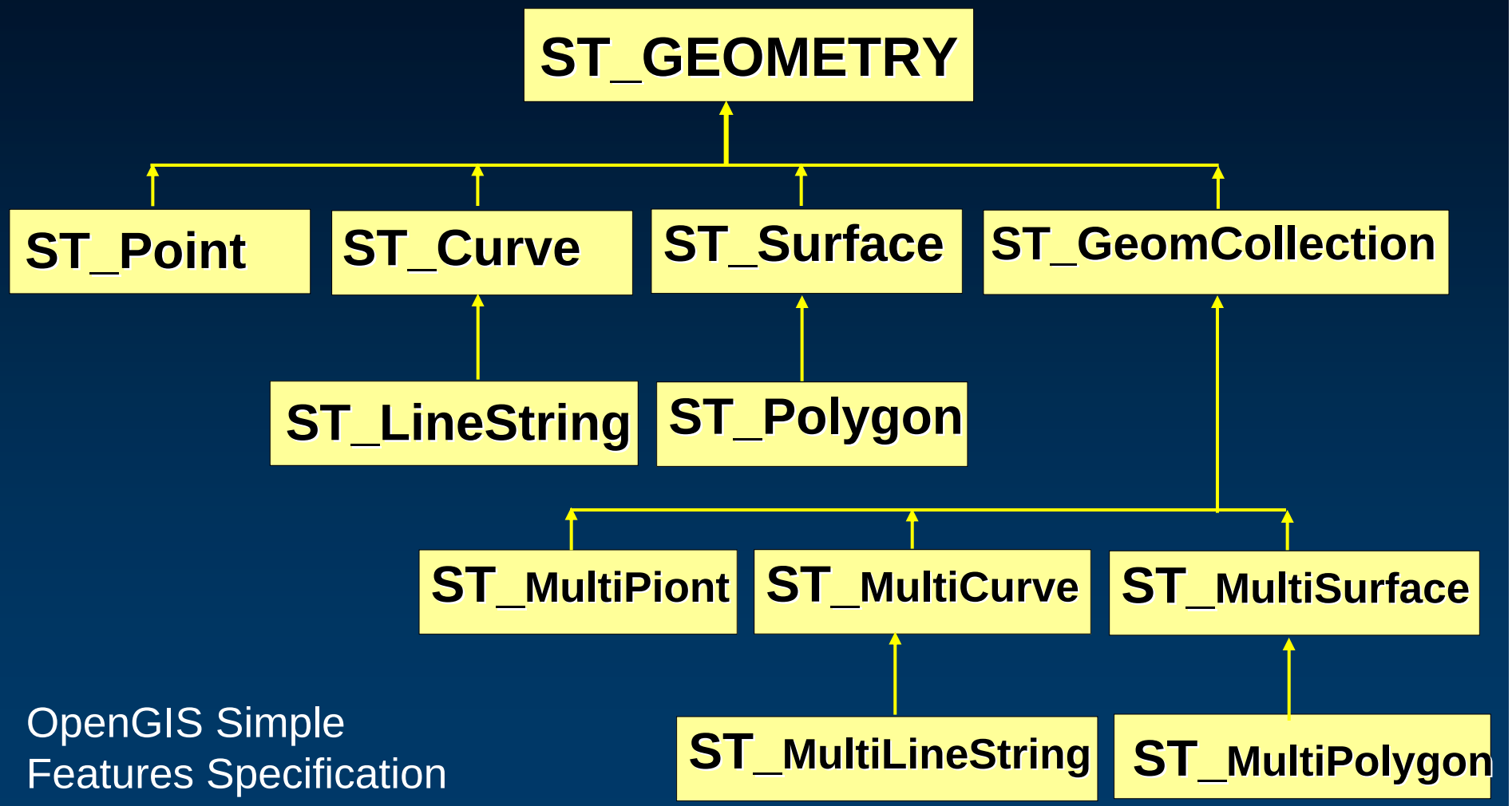
- Object types for geometry data
  - ST_Geometry, SDO_Geometry
- Spatial Index
- Relational and Geometry Operators
  - ST_Relate
  - ST_Intersects
  - ST_Buffer…

Geometry object type properties

ST_GEOMETRY

Numpts	– # of points
Entity	– Geometry type
MinX	– Minimum X
MinY	– Minimum Y
MaxX	– Maximum X
MaxY	– Maximum Y
MinZ	– Minimum Z
MaxZ	– Maximum Z
MinM	– Minimum Measure
MaxM	– Maximum Measure
Area	– Area of polygon
Len	– Length of line/polygon
SRID	– Spatial Reference
Geometry	– coordinates

ST_GEOMETRY object model



OpenGIS Simple Features Specification for SQL

http://www.opengeospatial.org/docs/99-049.pdf

ST_GEOMETRY constructor functions

- Well-Known Binary
  - ST_AsBinary
- Well-Known Text
  - ST_AsText
- ESRI ShapeFile
  - ST_AsShape

ST_GEOMETRY constructors

- Well-Known Binary - ST_AsBinary

```
BEGIN
   FOR item IN
   (
      SELECT area, ST_AsBinary(shape)
      FROM Buildings
      WHERE ST_Area(shape) > 80000
   )
   LOOP
     dbms_output.put_line('Area: '||item.area||'   '||
     ' ST_AsBinary Len:'||dbms_lob.GETLENGTH(item.shape1));
  END LOOP;
END;
```

Area: 92116.13714 ST_AsBinary Len: 237

ST_GEOMETRY constructors

- Well-Known Text - ST_AsText

```
SELECT area, ST_AsText(shape) as shape_wkt
FROM buildings
WHERE ST_Area(shape) < 100;

 AREA      SHAPE_WKT
----------  ----------------------------------------------------------------------------
91.362      POLYGON  (( 2217028.84 399516.70, 2217028.84 399507.82,
  2217039.12 399507.82, 2217039.12 399516.70, 2217028.84
  399516.70))
```

ST_GEOMETRY constructors

- ESRI ShapeFile - ST_AsShape

```
BEGIN
   FOR item IN
   (
      SELECT area, ST_AsShape(shape)
      FROM Buildings
      WHERE ST_Area(shape) > 80000
   )
   LOOP
     dbms_output.put_line('Area: '||item.area||'    '
     ' ST_AsShape Len:'||dbms_lob.GETLENGTH(item.shape1));
  END LOOP;
END;
```

