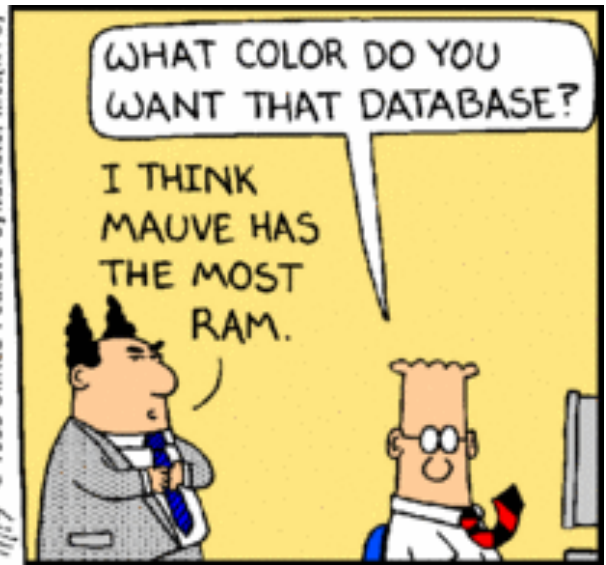


S. Adams E-mail: SCOTTADAMS@AOL.COM



1/17 © 1995 United Feature Syndicate, Inc.(NYC)



Databases

Lecture 1

Standard stuff

- Class webpage
- Textbook: get it somewhere; used is fine
 - Stay up with reading!
- Prerequisite: CS 241
- Coursework:
 - Homework, group project, midterm, final
- Be prepared to bring laptops every so often.

Group project

- You will design and implement your own database-driven website.
- Ideas: shopping, auctions, write a better BannerWeb, library/bibliography system, reviews a la Yelp, bank, finance/stocks, job postings, social networking a la Facebook, recipes, movies, apartments, ...
- Groups: probably 4-5 people, formed on your own.
- Spread out over the whole semester; check-ins along the way.

Why study databases?

- Academic reasons
- Programming reasons
- Business (get a job) reasons
- Student reasons

What will you learn?

- Database design
 - How do you model your data so it can be stored in a database?
- Database programming
 - How do I use a database to ask it questions?
- Database implementation
 - How does the database itself work; i.e., how does it store, find, and retrieve data efficiently?

What is the goal of a database?

- Electronic record-keeping, enabling **fast** and **convenient** access to the information inside.
- DBMS = Database management system
 - Software that stores individual databases and knows how to search the information inside.
 - RDBMS = Relational DBMS
 - Examples: Oracle, MS SQL Server, MS Access, MySQL, PostgreSQL, IBM DB2, SQLite

DBMS Features

- Support massive amounts of data
 - Giga-, tera-, petabytes
- Persistent storage
 - Data continues to live long after program finishes.
- Efficient and convenient access
 - Efficient: don't search the entire thing to answer a question!
 - Convenient: allow users to ask questions as easily as possible.
- Secure, concurrent, and atomic access

Example: build a better BannerWeb

- Professors offer classes, students sign up, get grades
- What are some questions we could ask?
 - Find my GPA.
 - ...

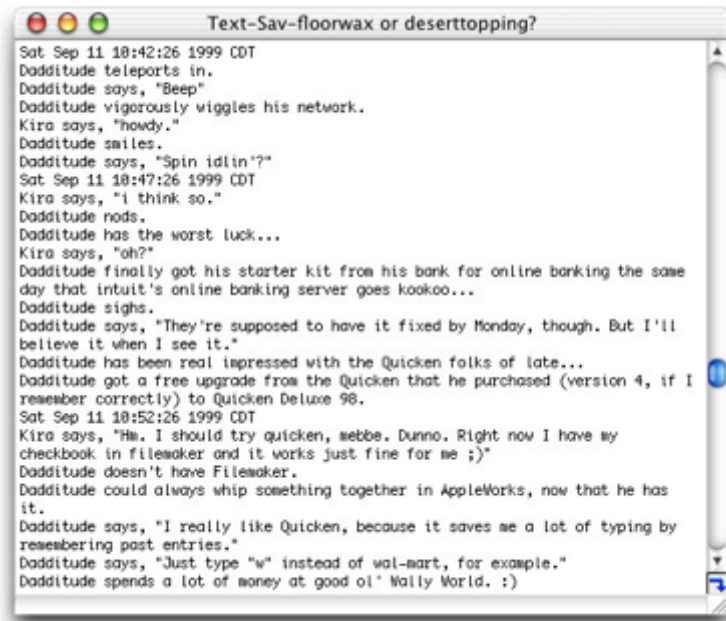
Obvious solution: Folders

- Advantages?
- Disadvantages?

