CMSC 424: Database Design

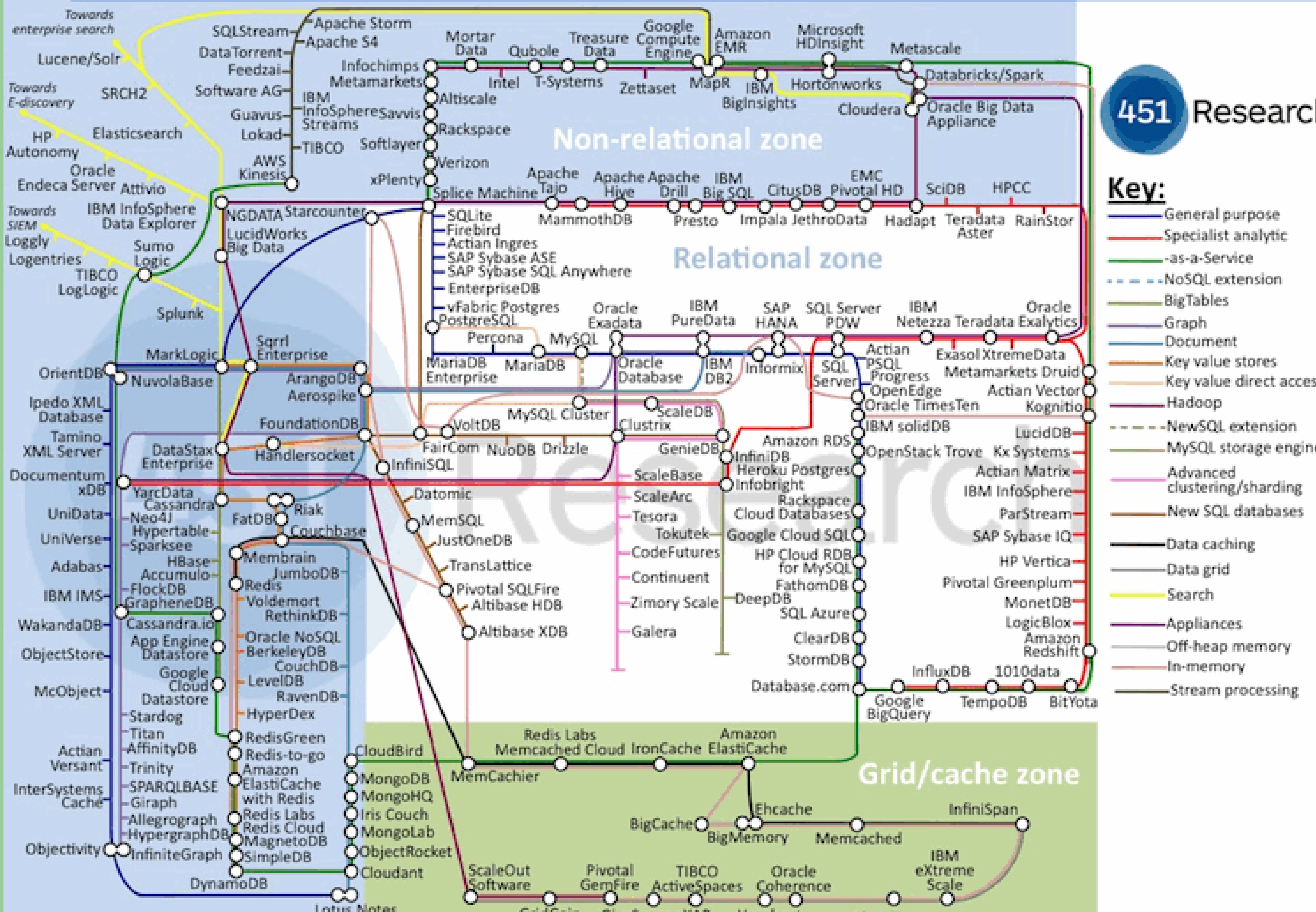
Spring 2018

Introductory Material

Database Systems

- Database: a structured, queryable collection of records
- Database management system (DBMS): software to manage and query the database
  - 20 billion dollar industry
  - Deployed everywhere
  - At the low end, you can get a DBMS for free; at the high end, they can go for 7 figures