Area: 92116.13714 ST_AsShape Len: 237

Type checking through inheritance

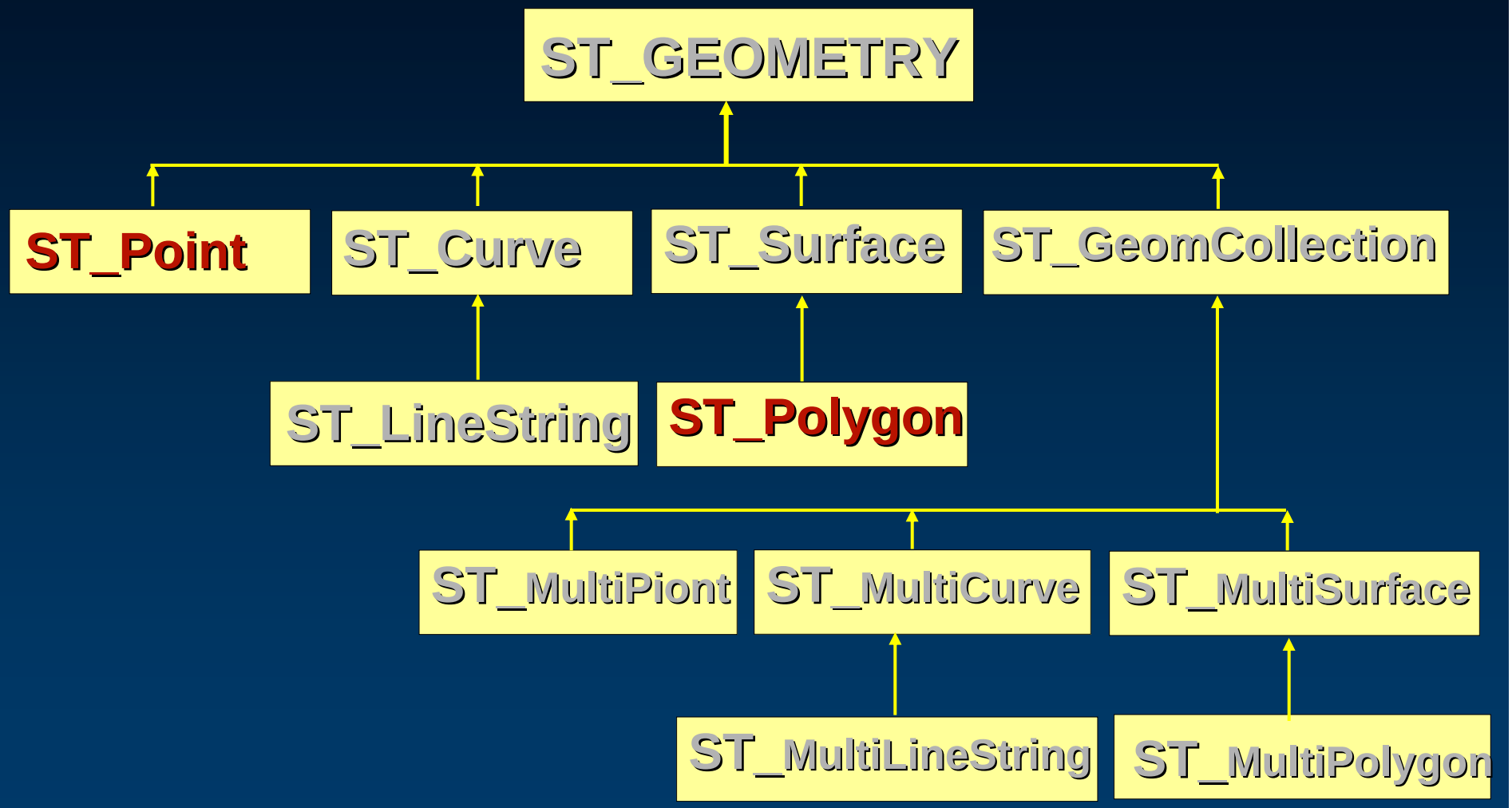
- Strong type checking is used to enforce type correctness at the subtype level

```
SQL> CREATE TABLE test_load (id INTEGER, shape ST_Polygon);

SQL> INSERT INTO test_load
        values (10, ST_Polygon ('polygon ((10 10,10 20,20 20,
                                                20 10,10 10))' ,1));
SQL> 1 row created.

SQL> INSERT INTO test_load (id, shape)
        values (2, ST_Point (50, 50, 1));
```

ST_GEOMETRY object model



Type checking through inheritance

- Strong type checking is used to enforce type correctness at the subtype level

```
SQL> CREATE TABLE test_load (id INTEGER, shape ST_Polygon);

SQL> INSERT INTO test_load
          values (10, ST_Polygon ('polygon ((10 10,10 20,20 20,
                                                 20 10,10 10))' ,1));
SQL> INSERT INTO test_load (id, shape)
          values (2, ST_Point (50, 50, 1));

ERROR at line 1:
ORA-00932: inconsistent datatypes: expected SDE.ST_POLYGON
  got SDE.ST_POINT
```

What is a Spatial Type…

- Object types for geometry data
  - ST_Geometry, SDO_Geometry
- Spatial Index
  - Vendor-specific Format
- Relational and Geometry Operators
  - ST_Relate
  - ST_Intersects
  - ST_Buffer…

Spatial Index

- Uses DBMS Extensible Indexing
- Search Algorithm - Grid or RTree Indexing
- Associated to ST_Geometry type and Operators
  - Provides access path information to the optimizer

What is a Spatial Type…

- Object types for geometry data
  - ST_Geometry, SDO_Geometry
- Spatial Index
- Relational and Geometry Operators
  - ST_Relate
  - ST_Intersects
  - ST_Buffer…

Spatial Type Operators

Geometry

"returns a GEOMETRY or property from the difference, comparison or inspection of geometries"

Relational

"…returns TRUE

if SHAPE1 has a spatial relationship {intersect, touches, contains, within,…} with SHAPE2

else

returns FALSE"

Geometry Operators

- **ST_Buffer** — return Geometry whose distance from Geom1 is less than or equal to a specified distance
- **ST_IsClosed** — return TRUE if Curve or MultiCurve is closed
- **ST_AsText** — return Well-Known Text from Geom
- **ST_Intersection** — return Geometry that represents the point set intersection of Geom1 and Geom2
- **ST_Envelope** — return Geometry that represents the rectangle of Geom1
- **ST_Union** — returns Geometry that represents the point set union of Geom1 and Geom2

Geometry Operators

```
SELECT b.building_name
FROM buildings b, parcels p
WHERE p.pid = 45078
        AND ST_Intersects (ST_Buffer (p.shape, 1000), b.shape)  = 1
        AND ST_Area(b.shape) > 4000;
```

"select building names from Buildings that intersect Parcels having a 1000 meter buffer around parcel 45078 and are greater than 4000 square feet "

More on Geometry Operators visit

http://www.opengeospatial.org/docs/99-049.pdf

Relational Operators

- ST_Contains — TRUE if Geom1 contains Geom2
- ST_Crosses — TRUE if Geom1 crosses Geom2
- ST_Disjoint — TRUE if Geom1- Geom2 do not intersect
- ST_Equals — TRUE if Geom1- Geom2 are the same
- ST_Intersects — TRUE if Geom1- Geom2 intersect
- ST_OrderingEquals — TRUE if Geom1- Geom2 are the same and in the same order
- ST_Overlap — TRUE if Geom1- Geom2 overlap and resultant geometry is the same type
- ST_Relate — TRUE if Geom1- Geom2 meet the conditions of the DE-9IM matrix**
- ST_Touches — TRUE if Geom1- Geom2 touch
- ST_Within — TRUE if Geom1 is completely within Geom2

Relational Operators

```
SELECT b.building_name
FROM buildings b, parcels p
WHERE p.zipcode = 90210
        AND ST_Contains(p.shape,b.shape) = 1
        AND ST_Area (b.shape) > 4000;
```