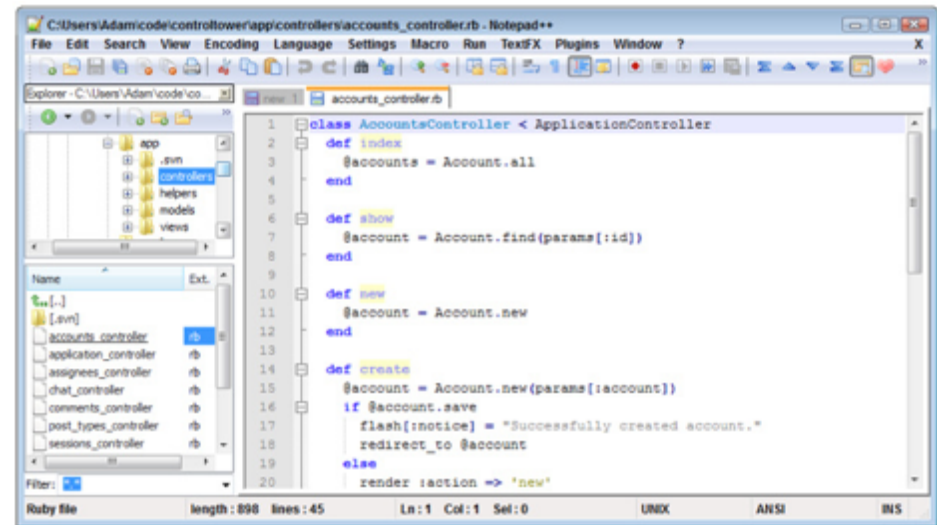


Obvious solution++

- Text files and Python/C++/Java programs



```
Sat Sep 11 18:42:26 1999 CDT
Dadditude teleports in.
Dadditude says, "Beep"
Dadditude vigorously wiggles his network.
Kira says, "howdy."
Dadditude smiles.
Dadditude says, "Spin idlin'?"
Sat Sep 11 18:47:26 1999 CDT
Kira says, "I think so."
Dadditude nods.
Dadditude has the worst luck...
Kira says, "oh?"
Dadditude finally got his starter kit from his bank for online banking the same
day that intuit's online banking server goes kookoo...
Dadditude sighs.
Dadditude says, "They're supposed to have it fixed by Monday, though. But I'll
believe it when I see it."
Dadditude has been real impressed with the Quicken folks of late...
Dadditude got a free upgrade from the Quicken that he purchased (version 4, if I
remember correctly) to Quicken Deluxe 98.
Sat Sep 11 18:52:26 1999 CDT
Kira says, "Ha. I should try quicken, sebba. Dunno. Right now I have my
checkbook in filemaker and it works just fine for me ;)"
Dadditude doesn't have Filemaker.
Dadditude could always whip something together in AppleWorks, now that he has
it.
Dadditude says, "I really like Quicken, because it saves me a lot of typing by
remembering past entries."
Dadditude says, "Just type 'w' instead of wal-mart, for example."
Dadditude spends a lot of money at good ol' Nally World. :)
```



```
class AccountsController < ApplicationController
  def index
    @accounts = Account.all
  end

  def show
    @account = Account.find(params[:id])
  end

  def new
    @account = Account.new
  end

  def create
    @account = Account.new(params[:account])
    if @account.save
      flash[:notice] = "Successfully created account."
      redirect_to @account
    else
      render :action => 'new'
    end
  end
end
```

Obvious solution++

- Let's use CSV:



Hermione,Granger,R123,Potions,A

Draco,Malfoy,R111,Potions,B

Harry,Potter,R234,Potions,A

Ronald,Weasley,R345,Potions,C

Another way:

File 1:

Hermione, Granger, R123

Draco, Malfoy, R111

Harry, Potter, R234

Ronald, Weasley, R345

File 2:

R123, Potions, A

R111, Potions, B

R234, Potions, A

R345, Potions, C

Problems

- Inconvenient – need to know Python/C++/Java to get at data!
- Redundancy/inconsistency
- Integrity problems
- Atomicity problems
- Concurrent access problems
- Security problems

Why are there problems?

