CS 455
Principles of Database Systems
Course Goals

- To understand modern solutions to problems related to the management of data
  - Centered around the relational data model

- Major course topics:
  - Relational Data Model
  - Relational Algebra
  - SQL
  - Relational Database Theory and Design
  - Database Performance Considerations
  - Transaction Management
Course Requirements

- Required Textbook

- Assumptions
  - Mastery of CS 261 topics
  - Basic understanding of set theory

- Desirable (but not necessary)
  - Familiarity of using Unix-based systems
  - Familiarity of using the command line
Course Page

- Course page
  - http://cs.pugetsound.edu/~dchiu/cs455
  - Weekly schedule, assignments
  - Notes (write down the following info)
    - Username: CS455
    - Password: p4ssword

- Submission page:
  - http://canvas.pugetsound.edu
  - Use drop boxes for homework and project submission
How to Reach Me

- How to reach me:
  - Office: Thompson Hall 390 B
  - Slack: univpugetsound.slack.com
  - Email: dchiu@pugetsound.edu
  - Phone: 2853 on campus line

- Office Hours: Open-door
  - When door is cracked/open: Walk in, Knock
  - When door is shut: Unavailable (see note on door)
  - You can also make an appointment
Grading

- Breakdown of Student Assessment
  - 33% - Homework Assignments
  - 20% - Team Project
  - 15% - Midterm Exam I
  - 15% - Midterm Exam II
  - 15% - Final Exam
  - 2% - Discretionary
Homework

- You can expect 4 to 6 homework assignments
  - Based on lectures and readings
  - Problem solving, coding

- Conditions:
  - -10% each day late (includes weekend days)

- Work on these alone!
Team Projects

- Work in teams of 3 - 4 (max)
  - Email me team membership by **this friday**
    (I will assign teams otherwise)

- Team projects culminate in a data-driven web app
  - Project I: Server Administration
  - Project II: Learning PHP + DB connectivity
  - Project III: Planning and Database Design
  - Project IV: Implementation
Team Projects (Cont.)

- Project Demo (Graded Check-ins)
  - Teams will schedule meeting with me offline to demo each phase
  - Be ready to answer questions. You will be graded on this.

- Project Policies
  - Equal amounts of work expected from each team member
    - Both writing and coding
  - -10% each day late (includes weekend days)
  - Previously unsuccessful projects
    - Usually stemming from poor communication between members
    - Workload imbalance
Exams

› Two midterms and a final will be based on
  • Lectures, homework, readings
  • Comprehensive, but weighted more heavily on new materials

› I will not test on anything I did not specifically cover in class
  • But things covered in class, but isn’t found in the book is fair game
    - This will happen, so don’t miss class!

› Calculator and full-page cheat sheet (front/back) allowed
Classroom Participation

- I want class to be **interactive**
  - Ask questions when you don't understand something
    - If questions are irrelevant to lecture, I'll be happy to answer it after class
  - I will pose lots questions each lecture
    - Don't be shy, or I'll pick on you

- Be careful how you present yourself
  - For many here, CS is completely new.
  - What you say may come across as condescending and discouraging to others
Tips on Note Taking for CS 455

- Slides are always available via the course web page
  - **DON'T:** spend your time copying down info on slides

- I do lots of drawing/illustrations
  - **DO:** Copy these down in your notes

- I do lots of code writing (relational algebra, SQL, etc.)
  - Both on the board, and within SQLite
  - **DO:** Copy these down
If you have a physical, psychological, medical, or learning disability, contact:

- Peggy Perno
- Office of Student Accessibility and Accommodation (SAA)
- 105 Howarth Hall
- 253.879.3395

Communications with SAA is confidential
Outline

- Course Syllabus
- What Is a Database?
- Life without Database Management Systems
- Features of a Good Data Management System (DBMS)
- Intro to the Relational Model
- Conclusion
What Is a Database?

- **Database**
  - A collection of *inter-related* data for "mini-worlds"
  - Mini-worlds represent some aspect of the real world

- Example mini-world and database: Amazon
  - Catalog: ~ 1 billion products, music, videos
  - Inventory needs to be continuously updated
    - ~ 40 purchases made per second
    - ~ Thousands of items added each day
Amazon Mini-World

Marketplace Department

Order

Product

Seller

Customer

Cloud computing Department, ...

