CS 2451:
Database Systems

Spring 2020
Instructors: Dr Roxana Leontie and Bhagi Narahari
TAs: Ayush Singh and Huzefa Raja
UTA: Billy Miller
LAs: Kevin Deems, Jonathan Minkin

http://www.seas.gwu.edu/~bhagiweb/cs2541

What Is a Database System

- A Database is a large collection of related data.
  - Not arbitrary, unrelated data
    - Definition changes with 'big data' databases
  - Models real-world *enterprise.*
    - *Entities: University = Students, Courses,Professors*
    - *Relationships: Students are taking courses*
  - *Data organized using a data model*
- A *Database Management System (DBMS)* is the software system to store/retrieve/manage the database.
  - Provides an interface over the database
  - Examples: Oracle, MySQL, MongoDB, Hadoop, Dynamo,….
- Database System: DBMS + Data (+ Applications)

Why Study Databases?

- Shift from computation to information/data
  - **Huge amount of data today**
- To effectively analyze data:
  - collect relevant data
  - store in manner amenable to efficient access
  - provide programming interface
- Most CS courses concentrate on code/system– this focuses on managing, manipulating and representing data
- As a S/W developer you may be required to
  - Query database, program with databases, design databases….
  - Full stack development (LAMP stack)
- And then there is Accreditation requirement ☺

Databases in the Real-World

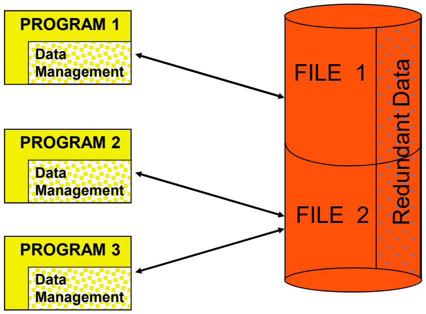
- Databases are everywhere in the real-world even though you do not often interact with the DBMS directly.
  - ~$50 billion annual industry
- Examples:
  - Retailers manage their products and sales using a database.
    - Wal-Mart has one of the largest databases in the world ~40 Petabytes !
  - Online web sites such as Amazon, eBay, etc..
  - Social media sites – Facebook (PHP+MySQL!!), Instagram,…
    - Facebook: >500 Terabytes data per day!
  - The university maintains all your registration information in a database.
- Lots of other examples..
- What data do ***you*** have?...concept of "ownership"

Why use a DBMS? Why not use file processing ?

- Why do we need a DBMS, instead of coding your app in C ?
  - i.e., why not just use File processing systems?
- A ***file-based system*** (file processing) is a set of applications that use files to store their data.

File Processing

Ques: When did most of you implement such a (conceptual) system?



File Processing- example

- “database” storing student course enrollment information
  - For each student we store a record containing name, course, grade

[Name, Course, Grade] ← *File system does not even know this*

John Smith, CS2461, C

Ketan Patel, CS1311, B ← *Records (Data) stored using some data structure (ex: linked list)*

Billy Miller, CS2541, A

…..

- Query1: Find all courses taken by Billy Miller
- Query 2: Find all of Billy’s Grades

Each of above queries have code associated with them…

1. Now consider query 3: Print Billy’s transcript (GPA, etc.)
2. Next consider, changing the data structure…Can we use the same code?

File Processing systems?

- Each application in a file-based system contains its own code for accessing and manipulating files. This causes several problems:
  - Code duplication of file access routines
  - Change in data (structure) requires change in code
  - High maintenance costs
  - Hard to support multi-user access to information
  - Difficult to connect information present in different files
  - Difficulty in developing new applications/handling data changes

DBMS - Data Independence and Abstraction

- The major problem with developing applications based on files is that the application is dependent on the file structure.
- there is no *program-data independence* separating the application from the data it is manipulating.
  - If the data file changes, the code that accesses the file may require changes to the application.
- A major advantages of DBMS is they provide data abstraction.
- *Data abstraction* allows the internal definition of an object to change without affecting programs that use the object through an external definition.

Data Independence

- Logical data independence

  Protects the user from changes in the logical structure of the data:

  could reorganize the **student** "schema" without changing how we query/store it
- Physical data independence

  Protects the user from changes in the physical structure of data:

  could change how **student** data/table is stored in memory without changing how the user would write the query

Data Independence: Example

What this means….

