Relational Databases and Beyond - Worboys

GIS Applications

Spring 2011

Database Management System Capabilities

The DBMS as the engine for GIS
Supports data storage and sharing, allows data to be modified and analyzed in the store
A database management system supports
- Quality control assurance and security
- Flexible access for different classes of users
- Flexible methods for retrieval
- Possibility to link pieces of information
- Possibilities for concurrent access by multiple users

Data Models

Provides the constructs for describing and structuring applications in the database

For Developers
- Represents the application domain in a way that it can be translated into a system design and implementation

For Users
- Provides a description of the structure of the system independent of the data or details of the implementation

Semantic data model (conceptual)
- Represents the meaning of the application domain as closely as possible

Implementation models (logical)
- Record-based, object-based, object-relational

Database Management Systems

Why use database management system for GIS?
What operations are supported by a database management system?
What are different types of database management systems?
What are issues in using database management systems for spatial data?
### Human Database Interaction

**Data definition**
- Description of the conceptual and logical organization

**Storage definition**
- Description of the physical structure, file location and indexing

**Database administration**
- Daily operation of the database

**Data manipulation**
- Insertion, deletion, modification and retrieval of data from the database

### Database Management

- The logical unit of interaction with the DBMS is the **transaction**
  - **Transactions**: Create, modify, delete
  - Transaction are either executed in entirety (committed) or rolled back
  - On commit, all changes since last commit are made permanent
  - Maintains ACID properties - atomicity (all or nothing), consistency, isolation (one transaction does not interfere with another) and durability (can survive a crash).

### Relational Data Models

Database is a collection of files with fixed format records

**Relational database** is a collection of tabular **relations** containing a fixed set of fields or attributes

- The data in a **relation** are structured as a set of rows
- A row or tuple consists of a list of values, one for each attribute
- The **primary key** is an attribute that uniquely identifies a record
- An attribute is associated with a domain – its range or valid values
- Relational databases require **atomic values**

**Relational schema** – describes the structure of the relations – the attributes, their domain, and any constraints
- Declared when the database is set up and not typically changed

**Relation** includes the data – changes frequently with insertions, deletions, or edits

**Database schema** – set of relation schemata

**Relational database** – set of relations
**Relation Schema**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Name</th>
<th>Type</th>
<th>Length</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute1</td>
<td>Well-ID</td>
<td>Numeric</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Attribute2</td>
<td>Depth</td>
<td>Float</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Attribute3</td>
<td>pH</td>
<td>Float</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Attribute4</td>
<td>Town</td>
<td>Character</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

**Relation**

<table>
<thead>
<tr>
<th>Well_ID</th>
<th>Depth</th>
<th>pH</th>
<th>Town</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>250.5</td>
<td>6.54</td>
<td>Frankfort</td>
</tr>
<tr>
<td>215</td>
<td>540.6</td>
<td>6.8</td>
<td>Bucksport</td>
</tr>
<tr>
<td>340</td>
<td>467.8</td>
<td>7.1</td>
<td>Dixmont</td>
</tr>
</tbody>
</table>

**Relational Database Operations**

Relational algebra operations work on one or more relations to define another relation without changing the original relations.

Both operands and results are relations, so output from one operation can become input to another operation.
Relational Database Operations

**Difference:** (−): builds the set difference of two relations. Let R and S be two tables with the same arity. R - S is the set of tuples in R but not in S.