Data Platforms Landscape Map – February 2014



Overview

- This course covers
  - How to use a DBMS
    - If you build an app, you will likely use a DB
  - How to build a DBMS
    - Database systems are highly concurrent, complex systems
      - Learning how they work:
        - Will make you a better programmer
        - Will open up more career opportunities

Prerequisites

- Minimum grade of C- in CMSC351 and CMSC330

Readings

- Textbook: “Database System Concepts”
  - Silberschatz, Korth & Sudarshan; 6th Edition, McGraw Hill.
- Readings listed on syllabus refer to 6th edition.
  - Earlier edition OK, but you have to figure out the mapping.
- Some other material linked to from the syllabus

Exams

- 2 Midterms
  - March 5 and April 9 in class
- Final exam on May 15 from 1:30-3:30PM

Projects / Programming Assignments

- Done individually
- You may ask, and answer, **general** questions on Piazza
- Assignments will use a mixture of languages, with Python and Java being the most common.

Clickers / In-class particaption

- We will be using clickers this semester
- Your answers will count toward final grade
- Any question that majority of class gets wrong will be deemed unfair and thrown out
- You can avoid answering questions for any three classes of your choosing
  - As soon as you answer the first question from a class, you can no longer count it as a missed class

Reading homeworks

- Online quizzes before class
- No late submissions accepted
  - But you can skip 2 for any reason

Course grade

- Projects/programming assignments: 30%
- Reading homeworks: 5%
- Mid-term exams: 32%
- Final Exam: 25%
- Class participation (clicker answers): 8%

Teaching Fellows

- Amit Chavan - Wed 9AM-noon
- Hui Miao - Thu 10AM-1PM
- Souvik Bhattacherjee - Tue 9AM-noon
- Virinchi Srinivas - Mon 9AM-noon
- Adam Ackerman - Mon noon - 1PM; Friday 11:30AM-1:30PM
- Ayokiitan Akalaa - Mon/Wed 2-3:30PM
- Kevin Wittmer - Mon/Wed 5:30-7:00PM
- Brian Mitchell - Thu 10:30AM - noon, Fri 10AM-11:30AM
- Anthony Pingelli - Mon 1PM-2PM, Wed noon-2PM

- Bottom line:
  - M/W Office hours 9AM-class time, 5:30-7PM
  - Tu/Th Office hours morning/early afternoon
  - Fri office hours 10AM-1:30PM

Assignment 0

- Assignment 0 (setting up your programming environment) will be released shortly
  - Vagrant+VirtualBox, PostgreSQL, IPython, etc.
- TA office hours start tomorrow in case you need help

True story

- Needed to create an online RSVP application
- Specs:
  - For a series of meetings
  - Meeting would only be held if >=X people committed to attending
  - Before each meeting, send out an e-mail, asking people to RSVP if they can come
  - Count responses, then decide if meeting is on

First Cut

- Have everyone register an account with some basic information
  - Form-based Web-page writing to a file (accounts.txt) in the backend (Perl)
- E-mail points to a different Web-form for people to sign-in and say ‘yes’ or ‘no’
  - Again, results written to file (meeting01-12.txt)
- Cron-script that periodically counts how many responses and either
  - Sends out another reminder e-mail OR
  - Makes final decision
  - Status file (decided.txt) edited to stop mind-changing

End of story?