- Two main reasons:
 - The description of how the files are laid out is buried within the Python/C++/Java code itself (if it's documented at all)
 - There is no support for **transactions** (supporting concurrency, atomicity, integrity, and recovery)
- **DBMSs handle exactly these two problems.**

Example

- RDBMS = Relational database management system.
- The relational model uses relations (aka tables) to structure data. (CS 172, boom!)
- Grades relation:



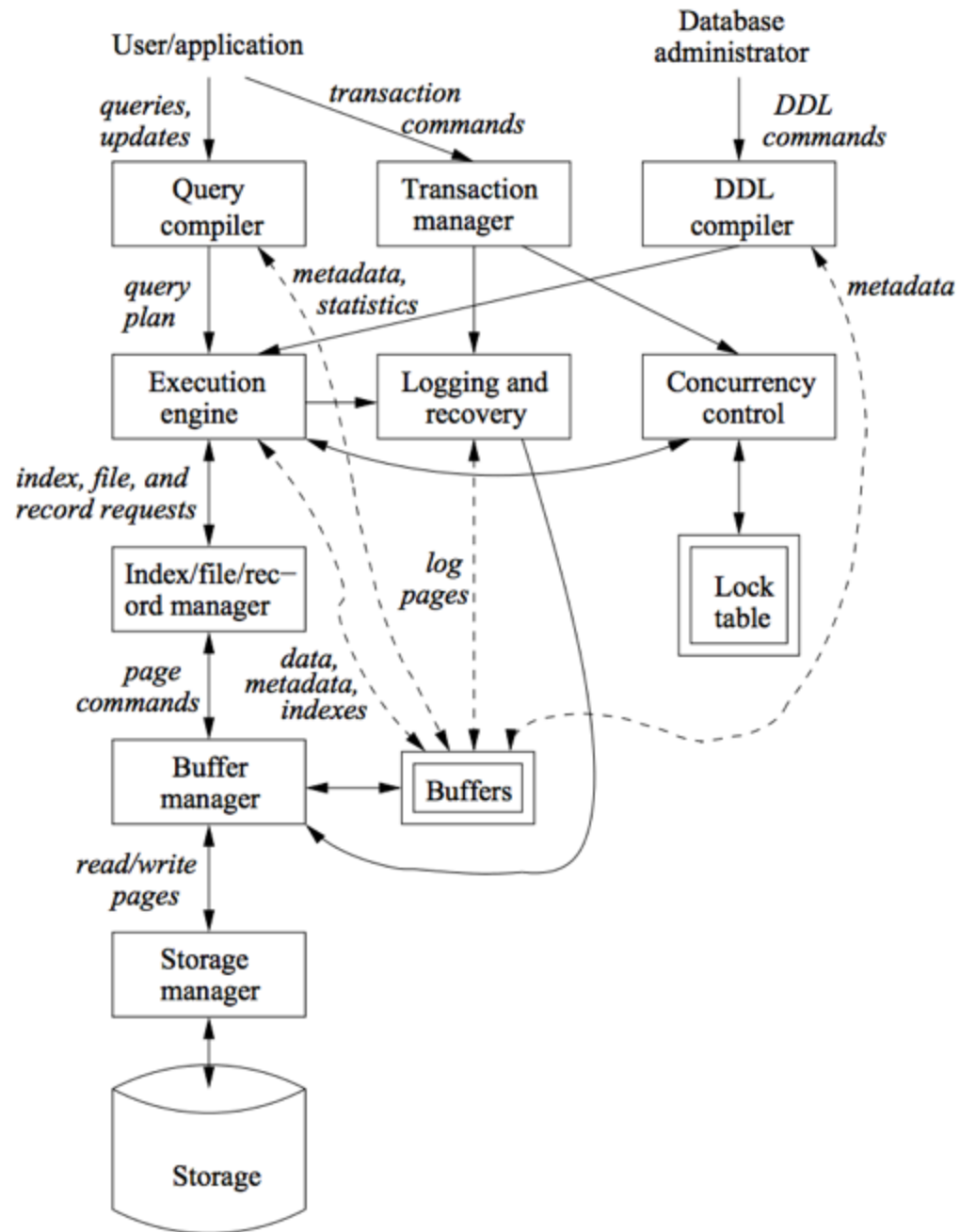
First	Last	Course	Grade
Hermione	Granger	Potions	A
Draco	Malfoy	Potions	B
Harry	Potter	Potions	A
Ronald	Weasley	Potions	C

First	Last	Course	Grade
Hermione	Granger	Potions	A
Draco	Malfoy	Potions	B
Harry	Potter	Potions	A
Ronald	Weasley	Potions	C