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>R DIFFERENCE S</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 1</td>
<td>A 2</td>
<td></td>
</tr>
<tr>
<td>B 2</td>
<td>B 3</td>
<td></td>
</tr>
<tr>
<td>F 4</td>
<td>F 4</td>
<td></td>
</tr>
<tr>
<td>E 5</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>A 1</td>
<td>C 2</td>
<td></td>
</tr>
<tr>
<td>C 3</td>
<td>D 4</td>
<td></td>
</tr>
</tbody>
</table>

**Cartesian Product:** combines the tuples of one relation with all the tuples of the other relation.

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>CROSS S</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 1</td>
<td>A 1</td>
<td>F 1 A 1</td>
</tr>
<tr>
<td>A 2</td>
<td>A 2</td>
<td>F 2 A 2</td>
</tr>
<tr>
<td>B 3</td>
<td>B 3</td>
<td>F 3 A 3</td>
</tr>
<tr>
<td>C 4</td>
<td>C 4</td>
<td>F 4 A 4</td>
</tr>
<tr>
<td>D 5</td>
<td>D 5</td>
<td>F 5 A 5</td>
</tr>
<tr>
<td>E 6</td>
<td>E 6</td>
<td>F 6 A 6</td>
</tr>
</tbody>
</table>

**Select:** (σ): Works on a single relation R and defines a relation that contains only those tuples (rows) of R that satisfy a specified condition (predicate).

**Project:** (π): Works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.

**Join:** (⋈): connects two relations by their common attributes. In its simplest form it is just the cross product of two relations. A join often requires that the attributes have the same name to identify the attribute(s) to be used in the join.

<table>
<thead>
<tr>
<th>Well-ID</th>
<th>Depth</th>
<th>pH</th>
<th>Town</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>250.5</td>
<td>6.54</td>
<td>Frankfort</td>
</tr>
<tr>
<td>215</td>
<td>540.6</td>
<td>6.8</td>
<td>Bucksport</td>
</tr>
<tr>
<td>340</td>
<td>467.8</td>
<td>7.1</td>
<td>Dixmont</td>
</tr>
</tbody>
</table>

**Projection – on Well-ID and Town**

<table>
<thead>
<tr>
<th>Well_ID</th>
<th>Town</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Frankfort</td>
</tr>
<tr>
<td>215</td>
<td>Bucksport</td>
</tr>
<tr>
<td>340</td>
<td>Dixmont</td>
</tr>
</tbody>
</table>

**Select – on pH > 7**

<table>
<thead>
<tr>
<th>Well-ID</th>
<th>Depth</th>
<th>pH</th>
<th>Town</th>
</tr>
</thead>
<tbody>
<tr>
<td>340</td>
<td>467.8</td>
<td>7.1</td>
<td>Dixmont</td>
</tr>
</tbody>
</table>
Operation - Join

<table>
<thead>
<tr>
<th>Well-ID</th>
<th>Depth</th>
<th>pH</th>
<th>Town</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>250.5</td>
<td>6.54</td>
<td>Frankfort</td>
<td>3421</td>
</tr>
<tr>
<td>215</td>
<td>540.6</td>
<td>6.8</td>
<td>Bucksport</td>
<td>10439</td>
</tr>
<tr>
<td>340</td>
<td>467.8</td>
<td>7.1</td>
<td>Dixmont</td>
<td>5632</td>
</tr>
</tbody>
</table>

Join – on Town

<table>
<thead>
<tr>
<th>Well-ID</th>
<th>Depth</th>
<th>pH</th>
<th>Town</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>250.5</td>
<td>6.54</td>
<td>Frankfort</td>
<td>3421</td>
</tr>
<tr>
<td>215</td>
<td>540.6</td>
<td>6.8</td>
<td>Bucksport</td>
<td>10439</td>
</tr>
<tr>
<td>340</td>
<td>467.8</td>
<td>7.1</td>
<td>Dixmont</td>
<td>5632</td>
</tr>
</tbody>
</table>

Relational Database Interaction and SQL

SQL – Structured Query Language – specialized query language for interaction with relational databases

- Serves as a data definition language (DDL)
- DDL used to create, modify or delete a relational schema

```sql
CREATE TABLE DEPARTMENTS (
    DEPT_ID NUMBER(10) NOT NULL PRIMARY KEY,
    NAME VARCHAR2 (50) NOT NULL
);
```

