Geog 480: Principles of GIS

Fundamental database concepts - I

Anand Padmanabhan

CyberInfrastructure and Geospatial Information Laboratory
CyberGIS Center for Advanced Digital and Spatial Studies
Department of Geography and Geographic Information Science
National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign
What is a database?

- A **database** is a collection of data organized in such a way that a computer can efficiently store and retrieve data.
  - A repository of data that is logically related
- A database is created and maintained using a general-purpose piece of software called a database management system (DBMS)
Database Characteristics

- **Reliability**
  - Operate correctly during unexpected events (e.g. hardware failure)

- **Integrity**
  - Consistent and correct

- **Security**

- **Concurrency**
  - Access the same data at same time

- **Data independence**
  - Should not expose internal working

- **Distributed access**

- **Interface**
  - Ease of use

- **Performance**

- **Self-describing**
  - Users can find out what is in the database
Common database applications

• **Home/office database**
  - Simple applications

• **Commercial database**
  - Store the information for businesses (e.g. customers, employees)

• **Engineering database**
  - Used to store engineering designs (e.g. CAD)

• **Image and multimedia database**
  - Store image, audio, video data

• **Geodatabase**
  - Store a combination of spatial and non-spatial data
DBMS Elements

- User interface and query language
- Query compiler and optimizer
- Constraint enforcer
- Runtime database processor
- Stored data manager
- System catalog
  - Metadata

[Diagram showing the flow of data and components of a DBMS system]

[Graphical representation of the DBMS system components, including user interface, query language, query optimizer, constraint enforcer, runtime database processor, stored data manager, system catalog, concurrency control/backup/recovery units, and stored data]
Metadata

• Data about data
• XML (eXtensible Markup Language)
• GML
  ○ [http://www.opengeospatial.org/standards/gml](http://www.opengeospatial.org/standards/gml)
Database Transaction

- A transaction is an atomic unit of interaction between user and database
- CRUD - four basic functions of persistent storage
  - Create – Insert
  - Read(Retrieve) – Select
  - Update(Modify) – Update
  - Delete(Destroy) – Delete
Transaction Support

• **Concurrency**
  - Interleaving
  - Lost update

• **Recovery control**
  - Atomicity
  - Independence

• **DBMS operations**
  - Commit
  - Rollback
Properties for Reliable Transaction

- **ACID properties**
  - Atomicity – all or nothing
  - Consistency – transaction moves DB from one valid state to another valid state
  - Isolation – transaction execution can be serialized
  - Durability – tolerant against failures
Database Models

- Relational
- Object-oriented
- Conceptual model
  - Designers
  - Machines
  - Users
The relational model

- A *relational database* is a collection of *relations*, often just called *tables*
- Each relation has a set of *attributes*
- The data in the relation is structured as a set of *rows*, often called *tuples*
- Each tuple consists of data items for each attribute
- Each *cell* in a tuple contains a single value
Example relation

<table>
<thead>
<tr>
<th>TITLE</th>
<th>DIRECTOR</th>
<th>CNTRY</th>
<th>YEAR</th>
<th>LNGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Bug’s Life</td>
<td>Lasseter</td>
<td>USA</td>
<td>1998</td>
<td>96</td>
</tr>
<tr>
<td>Traffic</td>
<td>Soderbergh</td>
<td>USA</td>
<td>2000</td>
<td>147</td>
</tr>
<tr>
<td>Die Another Day</td>
<td>Tamahori</td>
<td>UK</td>
<td>2002</td>
<td>132</td>
</tr>
<tr>
<td>Malcolm X</td>
<td>Lee</td>
<td>USA</td>
<td>1992</td>
<td>194</td>
</tr>
<tr>
<td>American Beauty</td>
<td>Mendes</td>
<td>USA</td>
<td>1999</td>
<td>122</td>
</tr>
<tr>
<td>Eyes Wide Shut</td>
<td>Kubrick</td>
<td>USA</td>
<td>1999</td>
<td>159</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**, that is, indivisible
  - E.g. multivalued attribute values are not atomic
  - E.g. composite attribute values are not atomic
  - A relation that contains only atomic items is said to be in **first normal form** (1NF)
- The special value **null** is a member of every domain
- The null value can cause complications in the definition of many operations
A relation scheme is the set of attribute names and the domain (data type) for each attribute name.

\( A_1, A_2, \ldots, A_n \) are attributes.

\( R = (A_1, A_2, \ldots, A_n) \) is a relation schema.

E.g. \( \text{Customer-schema} = (\text{customer-name}, \text{customer-street}, \text{customer-city}) \)

\( r(R) \) is a relation instance on the relation schema \( R \).

