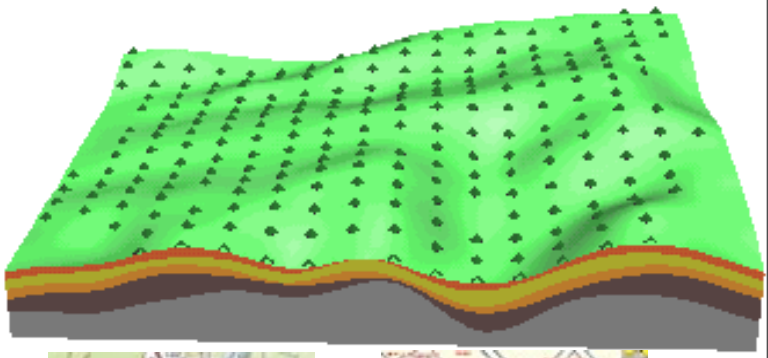
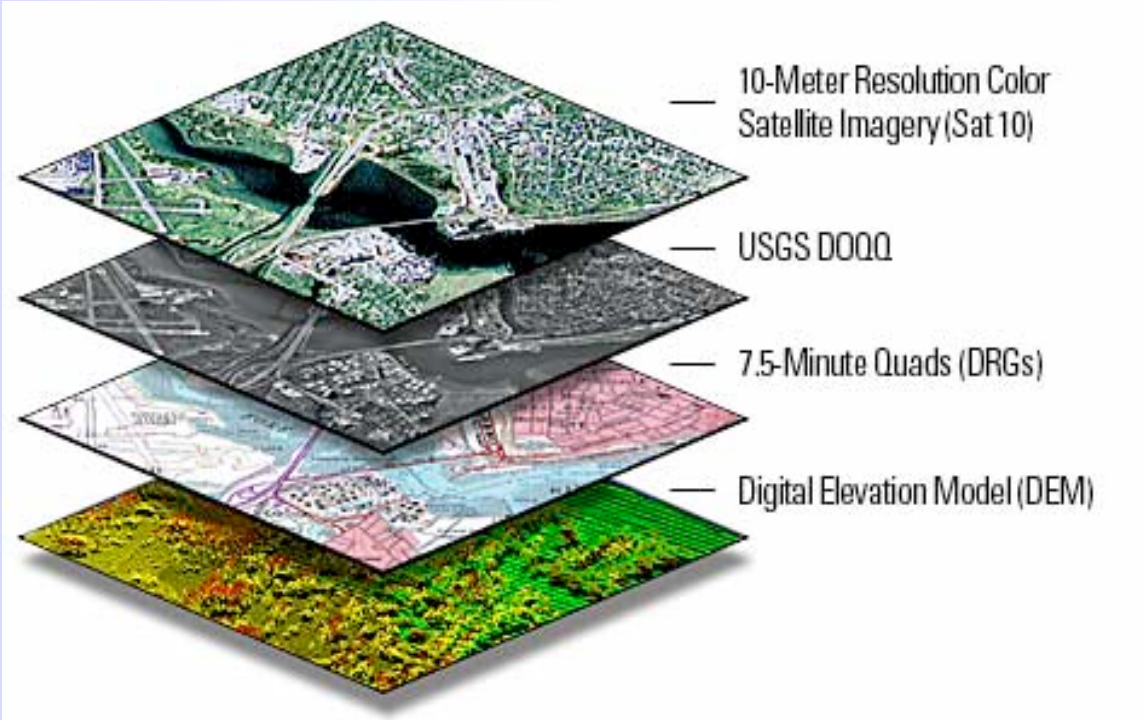
CSci 350
A Computing Perspective on GIS

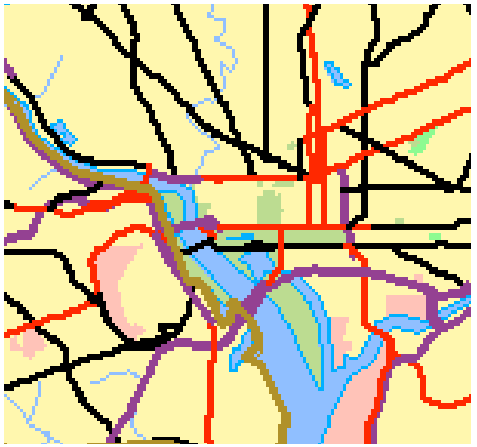
Data

- GIS deals with geospatial data

- Huge amounts of digital geospatial data available
  - from remote sensing, satellites, aerial photogr
  - from existing cartographic maps