Data Manipulation using SQL

Use of SQL as data manipulation language for data retrieval

```sql
SELECT *
FROM Wells
WHERE pH > 7
```

- The `SELECT` clause indicates an attribute to be retrieved from a relation – a `*` indicates all attributes
- The `FROM` clause indicates from which tables the data is to be taken, as well as how the tables JOIN to each other.
- The `WHERE` clause indicates the restrict condition

Data Manipulation using SQL

Relational joins are allowed by calling more than one relation in the `FROM` clause

Example: find wells that serve towns with populations greater than 10000

```sql
SELECT Wells.town
FROM Wells,
Towns
WHERE Wells.town = Towns.town and
population > 10000
```

Relational Algebra

```sql
SELECT town = town (Wells X Towns)
or using the equivalent join operation
Wells JOIN town = town Towns
```
Relational Databases and Geographic Information

Two approaches

**Integrated** - Put all data, spatial and non-spatial, in the relational database.

**Hybrid** - Store only the non-spatial data in the database and the spatial data in separate files (Arc/INFO)

Problems with Integrated approach

- Slow retrieval due to multiple joins
- Inappropriate indexes and access methods
- SQL not effective for spatial queries

Spatial data in relational table

<table>
<thead>
<tr>
<th>Census Blocks</th>
<th>Name</th>
<th>Area</th>
<th>Population</th>
<th>Boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1020</td>
<td>2</td>
<td>2395</td>
<td></td>
<td>Polygon((1,5), (5,5), (5,1), (1,1), (1,5))</td>
</tr>
</tbody>
</table>

What you would like to do.

SQL not designed for Spatial Queries

```sql
SELECT * 
FROM Wells 
WHERE pH > 7
```

```sql
SELECT Name 
FROM Census_blocks A, B 
WHERE adjacent(A, B) and A.name = 100
```
Object-based Data Models

Based on **objects or entities**
Something with an independent and uniquely identifiable existence in the application domain

Object-oriented Approach

- OO approach is used both as a method of **semantic data modeling** and as model for data handled by OO programming and database management
- OO model adapts OO programming constructs to database management systems

Object-oriented Constructs

**Encapsulation** – creates an identifiable collection of data and code (methods) that operate on the data

**State** of an object is determined by the data items within its wrapper

**Instance variables** – wrapped data items
- Values of the data items are themselves objects
- Objects invoke their own methods
- Objects send messages to invoke the methods of other objects
- Objects thus have state and behavior

<table>
<thead>
<tr>
<th>Object-oriented Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Well</strong></td>
</tr>
<tr>
<td>ID: Integer</td>
</tr>
<tr>
<td>Depth: Float</td>
</tr>
<tr>
<td>pH: Float</td>
</tr>
<tr>
<td>Town: String</td>
</tr>
<tr>
<td>Extent: Point</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Point</strong></td>
</tr>
<tr>
<td>ID: Integer</td>
</tr>
<tr>
<td>X Coord: Float</td>
</tr>
<tr>
<td>Y Coord: Float</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Polygon</strong></td>
</tr>
<tr>
<td><strong>Triangle</strong></td>
</tr>
<tr>
<td><strong>Octagon</strong></td>
</tr>
</tbody>
</table>

**Inheritance**

**Inheritance** – creation of a new class of object by modification of an existing class

A subclass inherits instance variables and methods of the originating or superclass

Subclasses may have additional instance variables and specialized methods

**Operator polymorphism**: operator with the same name has different implementations for different classes
Composition of Objects

**Composition** - allows modeling of complex structures

- **Aggregation** – collection of objects from different classes into an aggregate class
  - Runway, terminal, hanger, apron → airport

- **Association** – groups objects of the same class into an associated class
  - Towns → region

Object Oriented Database Management Systems

Support:
- Persistent objects, object classes, and inheritance hierarchies
- Non-procedural query languages for object class definition, manipulation and retrieval
- Efficient query handling
- Appropriate transaction processing