- A user of a relational database system should be able to use the database without knowing about how the precisely how data is stored, e.g.

```
SELECT Name, Courses
FROM Students
WHERE Name= 'Billy Miller'
```

Above "query" does not need to know how the data in Students in stored

After all, you don't worry about IEEE floating-point when you do division in a Java program or with a calculator

So what can we conclude thus far….

- File processing is not an effective/efficient solution
- Need a "database approach" that provides data independence
- So how do we specify business rules of the data, relationships within the data, who gets access to what data,…..**How to organize and manage the data ?**

Data Models: How to organize the data ?

- What is the data needed ?
  - Eg: What do we need to store to uniquely identify a student entity ?
- How to store & organize the data ?
  - How many attributes are really needed about a student/course/faculty
  - What is an efficient way to organize the data ?
    - This is why we need to study schema design and Normal forms
- How to query the data and generate reports for the end users ?
  - Need a database query language, such as SQL

Data Models and data representation

- All of the data have an implicit *data model*
  - Basic assumption on what is an item of data, how to interpret it, etc.
- A *data model* is a formal framework for describing data.
  - Data objects, relationships, constraints (business rules)
  - Provides primitives for data manipulation and data definition
  - Starting point to design of DBMS
  - Provides us with the mathematical basis to prove/assert properties and show correctness of algorithms
- The relational model was the first model of data that is *independent* of its data structures and implementation
  - Data organized as relations (tables)
  - A theory of normalization guides you in designing relations
  - Other data models: network, hierarchical, Object Oriented…
- With explosion in unstructured data and big data, new models emerged…NoSQL database models
  - Relational model is observed to be 'inefficient' for many such applications

How to define and use the database: Data Definition and Manipulation Languages

- data definition language (DDL) to specify database schema
  - What data, and how it is organized (logical level)
- Data manipulation language (DML) allows users to access or manipulate data as organized by data model
  - procedural DMLs: require user to specify what data and how to get it
  - non-procedural DMLs: require user to specify what data is needed without specifying how to get it.
  - Commercial languages – SQL

Relational DB Query Languages

- Formal query languages:
  - Relational algebra,
  - Relational Calculus,
  - Why study formal languages ?
- Commercial query languages: SQL
- SQL: "descendent" of SEQUEL; mostly relational algebra and some aspects of relational calculus
  - has procedural and non-procedural aspects
  - Has DDL and DML components

Database Schema

- Similar to types and variables in programming languages
- Schema – structure of the database
  - Ex: database contains information about Students and Courses and the relationships between them
  - Expressed in some data model – using a DDL
- Occurs at multiple levels:
  - Logical Level: Database design at the logical level
  - Physical Level: Database design at the physical level

Levels of Data Modelling

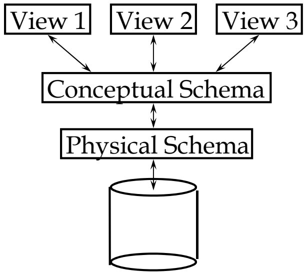
- **Logical Level**: describes data stored in the database and the relationship between them

```
ex:  type student {        name: string
                           street: string
                           GWID: integer }
```

- **Physical Level**: describes how a record is stored (i.e., how is data organized on the disk)
  - Ex: sorting, page alignment, index
- Big Idea: Logical and Physical level independence
  - Can change one without chaning the other !!
- Additional **View level**: application programs hide details of data types and can also hide some information (salary?) for security & privacy purposes

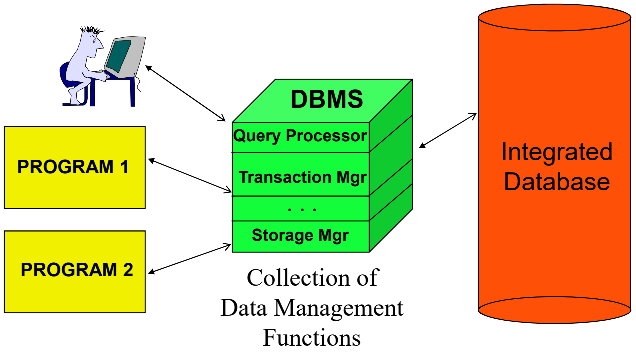
Summary- Levels of Abstraction

- Many *views*, single *conceptual (logical) schema* and *physical schema*.
  - Views describe how users see the data.
  - Conceptual/Logical schema defines logical structure
  - Physical schema describes the files and indexes used.