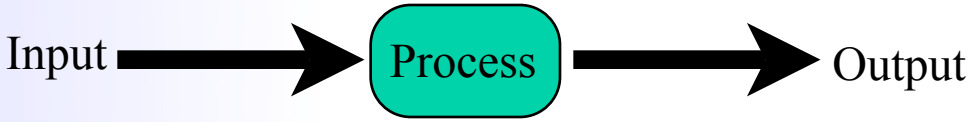






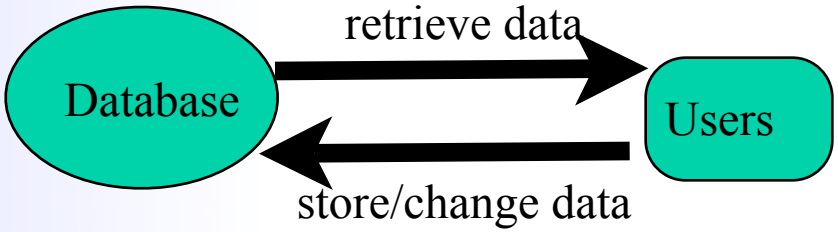
Databases

- You're used to thinking of data as files on disk and algorithms as manipulating this data and producing some output.



- Imagine a big system with many algorithms/modules/users and many types of data
  - Data is dynamic: insert new records, delete old records, change records
  - Data is heterogenous: many formats, many sources
  - Many users
  - How to store data across all modules?
  - Goals: no redundancy, integrity, efficiency, generality, security.

  - Data is stored in a database



Database

- A database is a large collection of interrelated data and the software that manages it.
- Essentially a databases consists of
  - collection of data
  - the software to manage this data: DBMS (Database Management System)

- A DBMS gives a holistic view to managing the data. It allows users to
  - define a database (specify data, types and constraints)
  - construct the database (stores the data into persistent storage)
  - update the data
  - query the data

- Main concept: data independence
  - users see a representation of the data independent of the physical storage
  - DBMS translates the user manipulation into efficient operation of the physical data

Database overview

- Levels of abstraction in a database
  - logical level
  - physical level
- Logical level:
  - Data modeling: define the database schema & constraints
    - Conceptual modeling
      - generically describe the entities and their relationships
    - Logical modeling
      - transform the conceptual model into the the data model of the DBMS
  - Data manipulation: access data using a data manipulation language provided by DBMS
- Physical level
  - Storage
  - Efficient access methods
  - Query optimization
  - Concurrency and recovery

DBMS components

- Scheduling
  - schedule processes, clients
- Query processing
  - compilation, optimization, evaluation
- Access methods
  - provides efficient structures to speed up data retrieval
- Concurrency control and error recovery
  - manage concurrent access to data; guarantees security and recovery in case of failures
- Data storage
  - efficient storage of data on disk. Possibly distributed over many disks.

E.g. : Concurrency issues

- Two transactions
  - T1: credit $1000 to my account
  - T2: debit $500

T1	T2
a = getBalance()	b = getBalance()
a += 1000	b - = 500
setBalance(a)	setBalance(b)

a = getBalance()	
a += 1000	
	b = getBalance()
	b - = 500
setBalance(a)	
	setBalance(b)

lost update

- interleaving improves efficiency but needs to be controlled carefully
  - some operations need to be atomic (not interruptible)

Databases

- Basic syllabus
  - Data models and query languages
  - Access methods and indexing
    - B-tree, R-tree, quadtrees, kd-trees
  - Join algorithms
  - Query optimization
  - Transaction management
    - concurrency control, error recovery
  - Data mining
    - clustering for very large databases, fast algorithms for mining classification rules
  - Distributed databases
  - Replication
  - XML
  - Stream processing

Most popular DB models

- Relational model (RDBMS)
  - most widely used
  - introduced 1970 by Edgar Codd [see paper, links]
  - has a query language (SQL)
- Object-Oriented model (OODBMS)
  - OO paradigm: data is structured as a hierarchy of classes interacting through methods
  - very flexible, easier to model the world
  - generally less efficient
  - lacks a query language
- Hybrid systems: ORDBMS
  - object-relational DBMS
  - provide an object-oriented shell on top of a relational DBMS core
  - combine the pros of both worlds
  - some restrictions/issues in mapping from object to relations
  - SQL does not permit true object manipulation like in OODBS