ST_Contains returns TRUE if p.shape contains b.shape

"select all buildings names from Buildings that are in Parcel ZipCode 90210 and completely contained in the parcel boundaries from 90210 and are greater than 4000 square feet "

Relational Operators

Remember…

- Difference between a.shape and b.shape
- Why FALSE “0” isn’t the opposite of TRUE “1”
- Use ST_Relate to solve simple and complex spatial relationships
- Avoid overloading predicate with multiple spatial operators

Relational Operators

Difference between a.shape and b.shape

Is it… ST_Contains (*a*.shape, *b*.shape)

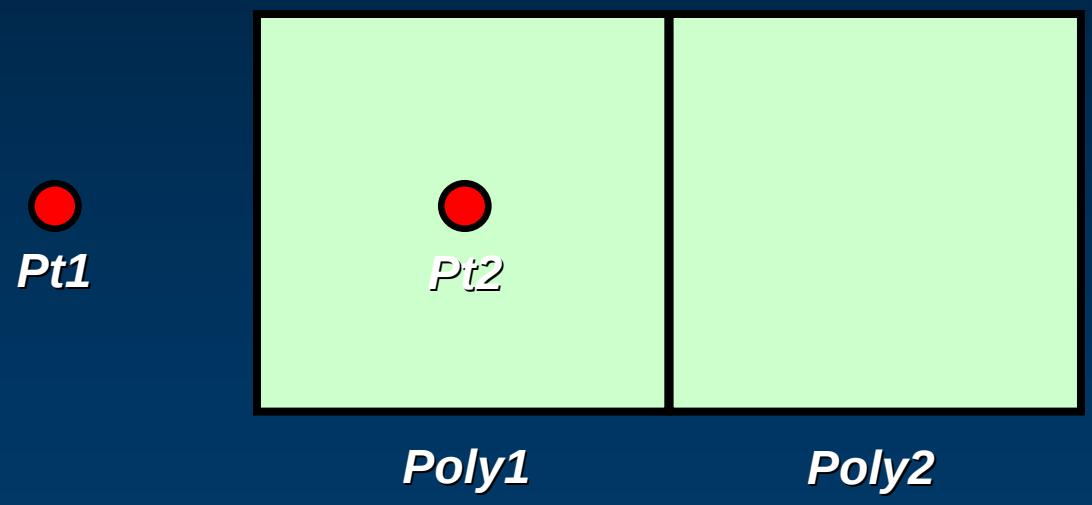
or

ST_Contains (*b*.shape, *a*.shape)

- **Review the operator description**
- **Check join aliases**
- **Test against local test case and *know the results!***

Why FALSE isn't the opposite of TRUE

- Two layers
  - One layer with two points
  - One layer with two polygons

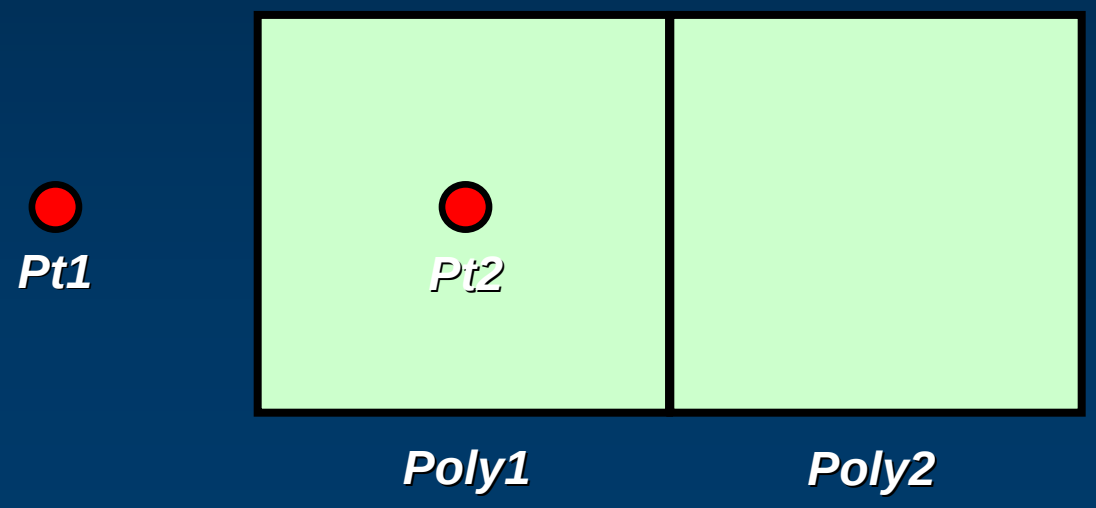


Group Exercise

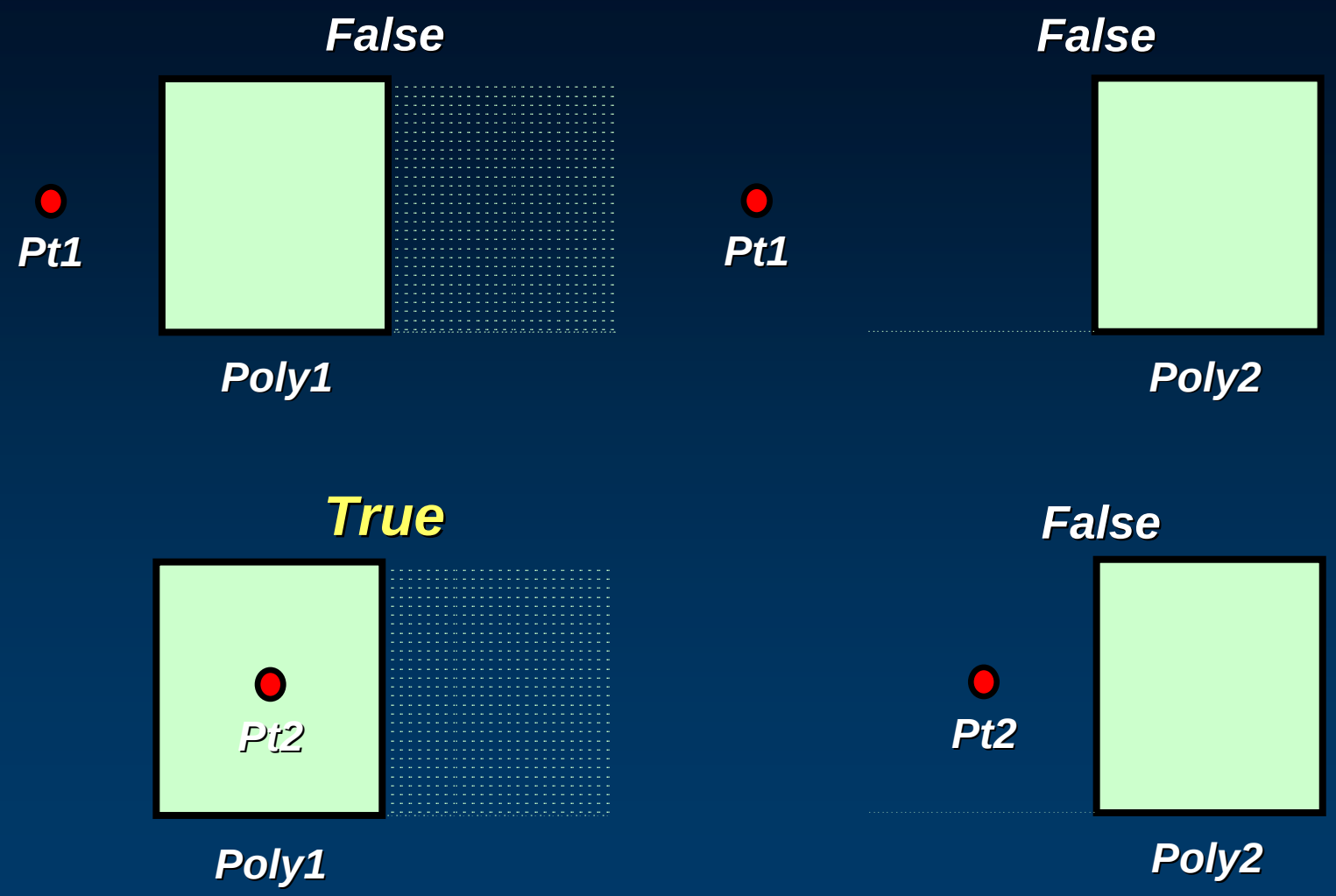
- Question 1

```
SELECT COUNT(*) FROM pnts pt, polys pl
WHERE st_intersects(pt.shape, pl.shape) = 1;
```