Schemas are defined using Data Definition Language (DDL); data is modified/queried using Data Manipulation Lang(DML).

Building DB applications: The Database System Approach - Abstract view



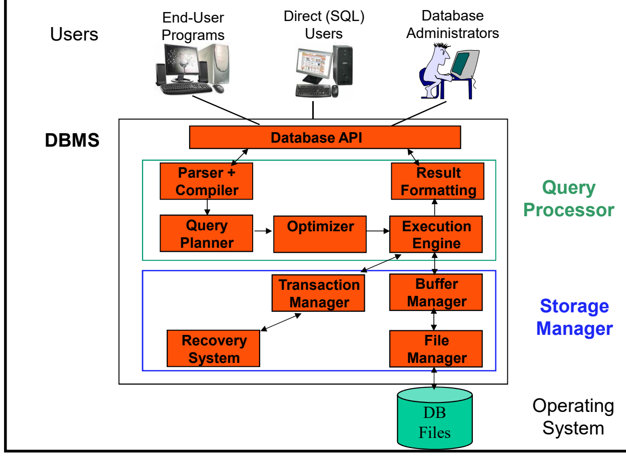
DBMS

- The data abstraction is provided by the DBMS
  - Separation b/w Logical and Physical, Query language parsing etc.
- A database management system provides *efficient*, *convenient*, and *safe multi-user* storage and access to *massive* amounts of *persistent* data.
  - ***Efficient & Convenient*** - Able to handle large data sets, complex queries without searching all files and data items, easy to write queries.
  - ***Scalability*** – Large/huge data.
  - ***Persistence & Safety*** - Data exists after program execution completes, handles loss of power.
  - ***Multi-user*** - More than one user can access and update data at the same time while preserving consistency….concept of transactions

Components of a DBMS

- A database management system provides *efficient*, *convenient*, and *safe multi-user* storage and access to *massive* amounts of *persistent* data.
- A DBMS is a complicated software system containing many components:
  - ***Query processor*** - translates user/application queries into low-level data manipulation actions.
    - Sub-components: query parser, query optimizer
  - ***Storage manager*** - maintains storage information including memory allocation, buffer management, and file storage.
    - Sub-components: buffer manager, file manager
  - ***Transaction manager*** - performs scheduling of operations and implements concurrency control algorithms.
    - You will learn more about storage management and concurrency in the Operating Systems course

DBMS Architecture: Complete Picture



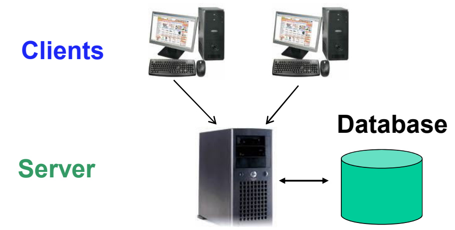
This course is about Database Design…

- Focus is on design of databases
  - Working at the logical level
- Internals of DBMS is not the focus in this course
  - BUT we will touch upon a few key concepts that make DBMS' work
  - DBMS design brings together several key concepts from Computer Science
    - Languages, Compilers/translation, Algorithms, Data structures, Operating systems….
    - Back in the "good old days" (~2009) one of the projects was to build a DBMS!!!

Database System Architectures & Application Development

- There are several different database architectures:
  - **File-server (embedded) architecture** - files are shared but DBMS processing occurs at the clients (e.g. Microsoft Access or SQLite)
    - You will work with this in Systems Programming 3410
  - **Two-Tier client-server architecture** - dedicated machine running DBMS accessed by clients (e.g. SQL Server)
  - **Three-Tier client-server architecture** - DBMS is bottom tier, second tier is an application server containing business logic, top tier is clients (e.g. Web browser-Apache/Tomcat-Oracle)
    - i.e., a LAMP Stack