- People are cranky – hate e-mails
  - “I never can make Monday meetings”
- Not everyone should be counted
- “yes”, “no” too restrictive
- Problem: implementation nightmare
  - Everything requires custom code
  - Bugs crept up through lack of data independence

It gets worse

- Concurrency became a problem
  - Need to lock entire file to edit just one line
  - In some cases, concurrency resulted in mis-counts

And worse …

- Performance became an issue with scale
  - find the e-mails of all people coming (from ID)
  - view status of all upcoming meetings

Lessons Learned

- Never tell anyone you know how to create a Web application (or mobile or whatever)
- There’s no such thing as a simple app
- Should have used a DBMS

Should have used a DBMS

- Generic query interface
- Handles data independence
- Handles concurrency
- Handles recovery from crash
- Designed for performance (indexes, address offsetting, result caching)

Relational Data Model

- Data stored in tables, tables have attributes

Relational Model

People

ID	Name	E-mail	Phone	Gender
1	Joe	joe@yale.edu	555-555-5555	M
2	Jill	jill@yale.edu	555-555-5555	F
3	Jasmine	jasmine@yale.edu	555-555-5555	F
4	Jeff	jeff@yale.edu	555-555-5555	M

Meetings

ID	Date	Time	Location
1	15-Jan	7:30AM	AKW 500
2	15-Jan	4:30PM	AKW 500
3	17-Jan	7:30AM	AKW 500

People-Meetings

Person ID	Meeting ID	Status
1	1	yes
2	1	yes
3	1	no
4	1	yes
1	2	yes
2	2	no
3	2	no
4	2	no

Other Models Possible

- (Joe, joe@yale.edu, 555-555-5555, M, (((meeting 1, Jan-15, 7:30AM, AKW 500), yes), ((meeting 2, Jan-15, 4:30PM, AKW 500), no))), (Jill, …)

Database queries

- SELECT e-mail
  FROM People
  WHERE Gender='F'

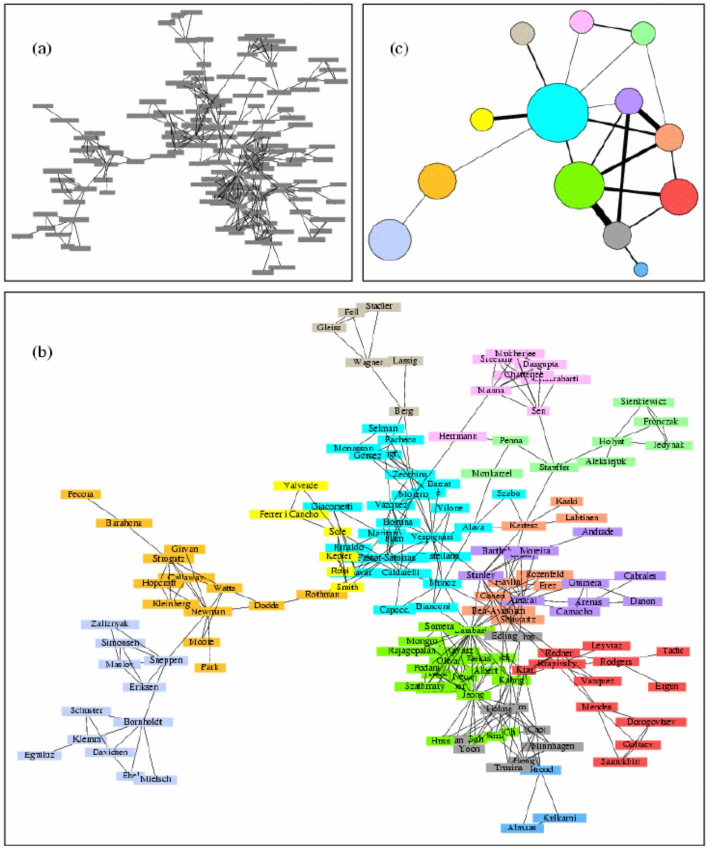
- SELECT People.name
  FROM People, People-Meetings
  WHERE People.ID=People-Meetings.PersonID
  AND People-Meetings.Status = 'yes'
  AND People-Meetings.MeetingID = 1