  – What is the value for COUNT(*)?



Answer



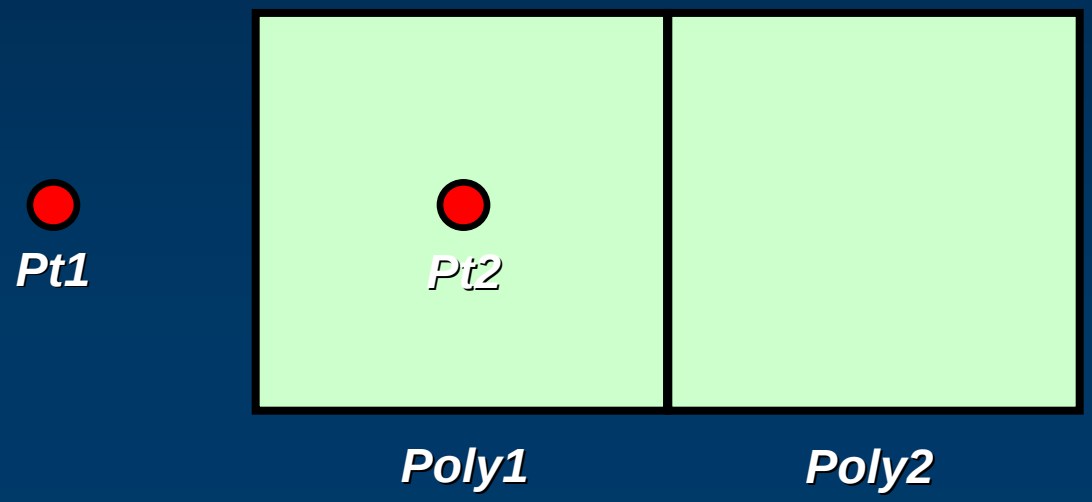
COUNT = 1

Group Exercise

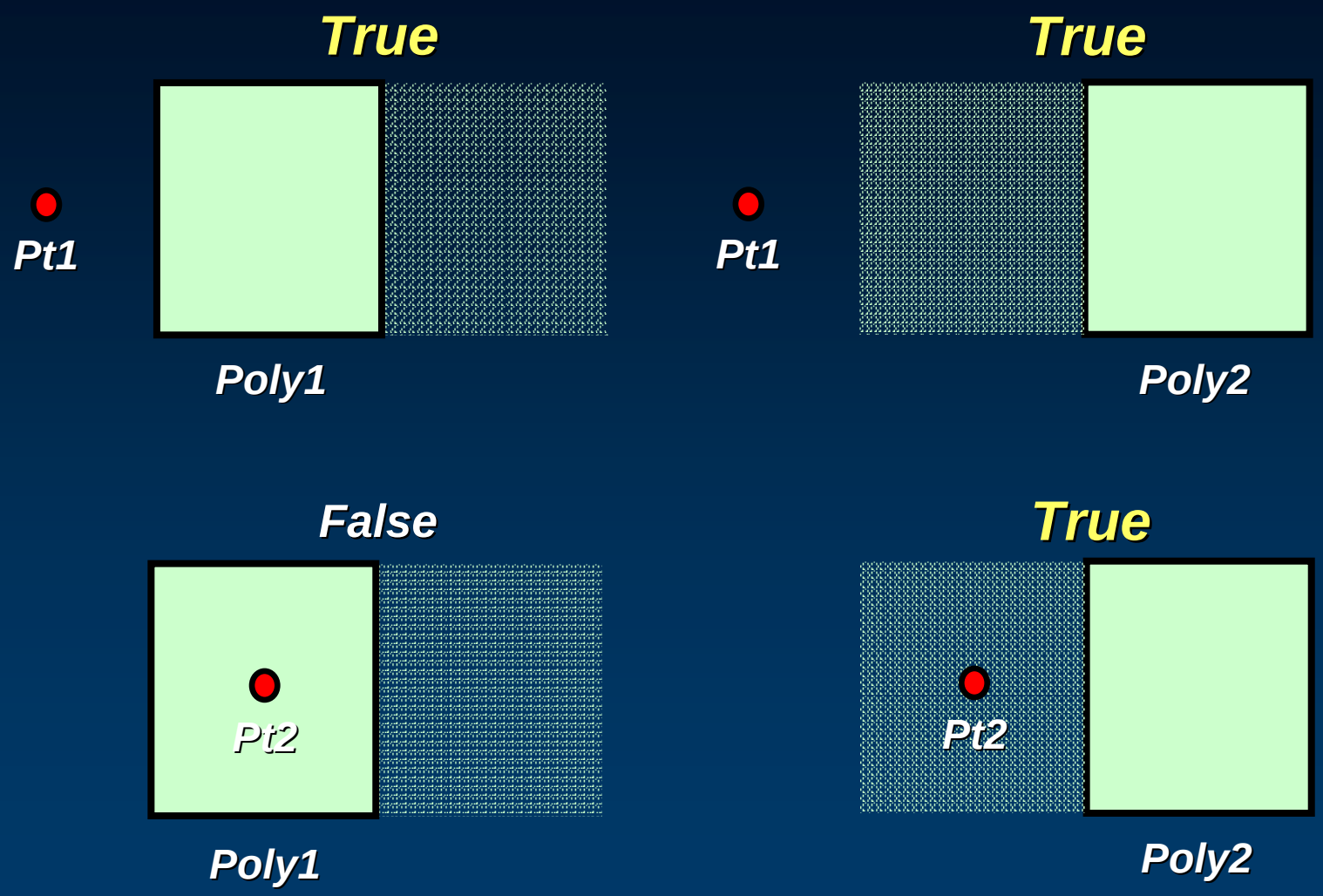
- Question 2

```
SELECT COUNT(*) FROM pnts pt, polys pl
WHERE st_intersects(pt.shape, pl.shape) = 0;
```

  – What is the value for COUNT(*)?