Human Resources Department

Employee

...
Example: Amazon Mini-World
Inter-Related Data

Who stocks what?

Order

Shipping

Receipt

Product

Name

Qty

Price

... 

Employee

Name

# dep

Hours

Seller

Company

Name

... 

Customer

Name

Addr

CC#
Inter-Related Data

- **Order**: What was ordered? Who fulfills an order?
- **Product**: Who stocks what?
- **Seller**: Who sells what?
- **Customer**: Who ordered it?
- **Employee**: What was ordered?
- **Shipping**: ...
What Makes Data Management Hard?

- “We need an entire course on files?”
  - Files are insufficient for data management.

- Consider two departments that manage their own sets of data:

<table>
<thead>
<tr>
<th>Marketplace Department</th>
<th>Human Resources Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT_NAME, QTY, PRICE, STOCKED_BY, HR_WORKED, LEAVE</td>
<td>NAME, HOURS, ON_LEAVE, ADDR</td>
</tr>
<tr>
<td>Bike, 42, $198.00, Adam, 36, no</td>
<td>Adam, 36, no, Tacoma</td>
</tr>
<tr>
<td>Umbrella, 203, $10.99, Julie, 44, yes</td>
<td>Brad, 40, no, Vashon Is. America</td>
</tr>
<tr>
<td>Chair, 3, $29.99, Brad, 40, no</td>
<td>America, 40, no, Tacoma</td>
</tr>
<tr>
<td>Puppy, 0, $300.00, Adam, 36, no</td>
<td>...</td>
</tr>
</tbody>
</table>
Data entry: Suppose an employee is not detail-oriented

- Hours for Brad is now inconsistent. Need to contact HR Dept

Marketplace Department

<table>
<thead>
<tr>
<th>PRODUCT_NAME</th>
<th>QTY</th>
<th>PRICE</th>
<th>STOCKED_BY</th>
<th>HR_WORKED</th>
<th>LEAVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bike</td>
<td>42</td>
<td>$198.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>Umbrella</td>
<td>203</td>
<td>$10.99</td>
<td>Julie</td>
<td>44</td>
<td>yes</td>
</tr>
<tr>
<td>Chair</td>
<td>3</td>
<td>$29.99</td>
<td>Brad</td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td>Puppy</td>
<td>0</td>
<td>$300.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>iPhone</td>
<td>106</td>
<td>$899.99</td>
<td>Brad</td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Earplugs</td>
<td>200</td>
<td>$5.99</td>
<td>Brad</td>
<td>4</td>
<td>no</td>
</tr>
</tbody>
</table>

Oops meant to type 40

Human Resources Department

<table>
<thead>
<tr>
<th>NAME</th>
<th>HOURS</th>
<th>ON_LEAVE</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>36</td>
<td>no</td>
<td>Tacoma</td>
</tr>
<tr>
<td>Brad</td>
<td>40</td>
<td>no</td>
<td>Vashon Is.</td>
</tr>
<tr>
<td>America</td>
<td>40</td>
<td>no</td>
<td>Tacoma</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Managing Inter-Relations: Adam takes some time off, contacts HR, who updates their own file, but forgets to contact Marketplace dept.

- Data now *inconsistent* again

### Marketplace Department

<table>
<thead>
<tr>
<th>PRODUCT_NAME</th>
<th>QTY</th>
<th>PRICE</th>
<th>STOCKED_BY</th>
<th>HR_WORKED</th>
<th>LEAVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bike</td>
<td>42</td>
<td>$198.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>Umbrella</td>
<td>203</td>
<td>$10.99</td>
<td>Julie</td>
<td>44</td>
<td>yes</td>
</tr>
<tr>
<td>Chair</td>
<td>3</td>
<td>$29.99</td>
<td>Brad</td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td>Puppy</td>
<td>0</td>
<td>$300.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>iPhone</td>
<td>106</td>
<td>$899.99</td>
<td>Brad</td>
<td>40</td>
<td>no</td>
</tr>
</tbody>
</table>

... Earplugs, 200, $5.99, Brad, 40, no

### Human Resources Department

<table>
<thead>
<tr>
<th>NAME</th>
<th>HOURS</th>
<th>ON_LEAVE</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>36</td>
<td>yes</td>
<td>Tacoma</td>
</tr>
<tr>
<td>Brad</td>
<td>40</td>
<td>no</td>
<td>Vashon Is. America</td>
</tr>
</tbody>
</table>

...
Problems Galore (Cont.)