Two Approaches
- Extend relational to handle OO
- Build database around OO programming language

Object Oriented Database Management Systems

Creating persistent objects – Class → Persistent Object

**Methods**
- Create new persistent object
- Delete persistent object
- Retrieve the state of a persistent object
- Modify a persistent object

Comparison:
- RDM - Call by value
- OODB - Call by reference
- Match values
- Navigate by OIDs

Object - Relational

Enhance relational model with object-oriented features

- Allows complex – user defined data types
- Inheritance, aggregation and object identity
- SQL3 adds support for persistent objects
- Maintains efficient performance of relational model
- Objects in the relational model are tuples of atomic values
- Object – relational systems allow non-atomic types
- Nested relations – value of an attribute is a relation
- Support of indexes for user defined data types
**SQL3**

Provides the basis for supporting object-oriented structures

- user-defined types (ADTs, named row types, and distinct types)
- type constructors for row types
- type constructors for collection types (sets, lists, and multisets)
- user-defined functions and procedures
- support for large objects (BLOBs and CLOBs)

```sql
CREATE OR REPLACE TYPE PERSON_T AS OBJECT
    (PERSON_ID NUMBER(10),
    LNAME VARCHAR2 (50),
    FName VARCHAR2 (50),
    BIRTH_DATE DATE)

CREATE OR REPLACE TYPE EMP_T AS OBJECT
    (EMP_ID NUMBER(10),
    PERSON PERSON_T,
    HIRE_DATE DATE)

CREATE TABLE EMP OF EMP_T
    (EMP_ID NOT NULL PRIMARY KEY)
```

**OOGIS**

- Allows definition of spatial objects and inheritance hierarchy
- Allows definition of methods on these classes
  - e.g. Area, distance, intersection, union

**Oracle Spatial Data Model**

A hierarchical structure consisting of elements, geometries, and layers.

**Element**

The basic building block of a geometry. The supported spatial element types are points, line strings, and polygons.

**Geometry or geometry object**

The representation of a spatial feature, modeled as an ordered set of primitive elements. It can consist of a single element, or a homogeneous or heterogeneous collection of elements.

**Layer**

A collection of geometries having the same attribute set.
**Geometry Types**

Includes primitive types and geometries composed of collections of these types.

- **Point**
- **Line String**
- **Polygon**
- **Arc Line String**
- **Arc Polygon**
- **Compound Polygon**
- **Compound Line String**
- **Circle**
- **Rectangle**

**Oracle Spatial Data Model**

Oracle Spatial defines the object type SDO_GEOMETRY as:

```sql
CREATE TYPE sdo_geometry AS OBJECT (
  SDO_GTYPE NUMBER,  -- identifies a coordinate (spatial reference) system
  SDO_SRID NUMBER,   -- identifies a coordinate (spatial reference) system
  SDO_POINT SDO_POINT_TYPE,
  SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY,  -- identifies a coordinate (spatial reference) system
  SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);
```

- **SDO_GTYPE** indicates the type of the geometry.
- **SDO_SRID** identifies a coordinate system (spatial reference system) to be associated with the geometry.
- **SDO_POINT** defined using the SDO_POINT_TYPE object type, which has the attributes X, Y, and Z, all of type NUMBER.