The relational model

- A relational database stores data as a collections of relations (tables)

- A relation: set of tuples associated with a relation scheme
  - ordering of tuples not significant
  - tuples are all distinct
  - columns are ordered wrt relation scheme
- A relation scheme
  - set of attributes
  - film = (name, director, country, year, length)
  - Primary key of a relation scheme
    - a group of attributes that uniquely identifies each tuple

- The software that manages the relational database: RDBMS

RDB example

Cinema database schema (from Worboys&Duckham, CRC Press)

- CINEMA(cin_id, name, manager, telno, town)
- SCREEN(cinema_id, screen_no, capacity)
- FILM(title, director, country, year, length)
- SHOW(cinema_id, screen_no, film_name)

- SCREEN relation

1	1	800
1	2	650
1	3	500
2	1	800
2	2	700
3	1	400

CINEMA relation

1	Majestic	Julie Jones	2348173874	Boston
2	Regal	Tom Franks	4659392004	Bangor
3	regal	Sydney Benson	4658293995	Waltham

Operations on relations

- basic operations supported by a relational database
  - union : R x R --> R
    - union(r1, r2) gives a new relation which is the union of r1 and r2
    - r1 and r2 must be compatible (have the same scheme)
  - difference: R x R --> R
    - r1 and r2 must be compatible (have the same scheme)
    - intersection(r1,r2) gives a new relation that contains all the tuples that appear both in r1 and r2
  - project : R --> R
    - p< attributeList > (relation)
    - it returns a new relation that has a subset of the attributes of the original
    - e.g. p < name > CINEMA =

Majestic
Regal

Operations on relations

- restrict/select
  - s <condition> (relation)
  - it select returns a new relation that has a subset of the tuples of the original
  - e.g: s <year > 2001>(FILM)
    returns a relation containing all movies with year > 2001

The hunted	Friedkin	USA	2003	94
The hours	Daldry	USA	2002	114
Die another day	Tamahori	UK	2002	132
X2	Singer	USA	2003	133

  - p<director> s <year > 2001>(FILM)

Friedkin
Daldry
Tamahori
Singer

Operations on relations

- natural join
  - X <att1 = att2> (r1, r2)
  - gives the relation formed from all combinations of tuples that agree on a specific common attribute
  - it "joins" the two relations, and produces a new table where each SHOW tuple knows its corresponding info about the movie it shows

- FILM(title, director, country, year, length)
- SHOW(cinema_id, screen_no, film_name)
- X <film_name=title> (FILM, SHOW) = (title, director, country, year, length, cinema_id, screen_no)
  - all info about the movie will be stored in all cinemas where the movie is shown

- Note: joins are expensive
  - time: for each record in first relation, need to go extract the matching tuples from the second relation
  - space: joined tables may be large (a lot of data redundancy)
- Query optimization: process queries as efficiently as possible (order matters)
  - S <cinema_id = 1> (X <film_name = title> (SHOW, FILM))
  - X <film_name = title> (SHOW, S <cinema_id = 1>(FILM))

SQL (Structured Query Language)

- SQL is the language through which the user interacts with the RDB
  - create and modify relation schemes
  - insert, modify & retrieve data from the database

- SQL provides generic functions. Users can describe what data they want, without knowing how the data is stored underneath; the DBMS performs the data manipulation as efficiently as possible

- Note: (entire) books available on SQL

SQL

- Create a domain
  - `CREATE DOMAIN domain-name data-type [default definition] [domain constraint]`
  - e.g. `CREATE DOMAIN cinema_id int`
  - e.g. `CREATE DOMAIN title string`
  - e.g. `CREATE DOMAIN gender character(1)`

- Create a relation scheme
  - a set of attributes, each with its domain, with additional properties relating the keys and integrity constraints

```
//assume we created domains film_title, director,country, year,length
CREATE TABLE FILM
    (TITLE film_title, DIRECTOR director,  COUNTRY country, YEAR year,
    LENGTH length)
    PRIMARY KEY (TITLE),
    CHECK (TITLE IS NOT_NULL));
```

SQL

```
//assume we created domains cin_id, screen_nb, film_name
CREATE TABLE SHOW
    (CINEMA_ID cin_id, SCREEN_NB screen_nb,  FILM_NAME film_name)
    PRIMARY KEY (CINEMA_ID, SCREEN_NB),
    FOREIGN_KEY(FILM_NAME) REFERENCES FILM(TITLE)
    CHECK (TITLE IS NOT_NULL),
    CHECK (CINEMA_ID IS NOT_NULL));
```

- A foreign key is a field that is a primary key in a different relation.
- Integrity: when a film is deleted in FILM, then any reference to it must also be deleted in SHOW.