- Many more problems...
  - Employee changes name or quits
  - Marketplace adds a "product ID" field to distinguish each product
  - 2 customers buy the same product at roughly the same time

<table>
<thead>
<tr>
<th>Marketplace Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT_NAME, QTY, PRICE, STOCKED_BY, HR_WORKED, OVERTIME?</td>
</tr>
<tr>
<td>Bike, 42, $198.00, Adam, 36, no</td>
</tr>
<tr>
<td>Umbrella, 203, $10.99, Julie, 44, yes</td>
</tr>
<tr>
<td>Chair, 3, $29.99, Brad, 40, no</td>
</tr>
<tr>
<td>Puppy, 0, $300.00, Adam, 36, no</td>
</tr>
<tr>
<td>iPhone, 106, $899.99, Brad, 40, no</td>
</tr>
<tr>
<td>Earplugs, 200, $5.99, Brad, 40, no</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Human Resources Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME, HOURS, ON_LEAVE, ADDR</td>
</tr>
<tr>
<td>Adam, 36, no, Tacoma</td>
</tr>
<tr>
<td>Brad, 40, no, Vashon Is.</td>
</tr>
<tr>
<td>America, 40, no, Tacoma</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
C'mon, We're Programmers.

- Still not convinced
- Claim: Do we need a “system” to manage these files for us?
  - We'll *program* our way out of any data-management problem.

```java
public class abstract Store {
    protected File mktplace_file;
    protected File employee_file;

    public abstract double get_price(String product_name);
    public abstract void set_price(String product_name, double new_price);
    public abstract void add_account(String fullname, String new_address);
    public abstract void delete_account(String fullname);

    //.. others omitted
}
```
Get Balance from Marketplace File

Marketplace Department

<table>
<thead>
<tr>
<th>PRODUCT_NAME</th>
<th>QTY</th>
<th>PRICE</th>
<th>STOCKED_BY</th>
<th>HR_WORKED</th>
<th>LEAVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bike</td>
<td>42</td>
<td>$198.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>Umbrella</td>
<td>203</td>
<td>$10.99</td>
<td>Julie</td>
<td>44</td>
<td>yes</td>
</tr>
<tr>
<td>Chair</td>
<td>3</td>
<td>$29.99</td>
<td>Brad</td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td>Puppy</td>
<td>0</td>
<td>$300.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
</tbody>
</table>

public double get_price(String product_name) {
    Scanner file = new Scanner(mktplace_file);

    // TODO implement this!
}

...
Problem 1: Logical Data Dependence

- Can we "effortlessly update database schema?"

### Marketplace Department (old schema)

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Price</th>
<th>Name</th>
<th>Age</th>
<th>Married</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bike</td>
<td>42</td>
<td>$198.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>Umbrella</td>
<td>203</td>
<td>$10.99</td>
<td>Julie</td>
<td>44</td>
<td>yes</td>
</tr>
<tr>
<td>Chair</td>
<td>3</td>
<td>$29.99</td>
<td>Brad</td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td>Puppy</td>
<td>0</td>
<td>$300.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
</tbody>
</table>

### Marketplace Department (proposed schema)

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Price</th>
<th>Name</th>
<th>Age</th>
<th>Married</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Bike</td>
<td>42</td>
<td>$198.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>3 Umbrella</td>
<td>203</td>
<td>$10.99</td>
<td>Julie</td>
<td>44</td>
<td>yes</td>
</tr>
<tr>
<td>8 Chair</td>
<td>3</td>
<td>$29.99</td>
<td>Brad</td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td>2 Puppy</td>
<td>0</td>
<td>$300.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>6 Bike</td>
<td>32</td>
<td>$87.50</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
</tbody>
</table>

```java
public double get_price(String product_name) {
    Scanner file = new Scanner(mktplace_file);

    // TODO implement this!
}
```