Two-Tier Client-Server Architecture

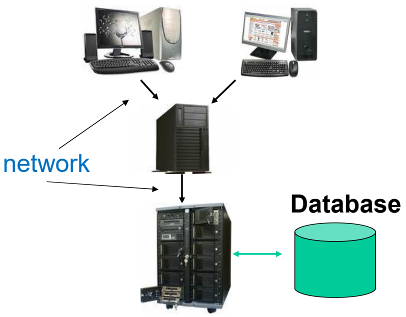


Tier 1: Client
- User Interface
- Business and Data Logic

Tier 2: Database Server
- Server-side validation
- Data storage/management- DBMS

- Advantages:
  - Only one copy of DBMS software on dedicated machine.
  - Increased performance.
  - Reduced hardware and communication costs.
  - Easier to maintain consistency and manage concurrency.

Three-Tier Client-Server Architecture – our approach



Tier 1: Client (Web/mobile)
- User Interface

HTML/CSS/Javascript

Tier 2: Application Server
- Business logic
- Data processing logic

PHP
Webserver/Apache

Tier 3: Database Server
- DBMS
  - Data storage/management

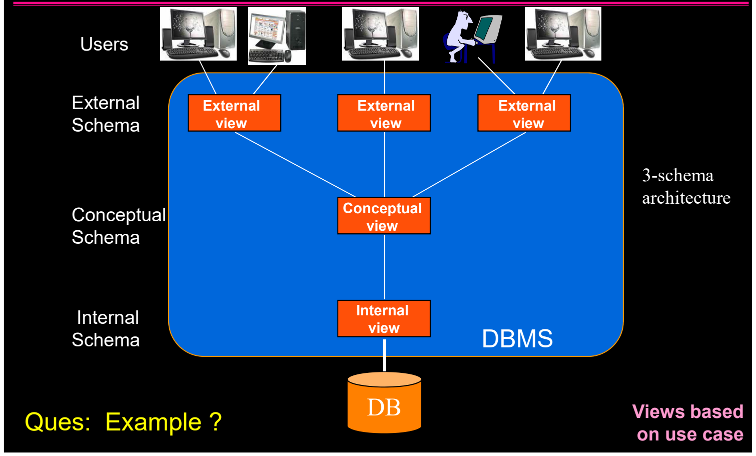
MySQL

- Advantages:
  - Reduced client administration and cost using thin web clients.
  - Easy to scale architecture and perform load balancing.

Proposed in 1975!!

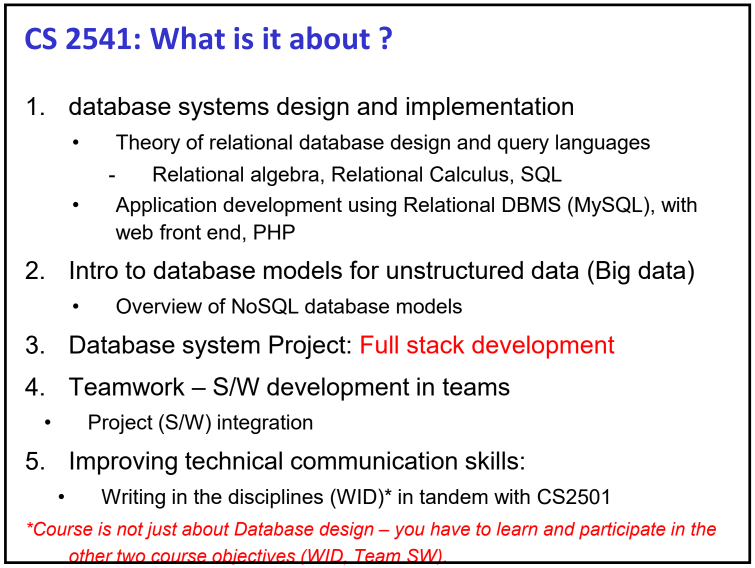
★

ANSI/SPARC Architecture



Ques: Example ?