SQL

- SELECT queries:

```
SELECT <items> FROM <tables> [WHERE condition] [GROUP-BY attribute-list]
[HAVING condition] [ORDER BY attribute]
```

  - FROM
    - indicates the source table(s) from which the data is to be retrieved.
    - can include optional JOIN clauses to join related tables to one another
  - WHERE
    - includes a comparison predicate, which is used to restrict the number of rows returned by the query. It eliminates all rows from the result set where the comparison predicate does not evaluate to True.
  - GROUP BY
    - used to group rows with related values into elements of a smaller set of rows.
    - GROUP BY is often used in conjunction with SQL aggregate functions or to eliminate duplicate rows from a result set.
  - HAVING
    - includes a comparison predicate used to eliminate rows after the GROUP BY clause is applied to the result set.
  - ORDER BY
    - identify which columns are used to sort the resulting data, and in which order they should be sorted (options are ascending or descending).

SQL

```
//find names of all directors who made movies after 2001
SELECT director FROM Film
    WHERE year>2001;


//find the details of film names and where they are showing
SELECT cinema_id, screen_nb, film_name, director
    FROM Show,Film
    WHERE Show.film_name = Film.title;


//select all books that are > 100$, include all columns, order by title
SELECT *
    FROM Book
    WHERE price > 100
    ORDER BY title;
```

Data modeling

You have a bunch of data.

- How do you model it?
- How do you model it with relations?

Data modeling

- Often GIS data comes already stored in a (relational) database
  - how to store GIS data in a relational db is basically standard (later)
  - GIS users “import” the data in the system

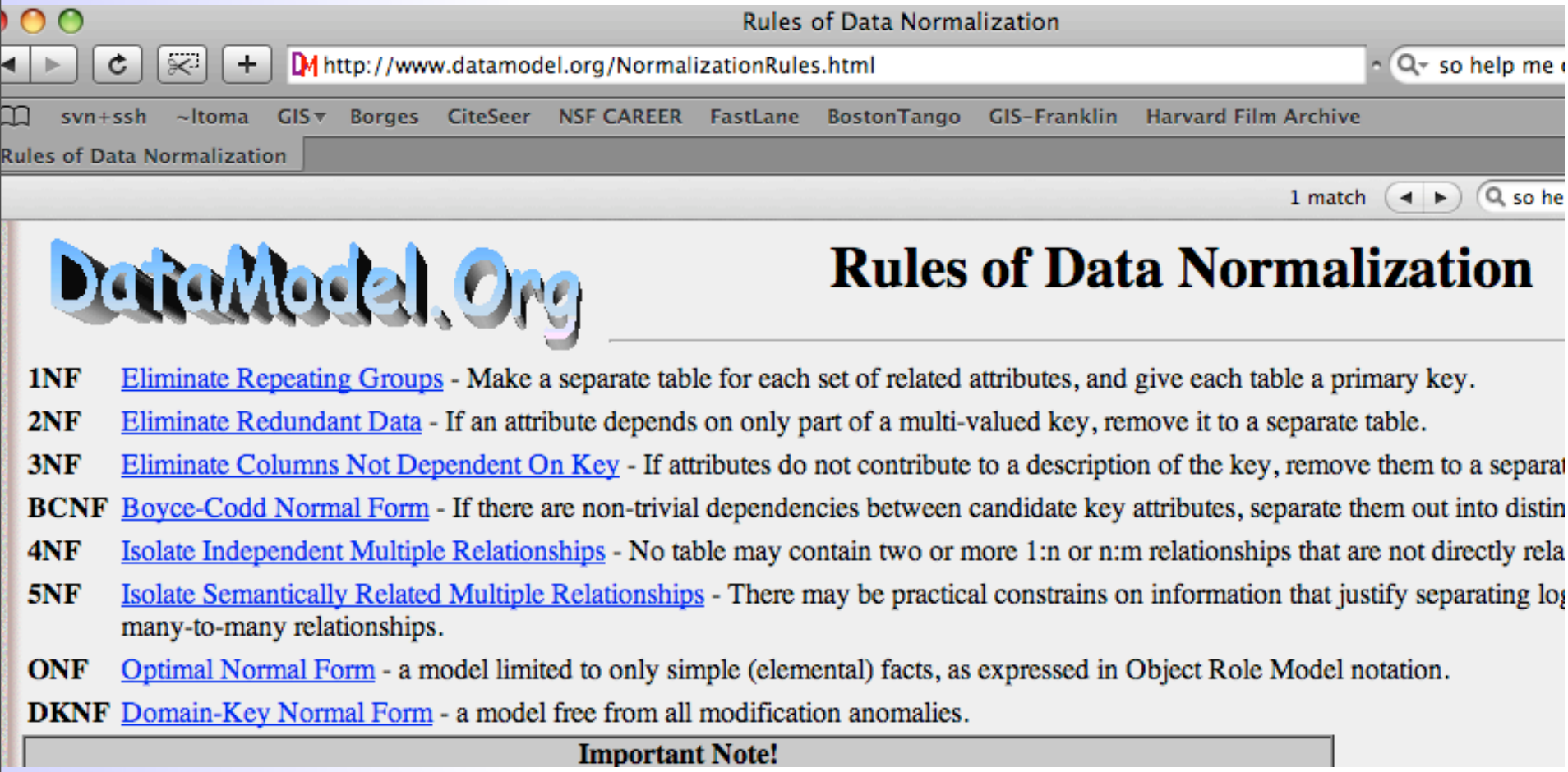
- But, if you were to model the relations yourself, how would you do it?
  - conceptual data modeling: given the structure of the data and its interactions, develop a set of relation schemes.
  - imagine thousands of relations, countless interactions
  - want to specify integrity constraints to maintain integrity of data
  - the importance of a good model is paramount for performance