Other Constructs

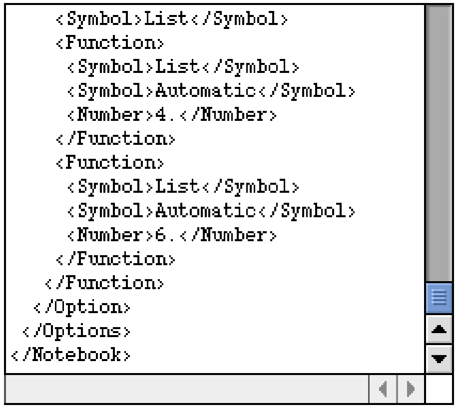
- INSERT, DELETE, UPDATE

Structured vs Unstructured Data

- Some data has a little more complicated structure
  - E.g graph structures
    - Map data, social networks data, the web link structure etc
  - Can convert to tabular forms for storage, but may not be optimal
  - Queries often reason about graph structure
    - *Find my "Erdos number"*
    - *Suggest friends based on current friends*
  - Growing importance in recent years in a variety of domains: Biological, social networks, web…

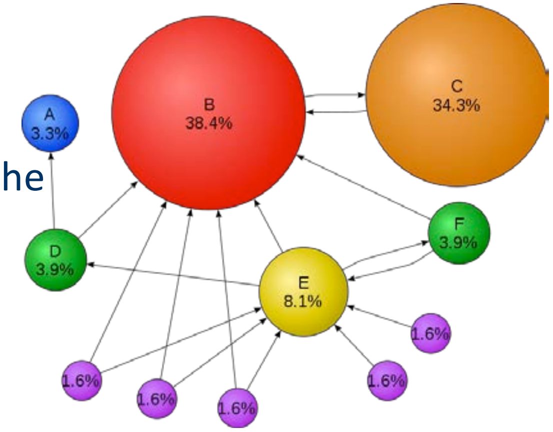


Structured vs Unstructured



```
      <Symbol>List</Symbol>
      <Function>
        <Symbol>List</Symbol>
        <Symbol>Automatic</Symbol>
        <Number>4.</Number>
      </Function>
      <Function>
        <Symbol>List</Symbol>
        <Symbol>Automatic</Symbol>
        <Number>6.</Number>
      </Function>
    </Function>
  </Option>
 </Options>
</Notebook>
```


- Increasing amount of data in a semi-structured format
  - XML/JSON – Self-describing tags (HTML ?)
  - Complicates a lot of things
- A huge amount of data is unstructured
  - Books, WWW
  - Amenable to pretty much only text search… so far
    - Information Retrieval research deals with this topic
  - What about Google search ?
    - Google search is mainly successful because it uses the structure (in its original incarnation)
- Video ? Music ?
  - Can represent in DBMS’s, but can’t really operate on them



circle size == page importance == **pagerank**
more incoming links → higher pagerank
incoming links from important pages → higher pagerank

DBMSs to the Rescue

- Massively successful for *highly structured data*
  - Why ? Structure in the data (if any) can be exploited for ease of use and efficiency
  - How ?
  - Two Key Concepts:
    - Data Modeling: Allows reasoning about the data at a high level
      - e.g. “emails” have “sender”, “receiver”, “…”
      - Once we can describe the data, we can start “querying” it
    - Data Abstraction/Independence:
      - Layer the system so that the users/applications are insulated from the low-level details

What we will cover…

- Representing information
  - data modeling
  - semantic constraints
- Languages and systems for querying data
  - complex queries & query semantics
  - over massive data sets
- Concurrency control for data manipulation
  - ensuring transactional semantics
- Reliable data storage
  - maintain data semantics even if you pull the plug
  - fault tolerance

What we will cover…

- Representing information
  - data modeling: *relational models, E/R models, XML/JSON*
  - semantic constraints: *integrity constraints, triggers*
- Languages and systems for querying data
  - complex queries & query semantics*: SQL*
  - over massive data sets*: indexes, query processing, optimization, parallelization/cluster processing, streaming*
- Concurrency control for data manipulation
  - ensuring transactional semantics: *ACID properties, distributed consistency*
- Reliable data storage
  - maintain data semantics even if you pull the plug: *durability*
  - fault tolerance: *RAID*

For next class

- Read Chapter 2 of textbook
  - (Also a good idea to skim chapter 1)