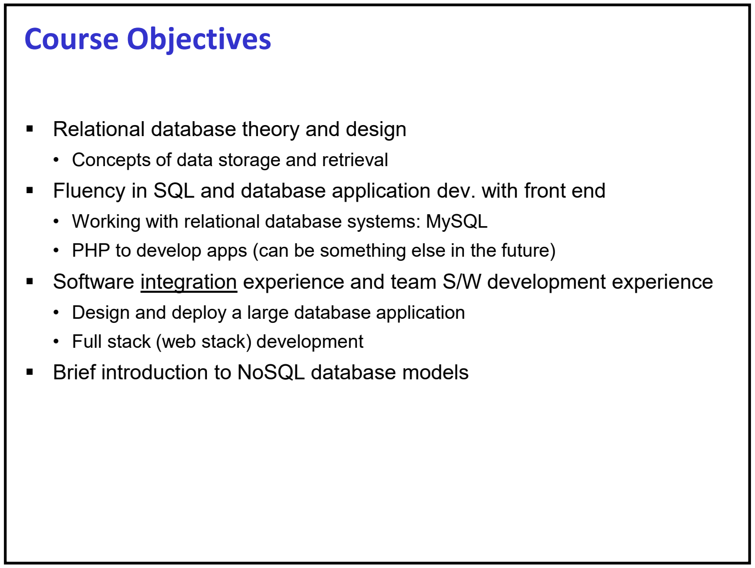
Views based on use case



CS 2541: What is it about ?

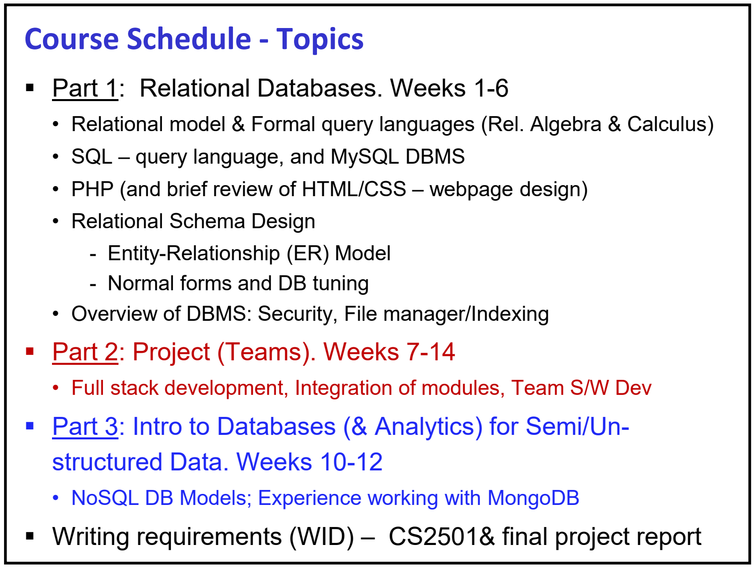
1. database systems design and implementation
   - Theory of relational database design and query languages
     - Relational algebra, Relational Calculus, SQL
   - Application development using Relational DBMS (MySQL), with web front end, PHP
2. Intro to database models for unstructured data (Big data)
   - Overview of NoSQL database models
3. Database system Project: Full stack development
4. Teamwork – S/W development in teams
   - Project (S/W) integration
5. Improving technical communication skills:
   - Writing in the disciplines (WID)* in tandem with CS2501

**Course is not just about Database design – you have to learn and participate in the other two course objectives (WID, Team SW).*