  - a conceptual model has to express the structure of data in a way that is accessible to non-specialists
  - has to be able to capture the complexity of data
  - has to be easily translatable to the data model of the DBMS

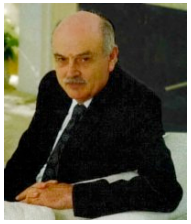
Data modeling

- Conceptual data modeling is done with the entity-relationship (ER) model
  - define entities
  - entities have attributes
    - e.g. town has centroid, name, population
    - e.g. road has road_id, class, start_point, end_point
  - connect entities with relationships
    - towns may lie on roads
  - relationships:
    - many-to-many, many-to-one, one-to-one
    - may have other attributes
  - develop a flowchart
- An ER model can be transformed into a database scheme.
  - Tradeoff space and time
  - store all data into a single relation with all the info
    - no need of joins but (huge) data redundancy
  - store many small relations; will need a lot of joins
    - no redundant data, but many joins
  - ==> data normalization

Codd data normalization



Trivia From Wikipedia, the free encyclopedia



- **Codd's 12 rules** are a set of thirteen rules (numbered zero to twelve) proposed by Edgar F. Codd,
- Codd produced these rules as part of a personal campaign to prevent his vision of the relational database being diluted, as database vendors scrambled in the early 1980s to repackage existing products with a relational veneer. Rule 12 was particularly designed to counter such a positioning. In fact, the rules are so strict that all popular so-called "relational" DBMSs fail on many of the criteria.

The rules

- **Rule 0:** The system must qualify as *relational*, as a *database*, and as a *management system*.
- For a system to qualify as a relational database management system (RDBMS), that system must use its *relational* facilities (exclusively) to *manage* the *database*.
- **Rule 1:** The *information rule*:
- All information in the database is to be represented in one and only one way, namely by values in column positions within rows of tables.
- **Rule 2:** The *guaranteed access rule*:
- All data must be accessible with no ambiguity. This rule is essentially a restatement of the fundamental requirement for primary keys. It says that every individual scalar value in the database must be logically addressable by specifying the name of the containing table, the name of the containing column and the primary key value of the containing row.
- **Rule 3:** *Systematic treatment of null values*:
- ...

Trivia

- Codd's most famous quote was when he was asked why he chose the word "normalization" to describe relational database modeling.
- Codd was reported to have replied

  "At the time, Nixon was normalizing relations with China. I figured that if he could normalize relations, then so could I".

Trivia

- Another Codd famous mnemonic on his definition for second-normal-form:

  "*A relation is in second normal form if the relation depends on the key, the whole key, and nothing but the key, so help me Codd*".