**SDO_GTYPE**

<table>
<thead>
<tr>
<th>Value</th>
<th>Geometry Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>UNKNOWN_GEOMETRY</td>
<td>Spatial ignores this geometry.</td>
</tr>
<tr>
<td>01</td>
<td>POINT</td>
<td>Geometry contains one point.</td>
</tr>
<tr>
<td>02</td>
<td>LINE or CURVE</td>
<td>Geometry contains one line string that can contain straight or circular arc segments, or both. (LINE and CURVE are synonymous in this context.)</td>
</tr>
<tr>
<td>03</td>
<td>POLYGON</td>
<td>Geometry contains one polygon with or without holes.</td>
</tr>
<tr>
<td>04</td>
<td>COLLECTION</td>
<td>Geometry is a heterogeneous collection of elements.</td>
</tr>
<tr>
<td>05</td>
<td>MULTIPolygon</td>
<td>Geometry has one or more points. (MULTIPolygon is a superset of POINT.)</td>
</tr>
<tr>
<td>06</td>
<td>MULTILINE</td>
<td>Geometry has one or more line strings.</td>
</tr>
<tr>
<td>or MULTICURVE</td>
<td>(MULTILINE and MULTICURVE are synonymous in the context, and each is a superset of both LINE and CURVE.)</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>MULTIPOLYGON</td>
<td>Geometry can have multiple, disjoint polygons (more than one exterior boundary). (MULTIPOLYGON is a superset of POLYGON.)</td>
</tr>
</tbody>
</table>
Oracle Spatial Data Model

SDO_ELEM_INFO

defined using a varying length array of numbers. This attribute interprets the ordinates stored in the SDO_ORDINATES attribute

• SDO_STARTING_OFFSET -- Indicates the offset within the SDO_ORDINATES array where the first ordinate for this element is stored. Offset values start at 1 and not at 0. Thus, the first ordinate for the first element will be at SDO_GEOMETRY.SDO_ORDINATES(1). If there is a second element, its first ordinate will be at SDO_GEOMETRY.SDO_ORDINATES(n), where n reflects the position within the SDO_ORDINATE_ARRAY definition

• SDO_ETYPE - Indicates the type of the element. SDO_ETYPE values 1, 2, 1003, and 2003 are considered simple elements. They are defined by a single triplet entry in the SDO_ELEM_INFO array. For SDO_ETYPE values 1003 and 2003, the first digit indicates exterior (1) or interior (2):
  1003: exterior polygon ring (must be specified in counterclockwise order)
  2003: interior polygon ring (must be specified in clockwise order)

Example: SDO_Geometry for a rectangle

• SDO_GTYPE = 2003. 2 indicates two-dimensional, 3 indicates a polygon.
• SDO_SRID = NULL.
• SDO_POINT = NULL.
• SDO_ELEM_INFO = (1, 1003, 3). The final 3 in 1,1003,3 indicates that this is a rectangle. Because it is a rectangle, only two ordinates are specified in SDO_ORDINATES (lower-left and upper-right).
• SDO_ORDINATES = (1, 1, 7, 5). These identify the lower-left and upper-right ordinates of the rectangle.

CREATE TABLE Bank_Mket (
  bank_id NUMBER PRIMARY KEY,
  name VARCHAR2(32),
  shape MDSYS.SDO_GEOMETRY);

INSERT INTO Bank_Mket VALUES (1, 'bank_a',
  MDSYS.SDO_GEOMETRY(
    2003, -- 2-dimensional polygon
    NULL, NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior)
    MDSYS.SDO_ORDINATE_ARRAY(1,1,7,5) -- only 2 points needed to -- define rectangle (lower left and upper right)
  ));

Oracle Spatial Data Model

SDO_ORDINATES

a varying length array of NUMBER type that stores the coordinate values that make up the boundary of a spatial object. This array must always be used in conjunction with the SDO_ELEM_INFO varying length array.
CREATE TABLE Bank_Markets (
  bank_id  NUMBER PRIMARY KEY,
  name     VARCHAR2(32),
  shape    MDSYS.SDO_GEOMETRY);

INSERT INTO bank_Markets VALUES(
  10,
  'bank_b',
  MDSYS.SDO_GEOMETRY(
    2003, -- 2-dimensional polygon
    NULL,
    NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1, 19,2003,1), -- polygon with hole
    MDSYS.SDO_ORDINATE_ARRAY(2,4, 4,3, 10,3, 13,5, 13,9, 11,13, 5,13, 2,11, 2,4, 7,5, 7,10, 10,10, 10,5, 7,5) ) )