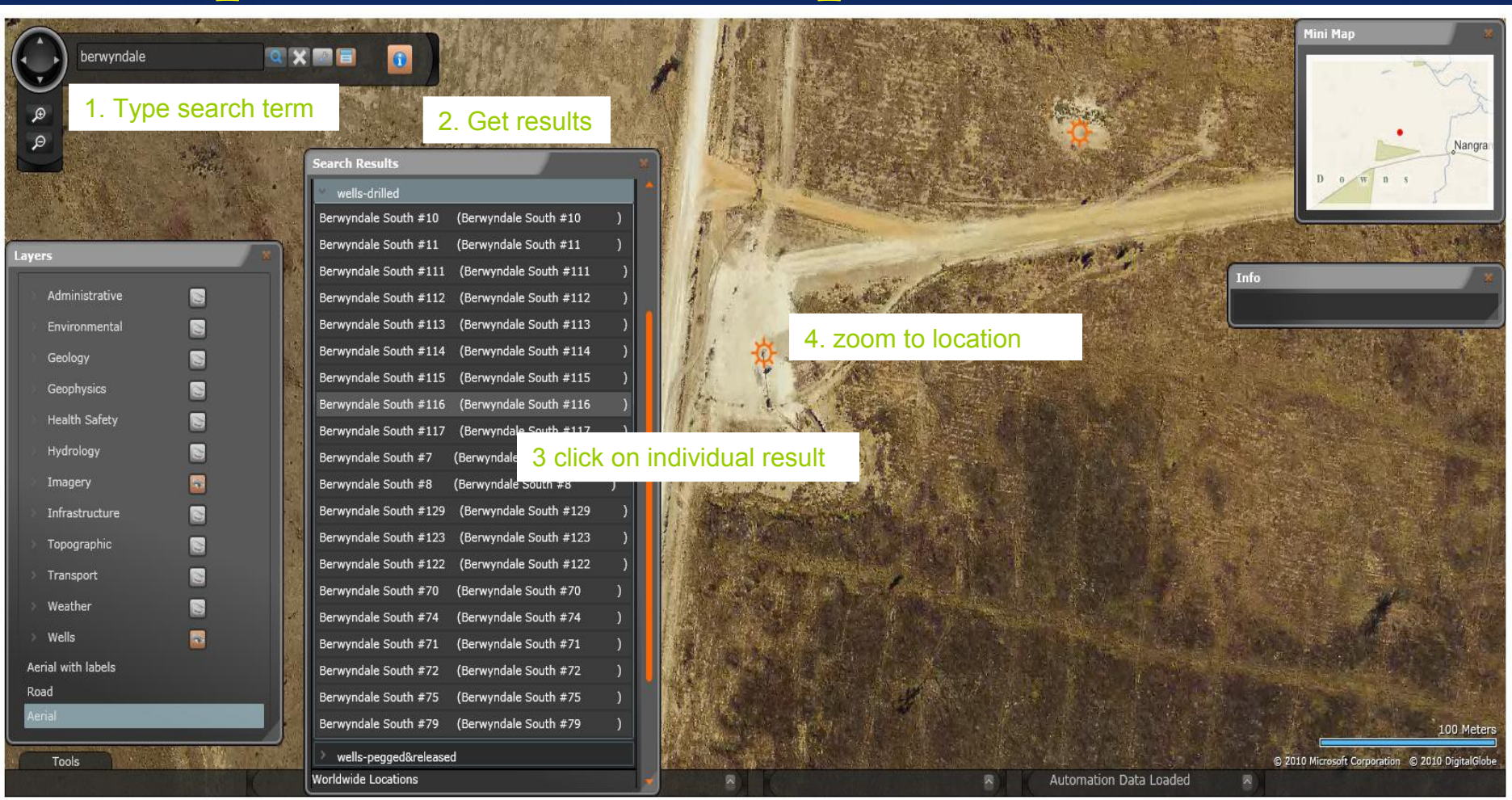
Using SQL Server 2008 Spatial and FullText to implement metadata/data search solution

Simon Greener,
The SpatialDB Advisor
simon@spatialdbadvisor.com.au

Requirements

- Build and integrate fast and flexible spatial and textual search against 700+ tables of spatial data
  - Data in SQL Server 2008
  - Search integrated within Open Source Silverlight client called DeepEarth
    - DeepEarth is a Silverlight control library for web based mapping.