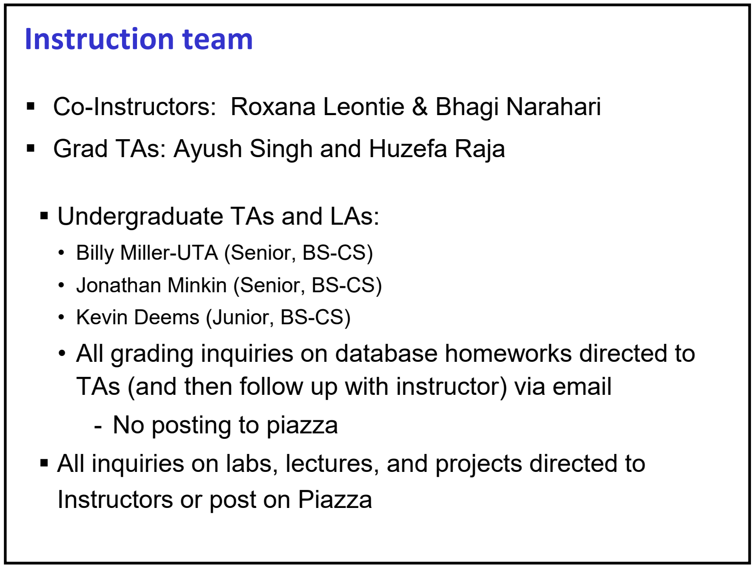
Course Objectives

- Relational database theory and design
  - Concepts of data storage and retrieval
- Fluency in SQL and database application dev. with front end
  - Working with relational database systems: MySQL
  - PHP to develop apps (can be something else in the future)
- Software integration experience and team S/W development experience
  - Design and deploy a large database application
  - Full stack (web stack) development
- Brief introduction to NoSQL database models



Course Schedule - Topics

- Part 1: Relational Databases. Weeks 1-6
  - Relational model & Formal query languages (Rel. Algebra & Calculus)
  - SQL – query language, and MySQL DBMS
  - PHP (and brief review of HTML/CSS – webpage design)
  - Relational Schema Design
    - Entity-Relationship (ER) Model
    - Normal forms and DB tuning
  - Overview of DBMS: Security, File manager/Indexing
- Part 2: Project (Teams). Weeks 7-14
  - Full stack development, Integration of modules, Team S/W Dev
- Part 3: Intro to Databases (& Analytics) for Semi/Un-structured Data. Weeks 10-12
  - NoSQL DB Models; Experience working with MongoDB
- Writing requirements (WID) – CS2501& final project report



Instruction team

- Co-Instructors: Roxana Leontie & Bhagi Narahari
- Grad TAs: Ayush Singh and Huzefa Raja

- Undergraduate TAs and LAs:
  - Billy Miller-UTA (Senior, BS-CS)
  - Jonathan Minkin (Senior, BS-CS)
  - Kevin Deems (Junior, BS-CS)
  - All grading inquiries on database homeworks directed to TAs (and then follow up with instructor) via email
    - No posting to piazza
- All inquiries on labs, lectures, and projects directed to Instructors or post on Piazza

In-class work

- You will learn through in-class activities/demos and exercises most classes (lecture+lab)
  - Must read the material and come to class
- If you are assigned an exercise during class (i.e., an "in class exercise"), you need to complete the exercises by the end of the class – no exceptions!
  - Each team is assigned to a table
  - We may ask a team to present solutions to class
- *If you do not come prepared to class/lab it is not going to be smooth sailing….*

Course Materials – "confusion will be my epitaph"!

- Course webpage – will have links to syllabus, lecture notes, online resources (and inclass exercises when applicable)
  - www.seas.gwu.edu/~bhagiweb/cs2541
  - Teams will be posted on this page
- Github – please make sure you have an active account before Wednesday!
  - Used to post and submit 'lab' assignments (requiring code)
  - Project submissions
  - Team project development
- Blackboard will be used for:
  - Homeworks and solutions, Project posting and team assignment
  - Electronic submission of non-programming homeworks
  - Reporting grades
- Piazza – for discussions

Piazza

- you've used this before, so you know the protocols:
  - The purpose of this:
    - to encourage you to ask and answer questions
      - Most of the time, you do better than we do!
    - *Be very careful not to border on plagiarism!*
    - *Don't post your HW solution to the world,*
  - Signup email sent…check your piazza account
  - Do not expect instant response or substitute Piazza for office hours!
    - Piazza is not manned 24 hours/7 days a week
    - Sometimes an answer may take more than 24 hours!
  - NO TA can excuse you from anything/or give any extensions
  - Posting on piazza, not the same as telling instructor things
    - E.g. : I'm going to miss the exam!
  - Do **NOT** wait until the last minute to ask for clairifications…
    - The instructors & TAs do NOT plan on spending their weekend checking Piazza!