Answer



COUNT = 3

Relational Operators

- ST_Relate
  - Compares spatial relationship using Dimensionally Extended 9 Intersection Model (DE-9IM)
  - Performs better than multiple (intersects,touches,…) predicate filters

Question…

" I want to find all Census Blocks within a Voting District that have a common linear boundary not inside a Voting District…"

Relational Operators - ST_Relate

		SHAPE B		
		Interior	Boundary	Exterior
SHAPE A	Interior	F	*	*
	Boundary	*	1	*
	Exterior	*	*	*

0 Dim Point

1 Dim Line

2 dim Polygon

T – Intersection(0,1,2)

F – no Intersection

"Census Blocks and Voting Dist. interiors do not Intersect "F" but have a common Linear "1" boundary "

```
SELECT c.shape
FROM census_blocks c, voting_districts v
WHERE v.district = 1394
        AND ST_Relate (c.shape,v.shape,'F***1****') = 1;
```

Relational Operators

For more information on DE-9IM format

http://edndoc.esri.com/arcobjects/9.2/CPP_VB6_VBA_VCPP_Doc/shared/reference/shape_comp_lang.htm

“Think” like an *optimizer*?

What is an *optimizer*

- The *optimizer* is the database's mechanism for using heuristics for choosing an execution plan (access path) for a SQL statement

- During the parsing phase the *optimizer* determines access paths, join methods and join orders by calculating a ***COST***

How does the optimizer calculate *COST*

- **During the optimization process, the optimizer discovers the following**
  - ***Selectivity***
  - ***Cardinality***

Selectivity

- Is the fraction of rows from the table which “meet” a predicate’s condition

```
SELECT COUNT(*) FROM parcels WHERE owner = ‘BROWN’
```

  - Parcels table contains 100 rows
  - 10 rows contain a NULL owner
  - 85 distinct owner values

Selectivity

- **Density equation**
  - **1 / number distinct values**
  - **1 / 85 = .011764**

- ***Selectivity* equation**
  - **density * (number rows – number nulls) / number rows**
  - **.011764 * (100 – 10) / 100 = .01**

Cardinality

- Computed ***cardinality*** is the number of rows in the result set after predicates are applied

```
SELECT COUNT(*) FROM parcels WHERE owner = 'BROWN'
```

- ***Cardinality*** equation
  - selectivity * number of rows
  - .01 * 100 = 1

- <u>Crucial for selecting join orders and choice of indexes</u>

Optimizer's *COST*

- Is calculated by the *optimizer* based upon the amount of *i/o* and *cpu* consumed to perform an operation
- Every operation of a SQL statement has a *cost*
  - Each access step
  - Each predicate filter
  - Each join
- *Cost* of *Full table scan* is the baseline
- *Optimizer* selects the best plan based on lowest *cost*

How does the *optimizer* derive values

- Table and Index statistics
  - It is critical to have accurate statistics
  - If the statistics do not represent the true data characteristics… the optimizer *will* choose a plan which is inefficient
  - If the data is dynamic, update statistics

Single predicate example

- How would the *optimizer* determine the best access path for the following statement
  - (attribute owner has a non-unique index)

```
SELECT apn, shape FROM parcels WHERE owner = 'BROWN'
```

Index scan?

Full table scan?

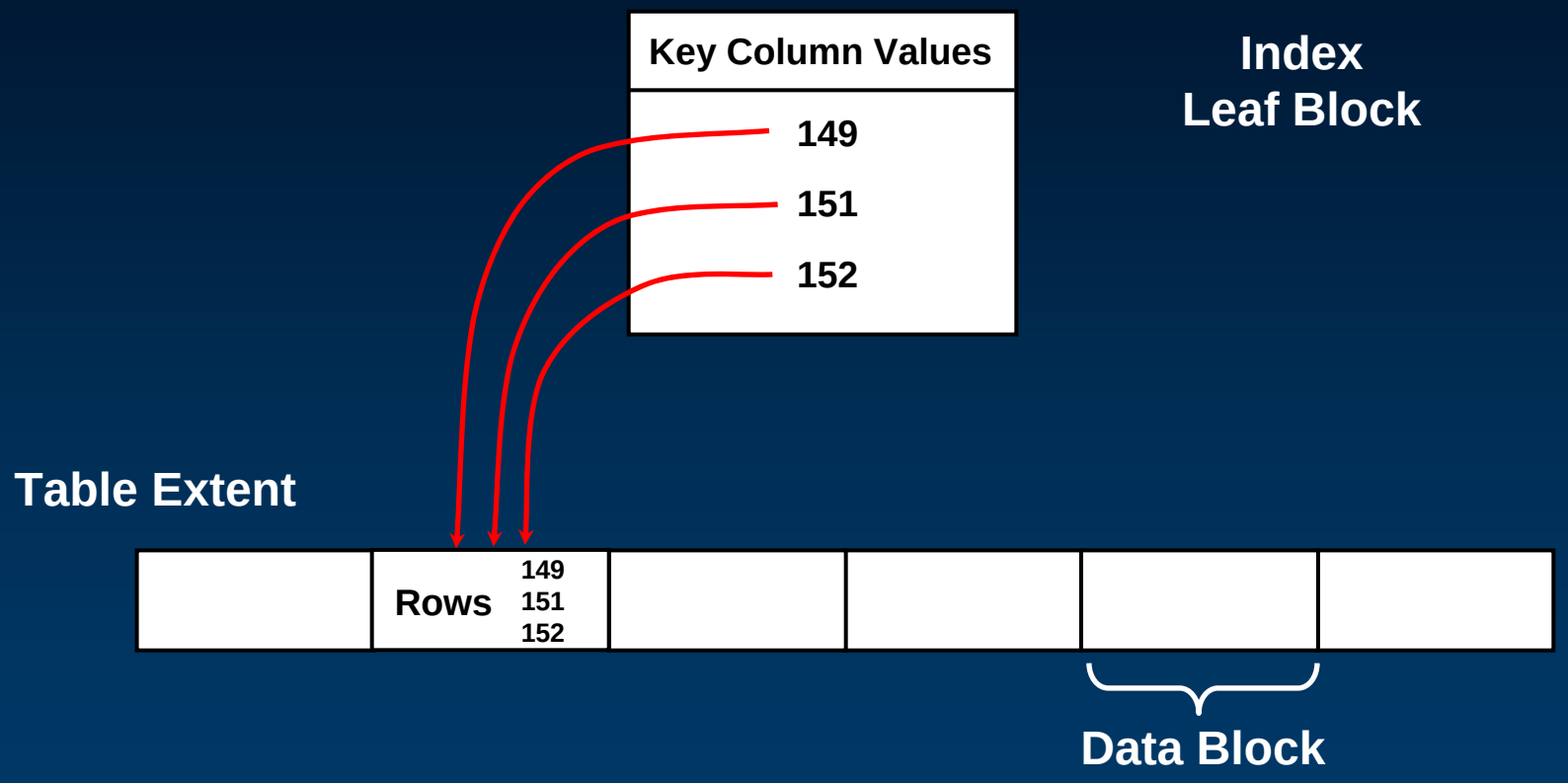
Answer: It depends!