E.g. \( \text{customer} \ (\text{Customer-schema}) \)
Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *Cinema* relation with unordered tuples

<table>
<thead>
<tr>
<th>TITLE</th>
<th>DIRECTOR</th>
<th>CNTRY</th>
<th>YEAR</th>
<th>LNGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Bug’s Life</td>
<td>Lasseter</td>
<td>USA</td>
<td>1998</td>
<td>96</td>
</tr>
<tr>
<td>Traffic</td>
<td>Soderbergh</td>
<td>USA</td>
<td>2000</td>
<td>147</td>
</tr>
<tr>
<td>Die Another Day</td>
<td>Tamahori</td>
<td>UK</td>
<td>2002</td>
<td>132</td>
</tr>
<tr>
<td>Malcolm X</td>
<td>Lee</td>
<td>USA</td>
<td>1992</td>
<td>194</td>
</tr>
<tr>
<td>American Beauty</td>
<td>Mendes</td>
<td>USA</td>
<td>1999</td>
<td>122</td>
</tr>
<tr>
<td>Eyes Wide Shut</td>
<td>Kubrick</td>
<td>USA</td>
<td>1999</td>
<td>159</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Keys

- A **candidate key** is an attribute or minimal set of attributes that will uniquely identify each tuple in a relation.
- One candidate key is usually chose as a **primary key**.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CINEMA</td>
<td>CIN_ID, NAME, MANAGER, TELNO, TOWN, GRID_REF</td>
</tr>
<tr>
<td>SCREEN</td>
<td>CINEMA_ID, SCREEN_NO, CAPACITY</td>
</tr>
<tr>
<td>FILM</td>
<td>TITLE, DIRECTOR, CNTRY, YEAR, LNGTH</td>
</tr>
<tr>
<td>SHOW</td>
<td>CINEMA_ID, SCREEN_NO, FILM_NAME, STANDARD, LUXURY</td>
</tr>
<tr>
<td>STAR</td>
<td>NAME, BIRTH_YEAR, GENDER, NTY</td>
</tr>
<tr>
<td>CAST</td>
<td>FILM_STAR, FILM_TITLE, ROLE</td>
</tr>
</tbody>
</table>
Relations

- A **relation scheme** is the set of attribute names and the **domain** (data type) for each attribute name.
- A **relation instance** is basically a **table**.
- A **database scheme** is a set of relation schemes.
- In a relation:
  - Each tuple contains as many values as there are attributes in the relation scheme.
  - Each data item is drawn from the domain for its attribute.
  - The order of tuples is not significant.
  - Tuples in a relation are all distinct from each other.
- In most relational systems, data items are **atomic**.
- The **degree** of a relation is its number of columns.
- The **cardinality** of a relation is the number of tuples.
A database consists of multiple relations.

Information about an enterprise is broken up into parts, with each relation storing one part of the information.

E.g.:  
- **account**: stores information about accounts
- **depositor**: stores information about which customer owns which account
- **customer**: stores information about customers

Storing all information as a single relation such as `bank(account-number, balance, customer-name, ..)` results in:
- repetition of information (e.g. two customers own an account)
- the need for null values (e.g. represent a customer without an account)
Operations on relations

- **Fundamental relational operators:**
  - Union, intersection and difference: usual set operations, but require both operands have the same schema
  - Selection: picking certain rows
  - Projection: picking certain columns
  - Products and joins: compositions of relations

- **Together, these operations and the way they are combined is called relational algebra combined:**
  - An algebra whose operands are relations or variables that represent relations

- **The relational model is said to be closed, because relational operators take one or more relations as input and return a relation**
Selection of tuples (Select Operation)

Relation r

Select tuples with $A = B$ and $D > 5$

$$\sigma_{A = B \text{ and } D > 5} (r)$$
Selection of Columns (Project Operation)

Relation \( r \)

- Select A and C
- Projection
- \( \Pi_{A, C}(r) \)
- Duplicate rows removed from result, since relations are sets
Joining two relations – Cartesian Product

Relations $r, s$:

$r \times s$:

If attributes of $r$ and $s$ are not disjoint, then renaming must be used.
Union of two relations

Relations \( r, s:\)

\[ r \cup s: \]

For \( r \cup s \) to be valid:

1. \( r, s \) must have the same arity (same number of attributes)
2. attribute domains of \( r \) and \( s \) must be compatible
Set difference of two relations

Relations $r$, $s$:

$r - s$:

*Set differences must be taken between compatible relations.*

1. $r$ and $s$ must have the same arity
2. attribute domains of $r$ and $s$ must be compatible
Set Intersection of two relations

Relation $r$, $s$:

$r \cap s$

Intersection must be taken between compatible relations
1. $r$ and $s$ must have the same arity
2. attribute domains of $r$ and $s$ must be compatible
Joining two relations – Natural Join

- Let \( r \) and \( s \) be relations on schemas \( R \) and \( S \) respectively.
- Then, the “natural join” of relations \( R \) and \( S \) is a relation on schema \( R \cup S \) obtained as follows:
  - Consider each pair of tuples \( t_r \) from \( r \) and \( t_s \) from \( s \).
  - If \( t_r \) and \( t_s \) have the same value on each of the attributes in \( R \cap S \), add a tuple \( t \) to the result, where
    - \( t \) has the same value as \( t_r \) on \( r \)
    - \( t \) has the same value as \( t_s \) on \( s \)

- Example:

  \( R = (A, B, C, D) \)
  \( S = (E, B, D) \)
  - Result schema = \((A, B, C, D, E)\)
  - \( r \) natural-join \( s \) is defined as:
    \[
    \Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \land r.D = s.D} (r \times s))
    \]
Natural Join Example

Relations r, s:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td>α</td>
<td>a</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>γ</td>
<td>a</td>
</tr>
<tr>
<td>γ</td>
<td>4</td>
<td>β</td>
<td>b</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>γ</td>
<td>a</td>
</tr>
<tr>
<td>δ</td>
<td>2</td>
<td>β</td>
<td>b</td>
</tr>
</tbody>
</table>

\( r \)

<table>
<thead>
<tr>
<th>B</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>α</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>β</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>γ</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>δ</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>ε</td>
</tr>
</tbody>
</table>

\( s \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td>α</td>
<td>a</td>
<td>α</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>α</td>
<td>a</td>
<td>γ</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>γ</td>
<td>a</td>
<td>α</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>γ</td>
<td>a</td>
<td>γ</td>
</tr>
<tr>
<td>δ</td>
<td>2</td>
<td>β</td>
<td>b</td>
<td>δ</td>
</tr>
</tbody>
</table>
Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>a</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>b</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>b</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>c</td>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>a</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>b</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>b</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>c</td>
<td>10</td>
<td>b</td>
</tr>
</tbody>
</table>

- $\sigma_{A=C}(r \times s)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>Symbol (Name)</td>
<td>Example of Use</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma$ (Selection)</td>
<td>$\sigma \text{ salary} \geq 85000 \ (\text{instructor})$</td>
<td>Return rows of the input relation that satisfy the predicate.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Pi$ (Projection)</td>
<td>$\Pi_{ID, \text{salary}} \ (\text{instructor})$</td>
<td>Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Join$ (Natural Join)</td>
<td>$\text{instructor} \Join \text{department}$</td>
<td>Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\times$ (Cartesian Product)</td>
<td>$\text{instructor} \times \text{department}$</td>
<td>Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\cup$ (Union)</td>
<td>$\Pi_{\text{name}} \ (\text{instructor}) \cup \Pi_{\text{name}} \ (\text{student})$</td>
<td>Output the union of tuples from the two input relations.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Structured Query Language (SQL)

- SQL provides users a standard way to manage (create, retrieve, modify and delete) relational databases.
- The SQL data-definition language (DDL) allows the specification of information about relations, including:
  - The schema for each relation.
  - The domain of values associated with each attribute.
  - Integrity constraints.
  - Other information such as
    - The set of indices to be maintained for each relation.
    - Security and authorization information for each relation.
    - The physical storage structure of each relation on disk.
Domain Types in SQL

- **char(n)**. Fixed length character string, with user-specified length \( n \).
- **varchar(n)**. Variable length character strings, with user-specified maximum length \( n \).
- **int**. Integer (a finite subset of the integers that is machine-dependent).
- **smallint**. Small integer (a machine-dependent subset of the integer domain type).
- **real, double precision**. Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n)**. Floating point number, with user-specified precision of at least \( n \) digits.
SQL

- Domain creation
- Relation scheme creation
- Data manipulation
  - Data retrieval
SQL – Query - Basic Structure

- SQL is based on set and relational operations with certain modifications and enhancements

- A typical SQL query has the form:
  ```sql
  select A_1, A_2, ..., A_n
  from r_1, r_2, ..., r_m
  where P
  ```
  - $A_i$s represent attributes
  - $r_i$s represent relations
  - $P$ is a predicate.