Textbooks/Software

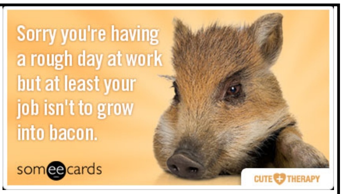
- Textbook:
  - Online notes and resources
    - Suggested readings/resources linked from course webpage (go to Lectures)
  - Reference books (if you want to purchase a textbook) are also listed in the syllabus
    - But you could do just as well with most any Database textbook
- MySQL and PHP…
  - You can install it locally on your laptop
  - We will use the install on SEAS server – gwupyterhub.seas.gwu.edu
- MongoDB (an example NoSQL database)
- Explore setting up your own DB services on AWS
  - We may have a short session on how to do this

Course Requirements: Grading

- Exam (midterm): 22.5%
  - Closed book, based on lectures and labs
  - Approximately weeks 6/7
- Homeworks, Lab Assignments, In-class: 35%
  - Homeworks include programming homeworks
  - In-Lab/Class exercises given out during class & equivalent to a "quiz"
- Team Project (and Teamwork): 42.5%
  - Phase 1 (15%) + Teamwork (7.5%) + Phase 2 (20%)
  - No final exam BUT final project demos are required
  - To pass project, your demos have to work…NO partial credit.
- Grades curved (and scaled as percentage of highest score in class)
  - Approximate grading method after curving and scaling

    A- to A: 90-100%   B- to B+ : 80 to < 90   C- to C+: 70 to < 80   D-: >60

The Project


Sorry you're having a rough day at work but at least your job isn't to grow into bacon.
someecards
CUTE THERAPY

- A significant part of your grade for the course is a large database systems project.
- In the project you will design & mplement a database system Full stack development:
  - Front End (HTML/CSS & optional Javascript)
  - Application server – in PHP
  - DBMS backend – MySQL
  - All the above are useful (high demand) skills
    - Note that limited background will be given on web programming.
- The project will involve working in teams of 2 to 4.
  - Larger teams must develop projects with more features.

Team Project: Requirements & Expectations

- Project broken into 2 (+ 0.1) phases:
  - Phase 0.1: theoretical (paper) design of the database (ER model)
  - Phase 1: teams to build an application assigned to your team
  - Phase 2: Work in new teams to integrate different applications and produce the final project
    - Different teams may be assigned different projects
    - This requires ***integration*** and NOT redesign
    - You take what you built in Phase 1 and integrate with systems built by others….
      - If you "hide" in Phase 1, then you will be exposed in Phase 2 !!
- You HAVE to deliver a working project…else Zero on project
- Agile SW Development process
  - Build the system iteratively rather than all in one (giant) step
  - works well with your teamwork assessment (weekly check-ins)

Why do we have team projects (team S/W development) ?

- Real World: Teamwork and S/W development in teams is the default!
  - Communication
  - Collaboration
  - Conflict resolution
  - Addition of using tools to enable collaborative SW dev (Github)
- And yeah, ABET accreditation requires this too!

Teamwork Assessment…part of your grade!

- You have to work in teams
  - Each team member required to ‘produce’ equitable share ‘product’
  - Teamwork will be assessed…
    - Not all team members may get the same grade on the project!
    - You must bring teamwork issues to attention of the instructor
- The second half of the course will have one session (lecture or a lab) dedicated to teamwork check-ins
  - Instruction team will meet with each team, and assess if the weekly deliverables are being met by each team member
- If you cannot commit time each week to working on the team project then please drop the course!
  - If you do not want to work in a team and do the work, then we do not want your attitude to negatively impact other students

Lab Sections: treated as one lab section

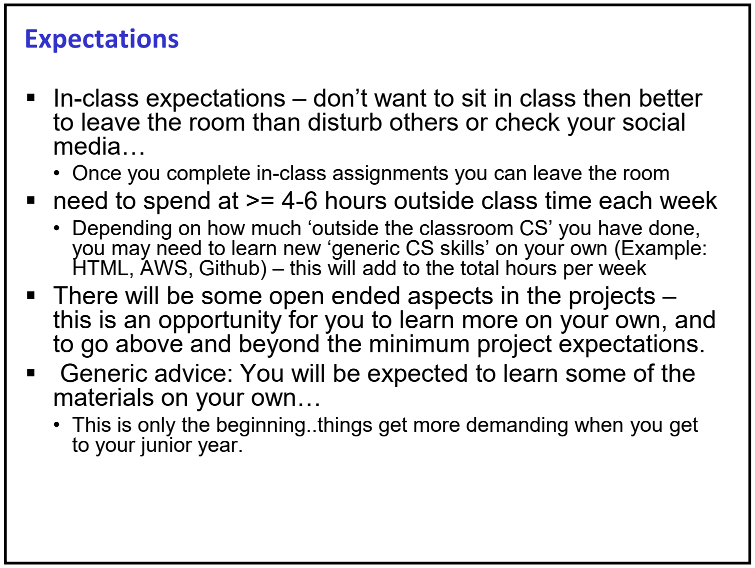
- Lab sections conducted by the instructor(s) and TAs:
  - Lead Instructor for Labs: Roxana
- Lab sections will cover
  - Intro MySQL
  - Short tutorials – including application development using
  - PHP, Front end
    - CSS? HTML ?
    - Javascript – tutorials provided by Kevin during office hours
  - Intro to a NoSQL DB - MongoDB
  - Clarifications on Programming Assignments
- In-class assignments in some weeks
  - Example: have to implement SQL queries during class; no extensions!..treated as a quiz