- How many rows are in the table
  - 1 – *Full table scan* (less *i/o*)
  - 1,000,000 – might still read the entire table
- How many rows in the table where owner = 'BROWN'
  - 1 of 1,000,000 – *Index scan*
  - 900,000 of 1,000,000 – *Full table scan*
  - 50% of the table – *Index scan* or *Full table scan*
  - 25% of the table – *Index scan* or *Full table scan*

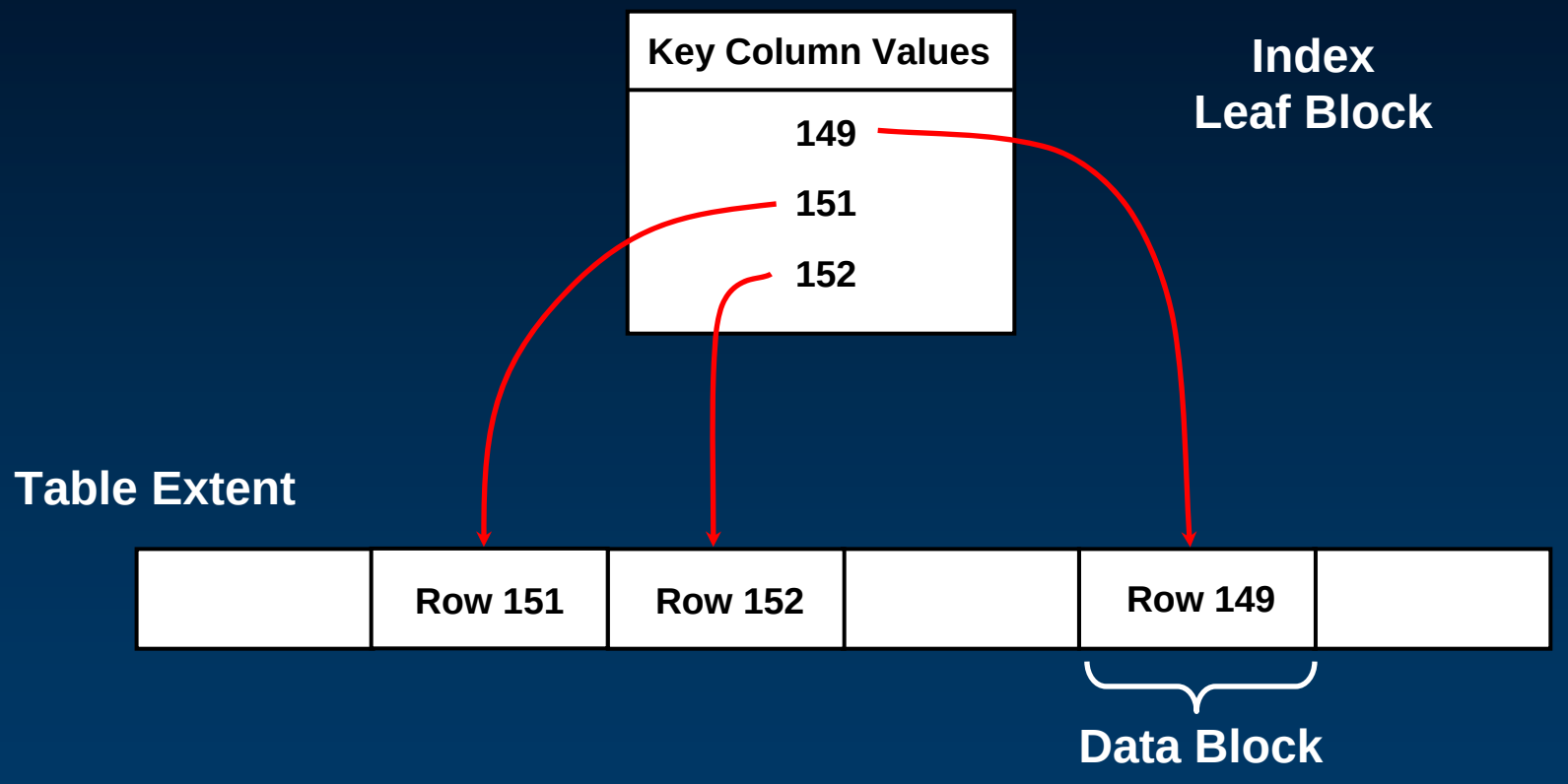
Index scan or *Full table scan* – clustering factor

- Represents how well the table is clustered based on the indexed attribute
- For the rows where owner = 'BROWN'
  - Are the rows stored in the same data block
  - Or are the rows dispersed among many data blocks
- When clustered, less *i/o* is required to retrieve the row

Example: Clustered table



Example: Non-clustered table



Clustering applies to spatial attributes

- Very important when the majority of queries are spatial
  - Every map display…
- SDE administration command for non-versioned layers

```
sdeexport –o create –t <table_name> -f <export_file>
  [-i <service>] [-s <server_name>] [-D <database>]
  –u <DB_User_name> [-p <DB_User_password>] [-N] [-q]
```

- Able to use SQL to spatially cluster data
  - See Knowledge Base article: 32423

When should an *optimizer* use a spatial index

```
SELECT * FROM parcels WHERE st_envintersects(shape,5,5,10,10) = 1
```

- Same question applies…
  - Is it better to perform a ***Full table scan*** or use the spatial index
- How does the *optimizer* know which to use

Full table scan or use the spatial index?

- Questions to consider
  - How many rows in the table?
  - How large is the search envelope (is it the entire layer)?
  - What is the *selectivity* for the search envelope?
  - What is the *cardinality*?
  - How much *i/o* will be required to use the index?
  - How much *cpu* might be consumed?
  - Is the data spatially clustered?

Gathering spatial statistics

- Execution plans are derived from statistics that are used by types, functions, and domain indexes
  - *dbms_stats.gather_table_stats*
  - Spatial filters are highly selective and spatial indexes are excellent for access paths
  - Spatial domain index statistics stored in *sde.st_geometry_index*

st_geometry_index statistic attributes

```
SQL> describe st_geometry_index

Name                                      Null?   Type
----------------------------------------- -------- -------------------------
INDEX_NAME                                         VARCHAR2(30)
UNIQUENESS                                         VARCHAR2(9)
DISTINCT_KEYS                                      NUMBER
BLEVEL                                             NUMBER
LEAF_BLOCKS                                        NUMBER
CLUSTERING_FACTOR                                  NUMBER
DENSITY                                            NUMBER
NUM_ROWS                                           NUMBER
NUM_NULLS                                          NUMBER
SAMPLE_SIZE                                        NUMBER
LAST_ANALYZED                                      DATE
```

Leveraging the extensible *optimizer*

- Utilizing ***st_geometry_index*** statistics we can assist the optimizer
  - Derive ***selectivity***
  - Calculate ***cardinality***
  - Set ***cost***

When should an *optimizer* use a spatial index

```
SELECT * FROM parcels WHERE st_envintersects(shape,5,5,10,10) = 1
```

- Either a ***Full table scan*** or use the spatial index…
- Which ever cost is less becomes the access path
  - If a ***Full table scan*** is performed st_envintersects becomes a filter for each row

How does the *optimizer* ***cost*** operators

```
SELECT * FROM parcels WHERE st_contains(shape,st_geom) = 1
```

- Perform a ***Full table scan*** and filter each row which intersects the input polygon?
- Use the polygon as an input argument, find all parcels within its envelope (using the spatial index) and then filter each row?