Required User Experience...



Approach - Metadata

- Australian spatial data providers concentrate mostly on metadata.
- Engines allow searches of metadata
- Most metadata is out of date and limited in scope (who wrote pages other than ANZLIC Page 0?)
  - But... MBR of dataset very important metadata
- Clumsy or limited access to search actual data

Approach - Data

- Data is the most up to date incarnation of reality (as measured).
- A Spatial "Dataset"
  - Search is probably 5% spatial / 95% textual.
    - And a textual search is probably 95% data / 5% metadata.
- Yet, "GIS" textual data is probably < 5% of an organisation's attribute data holdings.
  - Metadata/data search is predominantly a non-spatial problem!

Technology (3)

- Technology
  - Internet search engines index anything given to them, and allow flexible search facilities to users.
  - Database and other IT vendors also provide sophisticated index and search technologies.
  - GIS vendors offer little in the way of product for full organisational text/spatial indexing and searching.

Solution - Both/And

- Solution chosen:
  - Uses Metadata and Data
- Is based on:
  - SQL Server 2008 Spatial
  - SQL Server 2008 Full Text Indexing.
  - And a little T-SQL Programming

SQL Metadata

- SQL Server 2008 has no OGC metadata
- To support metadata queries we created some OGC "based" metadata:
- Our base table holds more than the standard (but isn't that always the case?):

```
CREATE TABLE public._geometry_columns (
      table_catalog varchar(225) NOT NULL,
      table_schema  varchar(225) NOT NULL,
      table_name    varchar(225) NOT NULL,
      primary_key_column   varchar(128) NULL,
      geometry_column      varchar(225) NOT NULL,
      coord_dimension      int NOT NULL,
      srid                 int NOT NULL,
      type                 varchar(254) NOT NULL,
      geometry_column_type varchar(10) NULL,
      description          varchar(1000) NULL,
      text_columns         varchar(4000) NULL,
      minx                 float NULL,
      miny                 float NULL,
      maxx                 float NULL,
      maxy                 float NULL,
      minz                 float NULL,
      maxz                 float NULL,
      minm                 float NULL,
      maxm                 float NULL,
 CONSTRAINT geometry_columns_pk PRIMARY KEY
);
```

```
CREATE VIEW public.Geometry_Columns
WITH SCHEMABINDING
AS
SELECT table_catalog as f_table_catalog,
       table_schema  as f_table_schema,
       table_name    as f_table_name,
       geometry_column as f_geometry_column,
       coord_dimension,
       srid,
       type
  FROM public._geometry_columns
```

Layer View

- Combined view for easy access to MBR:

```
create view public.layer_extent
WITH SCHEMABINDING
As
SELECT ROW_NUMBER() OVER (ORDER BY table_schema, table_name,
geometry_column) as [layer_id],
      gc.description,
      gc.table_catalog          as database_name,
      gc.table_schema           as owner,
      gc.table_name             as table_name,
      gc.geometry_column        as spatial_column,
      gc.geometry_column_type,
      gc.text_columns,
      case when gc.srid IS NULL then 0 else gc.srid end as srid,
      'POLYGON((' +
      STR([minx], 12, 8) + ' ' + STR([miny], 12, 8) + ',' +
      STR([maxx], 12, 8) + ' ' + STR([miny], 12, 8) + ',' +
      STR([maxx], 12, 8) + ' ' + STR([maxy], 12, 8) + ',' +
      STR([minx], 12, 8) + ' ' + STR([maxy], 12, 8) + ',' +
      STR([minx], 12, 8) + ' ' + STR([miny], 12, 8) + '))' as geoWKT,
      [minx],[miny],[maxx],[maxy],[minz],[maxz],[minm],[maxm]
  FROM [public]._geometry_columns gc
 WHERE minx is not null and maxx is not null
   and miny is not null and maxy is not null;
```