Academic Integrity Policy

- No collaboration (of any sort) on homeworks/ programming assignments
  - Including external resources, tutors, online
  - Okay to clarify questions
  - Not Okay to share solutions
  - Not okay under any circumstances to share or show Code
- No collaboration between teams on team projects
  - within team each team member must have clear role -- i.e., clearly partitioned tasks for each team member

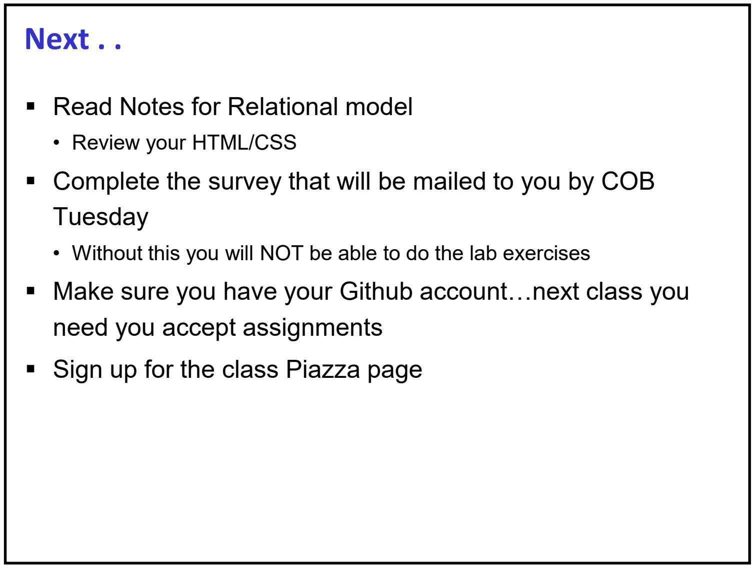
Academic Integrity

- Strictly enforced! You are here to learn – so keep that in mind
- Today’s CS job process: Technical interview is the first step – employers do not care about your 4.0 GPA if you do not pass the first technical interview!
  - Grad schools (for PhD) don’t care about high GPA if you do not have independent research experience (ideally with a published work)
- Violations will lead to at least a zero on the work and a grade lower than final grade
- Stay on top of your work – and come ask us questions!
- PDT: Plagiarism detection software tool
  - We may be running code submissions through software tool
  - Any pair of submissions with more than 25% similarity will be closely examined



Expectations

- In-class expectations – don’t want to sit in class then better to leave the room than disturb others or check your social media…
  - Once you complete in-class assignments you can leave the room
- need to spend at >= 4-6 hours outside class time each week
  - Depending on how much ‘outside the classroom CS’ you have done, you may need to learn new ‘generic CS skills’ on your own (Example: HTML, AWS, Github) – this will add to the total hours per week
- There will be some open ended aspects in the projects – this is an opportunity for you to learn more on your own, and to go above and beyond the minimum project expectations.
- Generic advice: You will be expected to learn some of the materials on your own…
  - This is only the beginning..things get more demanding when you get to your junior year.



Next . .

- Read Notes for Relational model
  - Review your HTML/CSS
- Complete the survey that will be mailed to you by COB Tuesday
  - Without this you will NOT be able to do the lab exercises
- Make sure you have your Github account…next class you need you accept assignments
- Sign up for the class Piazza page