How does the *optimizer* ***cost*** operators?

- It is still determined by ***cost***
  - The ***cost*** of the ***Full table scan*** (baseline)
  - The ***cost*** of using the spatial index and the ***cost*** of the operator

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name      | Rows | Bytes |  Cost (%CPU) | Time      |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |           |        1 |    190 |            4   (0) | 00:00:01 |
|*  1 |   TABLE ACCESS BY INDEX ROWID            | PARCELS|        1 |    190 |            4   (0) | 00:00:01 |
|*  2 |     DOMAIN INDEX (Sel: .000285)          | PAR_IDX |        1 |        |            4   (0) | 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

  2 – access ("SDE"."ST_CONTAINS"(“SHAPE,ST_GEOMETRY”)=1)
```

How does the *optimizer* choose which predicates to apply first

```
SELECT * FROM parcels
  WHERE st_envintersects(shape,5,5,10,10) = 1
  AND owner = 'BROWN'
```

- ***Optimizer*** **calculates the following** ***costs***
  - ***Full table scan***
  - **Using the spatial index**
  - ***Index scan*** **using the owner index**

Example: predicate *cost*

- *Full table scan cost* 95
- Using the spatial index *cost* 50
- *Index scan* using the owner index *cost* 12

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name      |  Rows | Bytes | Cost (%CPU) | Time      |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |           |     1 |   190 |          12   (0) | 00:00:01 |
|*  1 |   TABLE ACCESS BY INDEX ROWID            | PARCELS|     1 |   190 |          10   (0) | 00:00:01 |
|*  2 |     INDEX SCAN                           | OWN_IDX |     1 |       |           2   (0) | 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

  1 - filter("SDE"."ST_ENVINTERSECTS"("P"."SHAPE",5,5,10,10)=1)
  2 - access("OWNER"='BROWN')
```

I know my data better then the *optimizer*…

- There will be cases when the calculated ***selectivity*** doesn't accurately represent the data

- ESRI provides the sde.spx_util for setting statistics and ***selectivity***

```
sde.spx_util.set_index_stats
sde.spx_util.set_column_stats
sde.spx_uitl.set_operator_selectivity
```

  – See Knowledge Base article: 32605

Example: Changing *selectivity* for an operator

```
SELECT * FROM buildings
  WHERE st_within(shape,st_geometry) = 1
  AND bld_type = 49
```

- **Current execution plan**

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name        | Rows | Bytes | Cost (%CPU) | Time      |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |             |    1 |   190 |           5   (0) | 00:00:01 |
|*  1 |   TABLE ACCESS BY INDEX ROWID            | BUILDINGS|    1 |   190 |           5   (0) | 00:00:01 |
|*  2 |     DOMAIN INDEX (Sel: .00016)           | BUILD_IDX |   48 |       |           5   (0) | 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

  1 – filter (“BLD_TYPE=49”)
  2 - access("SDE"."ST_WITHIN"("SHAPE,ST_GEOMETRY”)=1)
```

Example: Changing *selectivity* for an operator

- Change st_within *selectivity*
  - From .00016 to .01 (increases the *cardinality*)

```
SQL> exec spx_util.set_operator_selectivity
          (‘tb’,‘buildings','shape',‘st_within',.01);
```

- New Execution Plan

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name        | Rows | Bytes | Cost (%CPU)  | Time      |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |             |      10 |   1900 |            3   (0) | 00:00:01 |
|*  1 |   TABLE ACCESS BY INDEX ROWID            | BUILDINGS|      10 |   1900 |            3   (0) | 00:00:01 |
|*  2 |     INDEX SCAN                           | BUILD_IDX |      10 |        |            3   (0) | 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

  1 – filter ("SDE"."ST_WITHIN"("SHAPE,ST_GEOMETRY”)=1)
  2 - access (“BLD_TYPE=49”)
```

Statistic type provides

- Function *cost*
  - Assists in defining operator precedence
- *Selectivity*
- Domain index *cost*
  - iocost
  - cpucost (dbms_odci.estimate_cpu_units)
  - networkcost (always 0)
  - indexcostinfo (*selectivity*)

- <u>Without statistics optimizer uses default values</u>
  - *selectivity* 1% and spatial index *cost* 2

Writing a complex spatial query

Business requirement

- Define your objective prior to writing SQL
  - Identify all parcels within a specified area…
  - A parcel may not contain an existing building…
  - The parcel's area must be greater than 1 acre…
  - Generate a report listing the parcels by area in descending order…
- Ensures <u>you</u> understand exactly what <u>your</u> query will answer

SQL translation

- **Step 1**
  - **Identify parcels which do NOT have an existing building within a specified area…**
  - **Don't fall for the 'NOT' trap**
  - **NOT does not mean using an operator equality 0 (false)**

```
SELECT p.objectid
FROM parcel p, building b
WHERE st_intersects (p.shape, b.shape) = 1
AND st_envintersects (b.shape, 5, 5, 10, 10) = 1
```

  - **st_intersects used to identify which parcels intersect buildings**
  - **st_envintersects used for locating all buildings within the specified area**

SQL translation

- **Step 2**
  - **Using the result set from step 1 (parcels which intersect buildings) remove parcels from the outer select with NOT IN**

```
SELECT par.apn "PARCEL ID", ROUND(par.shape.area) "AREA"
FROM parcel par
WHERE par.objectid NOT IN
   (SELECT p.objectid
    FROM parcel p, building b
    WHERE st_intersects (p.shape, b.shape) = 1
    AND st_envintersects (b.shape, 5, 5, 10, 10) = 1)
AND st_envintersects (par.shape, 5, 5, 10, 10) = 1
```

  - **st_envintersects used for locating all parcels within the specified area**

SQL translation

- **Step 3**
  - **Add the filter to only return parcels with an area > 1 acre**

    AND par.shape.area > 43560

  - **st_geometry exposes the property of area and length as an attribute**

- **Step 4**
  - **Set the ORDER BY clause to report the parcels in descending order by area**

    ORDER BY par.area DESC

SQL translation

- **Final statement to execute**

```
SELECT par.apn "PARCEL ID", ROUND(par.shape.area) "AREA"
FROM parcel par
WHERE par.objectid NOT IN
   (SELECT p.objectid
    FROM parcel p, building b
    WHERE st_intersects (p.shape, b.shape) = 1
    AND st_envintersects (b.shape, 5, 5, 10, 10) = 1)
AND st_envintersects (par.shape, 5, 5, 10, 10) = 1
AND par.shape.area > 43560
ORDER BY par.area DESC;
```