Population

- Created T-SQL function to populate a row:
- ```
  CREATE PROCEDURE
  public.populate_geometry_column(
      @schema_name nvarchar(128),
      @object_name nvarchar(128),
      @column_name nvarchar(128)
  );
  ```
- And one for iterating over all GEOGRAPHY columned data:
- `CREATE Procedure public.populate_geometry_columns`
- ASIDE: Simple existence of populated GEOMETRY_COLUMNS view made OGC compliant software and tools, like ogr2ogr, work!


  ```

Full text indexing

- Need to create a full text CATALOG to hold our full text indexes
- ```
  SELECT DATABASEPROPERTY(DB_NAME(),
  'IsFullTextEnabled');

  EXEC sp_fulltext_database 'enable';

  CREATE FULLTEXT CATALOG qpodindex AS
  DEFAULT;
  ```


  ```

Full Text Index Table

- Wrote T-SQL procedure to create full-text index entry for all text columns in a table:
- ```
  Create Procedure
  public.build_full_text_index(
       @p_schema_name nvarchar(128),
       @p_table_name  nvarchar(128)
  )
  ```
- And a T-SQL procedure to iterate over all tables with geography columns:
- ```
  CREATE Procedure public.FullText_Index_All_Tables
  ```

T-SQL Search Function: Returns matching tables....

- Searches:
  - GEOMETRY_COLUMNS metadata view using MBR of map window
  - Executes FullText search against candidate tables
-
```
CREATE PROCEDURE [public].[QPOD_SPATIAL_SEARCH]
 ( @minx          numeric(12,8),
   @miny          numeric(12,8),
   @maxx          numeric(12,8),
   @maxy          numeric(12,8),
   @search_terms nvarchar(MAX) = NULL )
AS
BEGIN
  …
  INSERT INTO @Tables ( [table_name] )
  VALUES ( @db + '.' + @own + '.' + @tn + '.' + @spcol )
  …
  SELECT [table_name] FROM @tables;
END;
```

-
```
exec [public].QPOD_SPATIAL_SEARCH @minx = -120.6, @miny = 46.5,
@maxx = -120.4, @maxy = 46.7, @search_terms = 'ABILENE';
```

FreeText Matching

- Solution supports all forms of SQL Server 2008 FullText matching functionality:
  - SELECT … FROM <table> WHERE WHERE
  - FREETEXT(
    - *[1], 'vital safety components');
  - CONTAINS(*[1]
    - ,'rabbit **AND NOT** food');
    - ,'detonate **NEAR** quickly');
    - ,'FORMSOF(INFLECTIONAL, life)'; (eg life/lives)
    - Etc

(http://msdn.microsoft.com/en-us/library/ms187787.aspx)

[1] Means all indexed columns in the table

Issues...

- SQL Server 2008 has independent:
  - Geometry and Geography spatial data types
  - Geography has less functionality than Geometry eg can't get MBR of Geography
    - T-SQL one cannot create “overloaded” functions one taking Geography one taking Geometry

```
Create Function public.NDims (
                    @geom geometry ) …
Create Function public.NDims (
                    @geom geography ) …
```

Issues continued....

- Can only use Geometry functions by "casting" Geography to Geometry via WKT...
- ```
  Create Function public.toGeometry(
              @p_geog geography,
              @p_srid Int = null)
   returns geometry …
  ```
- ```
  Create Function public.toGeography(
              @p_geom geometry,
              @p_srid Int = null)
   returns geography …
  ```
- ```
  select min( a.geom.STEnvelope().STPointN(1).STX ) as minx,
         min( a.geom.STEnvelope().STPointN(1).STY ) as miny,
         max( a.geom.STEnvelope().STPointN(3).STX ) as maxx,
         max( a.geom.STEnvelope().STPointN(3).STY ) as maxy
    from (select toGeometry(geography,0) as geom
            from schema.object b
           where geography is not null ) a ;
  ```


  ```

Result: Flexible and Fast search...