- Relational model is an abstraction.
- Separates the **logical view** (as viewed by the DB user) from the **physical view** (DB's internal representation of the data)

First	Last	Course	Grade
Hermione	Granger	Potions	A
Draco	Malfoy	Potions	B
Harry	Potter	Potions	A
Ronald	Weasley	Potions	C

- Simple query language (SQL) for accessing/modifying data:
- Find all students who are getting a B.
 - `SELECT First, Last FROM Grades WHERE Grade = "B"`



Transaction processing

- One or more DB operations can be grouped into a **transaction**.
- For a DBMS to properly implement transactions:
- **A**tomicity: All-or-nothing execution of transactions.
- **C**onsistency: A DB can have consistency rules that should not be violated.
- **I**solation: Each transaction must **appear** to be executed as if no other transactions are happening simultaneously.
- **D**urability: Any changes a transaction makes must never be lost.

On to the real stuff now...

Data Models

- A notation (description) of a description of data.
 - Better: a description of how to conceptually structure the data, what operations are possible on the data, and any constraints on the data.
- Structure: how we view the data abstractly
- Operations: what is possible do do with the data?
- Constraints: how can we control what data is legal and what is not?

Relational model

First	Last	Course	Grade
Hermione	Granger	Potions	A
Draco	Malfoy	Potions	B
Harry	Potter	Potions	A
Ronald	Weasley	Potions	C

- Structure: relation (table)
- Operations: relational algebra (select certain rows, certain columns, where things are T/F)
- Constraints: can enforce restrictions like Grade must be in {A, B, C, D, F}

Semi-structured model

```
<Grades>
  <StudentGrade>
    <Student>Hermione Granger</Student>
    <Course>Potions</Course>
    <Grade>A</Grade>
  </StudentGrade>
  <StudentGrade>
    <Student>Draco Malfoy</Student>
    <Course>Potions</Course>
    <Grade>B</Grade>
  </StudentGrade>
  ...
</Grades>
```


Semi-structured model

- Structure: Trees or graphs
 - e.g., XML
- Operations: Follow paths in the implied tree from one element to another.
 - e.g., XQuery
- Constraints: can constrain data types, possible values, etc.
 - e.g., DTDs (document type definition), XML Schema

Object-relational

- Similar to relational, but
 - Values in a table can have their own structure, rather than being simple strings or ints.
 - Relations can have associated methods.

Relational model is most common

- Simple: built around a single concept for modeling data: the relation or table.
 - A relational database is a collection of relations.
 - Each relation is a table with rows and columns.
 - An RDBMS can manage many databases at once.
- Supports high-level programming language (SQL)
 - Limited but useful set of operations.
- Has elegant mathematical theory behind it.

Relation Terminology

- Relation == 2D table
 - **Attribute** == column name
 - **Tuple** == row (not the header row)
- Database == collection of relations

First	Last	Course	Grade
Hermione	Granger	Potions	A
Draco	Malfoy	Potions	B
Harry	Potter	Potions	A
Ronald	Weasley	Potions	C

Relation Terminology

- A relation includes two parts:
 - The relation **schema** defines the column headings of the table (attributes/fields)
 - The relation **instance** defines the data rows (tuples, rows, or records) of the table.

First	Last	Course	Grade
Hermione	Granger	Potions	A
Draco	Malfoy	Potions	B
Harry	Potter	Potions	A
Ronald	Weasley	Potions	C

Schema

- A schema is written by the name of the relation followed by a parenthesized list of attributes.
 - Grades(First, Last, Course, Grade)
- A **relational database schema** is the set of schemas for all the relations in a DB.

First	Last	Course	Grade
Hermione	Granger	Potions	A
Draco	Malfoy	Potions	B
Harry	Potter	Potions	A
Ronald	Weasley	Potions	C

Domains

- A relational DB requires that every component of a row (tuple) have a specific elementary data type, or **domain**.
 - string, int, float, date, time (no complicated objects!)

**Grades(First:string, Last:string,
Course:string, Grade:char)**

Equivalent representations of a relation

First	Last	Course	Grade
Hermione	Granger	Potions	A
Draco	Malfoy	Potions	B
Harry	Potter	Potions	A
Ronald	Weasley	Potions	C

Grades(First, Last, Course, Grade)

- Relation is a **set** of tuples, not a list.
- Attributes in a schema are a **set** as well.
 - However, the schema specifies a "standard" order for the attributes.
- How many equivalent representations are there for a relation with m attributes and n tuples?