- This query is equivalent to the relational algebra expression:
  \[ \Pi_{A_1, A_2, ..., A_n} (\sigma_P (r_1 \times r_2 \times ... \times r_m)) \]

- The result of an SQL query is a relation.
SQL (cont.)

- Create – CREATE TABLE {TableName}(a1 int, a2 varchar(10))
- Add Rows – INSERT INTO {TableName} values (480, ’g480’)
- Update attribute values – UPDATE TABLE {TableName} set a2=’geog480’ where a1=480
- Delete rows – DELETE from {TableName} where a1=480

Aggregate Functions
- **avg**: average value
- **min**: minimum value
- **max**: maximum value
- **sum**: sum of values
- **count**: number of values
Hands on

• **Connecting to Server**
  - Use openssh client (Start → All Programs → SSH Tools -> XAgent)
  - Type: `ssh netid@geog480.cigi.illinois.edu`
  - Enter your netid passwd when prompted
  - If using other clients
    - `hostname = geog480.cigi.illinois.edu`
    - `username = netid`
    - `port = 22`
    - Enter your netid passwd when prompted

• **If successful, you just logged in a Linux system (Ubuntu)**
Unix Basics

- Folder and directories

```
/  etc  dev  home  usr  var
  |    |    |    |    |
  jono mako cory lib
  |    |    |    |
  work photos
```
Unix Basic Commands

Directory command:

```
pwd          Print the name of the working directory
cd Exercise1 Change the working directory to Exercise1
mkdir Exercise2 Make a new directory and call it Exercise2
rmdir temp   Delete the (empty) directory temp
```

Basic file command:

```
ls            List the files and directories in the working (current) directory
cat File1     Display the contents of the file
mv File1 File3 Change the name of (move) file File1 to File3
cp File1 File3 Make a copy of File1 and call it File3
rm File4      Erase (remove) the file File4
less File1    Display the contents of File1 a page at a time, q to stop displaying
```
Connecting to Database

- `psql -U username -d database_name`
  - `username = geog480`
  - `database_name = fall14`
  - Enter passwd when prompted (same as username)

- **Postgres Commands**
  - `\l` List all accessible databases
  - `\c [db_name]` connect to a DDB
  - `\dt` List all the tables in current DB
  - `\?` Help
  - `\q` Quit
Database Operations

• **Create Table**
  
  o create table REPLACE_ME_your_netid (key int, attr varchar(20), value float);

• **Insert a row**
  
  o insert into your_netid values(1, 'attr0', 100);
  o insert into your_netid values(2, 'attr1', 101);
  o insert into your_netid values(3, 'attr1', 102);
  o insert into your_netid values(4, 'attr1', 103);

• **List contents of table** (Notice that the select statement allows you to view contents in the table and the where clause allows you to filter what the records you want to view)
  
  o select * from your_netid;
  o select * from your_netid where attr='attr1';
  o select * from your_netid where key=2;
  o select key, value from your_netid limit 1;
• **Update table contents**
  - update your_netid set value=1 where key=1;
  - update your_netid set value=105 where key=3;

• **Sorting**
  - select * from your_netid Order by key asc;
  - select * from your_netid Order by key desc;

• **Counting**
  - select count(*) from your_netid;
  - select count(*) from your_netid where attr like '%1';

• **Max/Min/Avg**
  - select max(value) from your_netid; select avg(value) from your_netid where attr ilike '%1%';

• **Delete Rows**
  - delete from your_netid where key=4;
• Create a new table
  o create table your_netid_2 (key int, attr1 varchar(20), value float);

• Copying a CSV file (postgres specific)
  o \COPY your_netid_2 FROM 'your_file' with CSV HEADER
  o You may use /srv/cigi/code/test.csv for your_file

• Join your 2 tables
  o Natural Join
    • select * from your_netid natural join your_netid_2
  o Join
    • select your_netid.*, your_netid_2.attr1, your_netid_2.value your_netid inner join your_netid_2 on your_netid.key=your_netid_2.key
      o Alternative: select * from your_netid, your_netid_2 where your_netid.key=your_netid_2.key
    • Try left join, right join, full join
• Drop Table
  o DO NOT RUN THIS DURING Hands On (drop table your_netid;)

Relational databases and spatial data

• Several issues prevent unmodified databases being useful for spatial data
  o Structure of spatial data does not naturally fit with tables
  o Performance is impaired by the need to perform multiple joins with spatial data
  o Indexes are non-spatial in a conventional relational database

• An extensible RDBMS offers some solutions to these problems with
  o user defined data types
  o user-defined operations
  o user-defined indexes and access methods
  o active database functions (e.g., triggers)
• End of this topic