Optimizer's execution plan

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |          |     6 |  2076 |    20  (10)| 00:00:01 |
|   1 |  SORT ORDER BY                     |          |     6 |  2076 |    20  (10)| 00:00:01 |
|*  2 |   HASH JOIN ANTI                   |          |     6 |  2076 |    19   (6)| 00:00:01 |
|*  3 |    TABLE ACCESS BY INDEX ROWID     | PARCEL   |     7 |  2387 |    12   (0)| 00:00:01 |
|*  4 |     DOMAIN INDEX (Sel: 21.1498601)| A24_IX1  |       |       |    12   (0)| 00:00:01 |
|   5 |    VIEW                            | VW_NSO_1 |     1 |     5 |     6   (0)| 00:00:01 |
|   6 |     NESTED LOOPS                   |          |     1 |   599 |     6   (0)| 00:00:01 |
|   7 |      TABLE ACCESS BY INDEX ROWID   | BUILDING |     1 |   258 |     3   (0)| 00:00:01 |
|*  8 |       DOMAIN INDEX (Sel: .000184)  | A20_IX1  |       |       |     3   (0)| 00:00:01 |
|   9 |      TABLE ACCESS BY INDEX ROWID   | PARCEL   |     1 |   341 |     6   (0)| 00:00:01 |
|* 10 |       DOMAIN INDEX (Sel: .00092)   | A24_IX1  |       |       |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("OBJECTID"="$nso_col_1")
   3 - filter("A"."SYS_NC00026$">43560)
   4 - access("SDE"."ST_ENVINTERSECTS"("A"."SHAPE",6225268,2295547,6230258,2300178)=1)
   8 - access("SDE"."ST_ENVINTERSECTS"("B"."SHAPE",6225268,2295547,6230258,2300178)=1)
  10 - access("SDE"."ST_INTERSECTS"("P"."SHAPE","B"."SHAPE")=1)
```

Final step: Displaying the result set

PARCEL ID	AREA
253080001	4702646
221280002	809172
221300008	611417
221232019	415702
221300006	410333
221300008	393238
221200028	382588
221132015	164814
221070003	145025

9 rows selected.

What to remember

- **Statistics are critical** for the *optimizer* in determining the optimal execution plan (access path)
- It is important for spatial data to be clustered for improving *i/o*
- For complex queries you have the ability to define operator *selectivity*

Presentation materials

- PowerPoint presentation and code are posted on the conference web site
  - http://www.esri.com/events/devsummit/index.html
- EDN – downloads and videos

Other Sessions to Attend

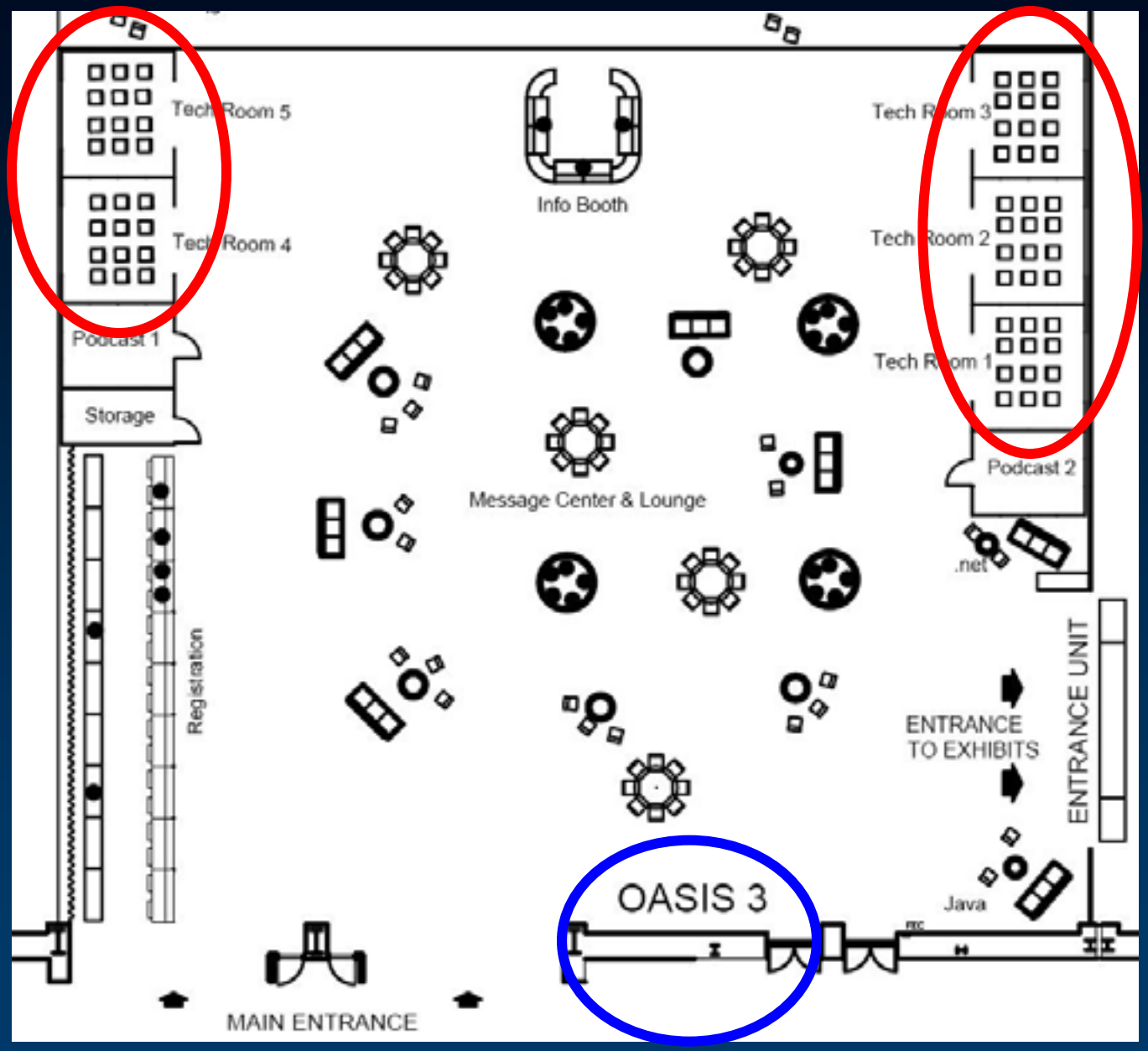
All sessions are in the Catalina/Madera rooms

- **Turbocharged Geodatabase Programming**
  - Wednesday, 21st , 1:00 - 2:15 pm
- **Distributed Geodatabase for Developers**
  - Wednesday, 21st, 4:30 - 5:45 pm
- **Raster Data Management in ArcGIS 9.2**
  - Thursday, 22nd , 8:30 - 9:45 am

Further Questions

- TECH-TALK
  - What: Opportunity to ask questions and discuss concerns with presenters and other GDB team members
  - Where: Room 5
  - When: During the next 30 minutes
- Meet the Teams
  - When: Various times
- ESRI Developers Network (EDN) website
  - http://edn.esri.com

Tech Talk
Map



Session Evaluations Reminder

Don’t forget the tech talk (Room 5)

Please turn in your session evaluations.

. . . Thank you