**Problem:**
Changes to database schema requires changes to application code
Problem 2: Physical Data Dependence

- Can we "persistently store large data sets?"
  - Our marketplace explodes in growth and popularity
    - Product file getting too large (millions of records)

<table>
<thead>
<tr>
<th>Bike</th>
<th>42</th>
<th>$198.00</th>
<th>Adam</th>
<th>36</th>
<th>no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Umbrella</td>
<td>203</td>
<td>$10.99</td>
<td>Julie</td>
<td>44</td>
<td>yes</td>
</tr>
<tr>
<td>Chair</td>
<td>3</td>
<td>$29.99</td>
<td>Brad</td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td>Puppy</td>
<td>0</td>
<td>$300.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
</tbody>
</table>

(Millions of rows later)
Problem 2: Physical Data Dependence

- One solution: Split up the file onto *multiple* computers.
  - *What are the consequences for this change?*

```
<table>
<thead>
<tr>
<th>File</th>
<th>Record</th>
<th>Product</th>
<th>Price</th>
<th>Name</th>
<th>Age</th>
<th>Married</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod1.txt</td>
<td>0</td>
<td>Bike</td>
<td>$198.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Umbrella</td>
<td>$10.99</td>
<td>Julie</td>
<td>44</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Chair</td>
<td>$29.99</td>
<td>Brad</td>
<td>40</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Shimmie</td>
<td>$30.00</td>
<td>Sarah</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>prod2.txt</td>
<td>2</td>
<td>Puppy</td>
<td>$300.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Bike</td>
<td>$87.50</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td>prod305.txt</td>
<td>0</td>
<td>Puppy</td>
<td>$300.00</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>Bike</td>
<td>$87.50</td>
<td>Adam</td>
<td>36</td>
<td>no</td>
</tr>
</tbody>
</table>
```

(1000s more files and servers not shown)
Can we "persistently store large data sets?"

Splitting helps with performance, but risk of data loss increases.

- Solution: Replicate files onto *multiple* computers.

  - *What are the consequences of replication?*

(1000s more files and servers not shown)
Problem 3: Limitations to Queries

- Can we "Handle arbitrary questions on the data?"
  - Recall we wrote an API to deal with our file, specifically, so... no.

```java
public class abstract Store {
    protected File mktplace_file;
    protected File employee_file;

    public abstract double get_price(String product_name);
    public abstract void set_price(String product_name, double new_price);
    public abstract void add_account(String fullname, String new_address);
    public abstract void delete_account(String fullname);
}
```

**Some common queries:** *(Would need to write a method for EVERY query)*
1) Return the names of everyone who stocks chairs
2) Return the number of customers who live in Tacoma
3) Find the city with the lowest number of active customers
   ...

Outline

- Course Syllabus
- What Is a Database?
- Life without Database Management Systems
- Features of a Good Data Management System (DBMS)
- Intro to the Relational Model
- Conclusion
What are some things we need to support for databases?

- Persistently store (possibly) large data sets
What are some things we need to support for databases?

- Persistently store large data sets

  - Data must be persisted on disk (files), which is slow...
  - What format do I store the data?
  - How is the data accessed? Randomly? Sequentially?
  - How do deal with disk failure?
  - ...
What are some things we need to support for databases?

• Persistently store large data sets

• Efficiently query and update the database
  - Handle arbitrary questions about data
  - Handle complicated updates and deletions
What are some things we need to support for databases?

- Persistently store large data sets
- Efficiently query and update the database
  - Handle arbitrary questions about data
  - Handle complicated updates and deletions
    - New questions shouldn't require a new procedure to be written!

Easy question: Return the credit card number for "Brad"

Harder one: How much does the average Tacoma resident spend per year?

A complicated update: Increase an employee's pay +1.5% if they've worked overtime and are over 50.
What are some things we need to support for databases?

- Persistently store large data sets
- Efficiently query and update the database
  - Handle arbitrary questions about data
  - Handle complicated updates and deletions
- Effortlessly update the database schema

Add a field to store customers' credit scores.

Need to store info on sellers now.

Remove fields, split a file into two, etc.
What are some things we need to support for databases?

- Persistently store large data sets
- Efficiently query and update the database
  - Handle arbitrary questions about data
  - Handle complicated updates and deletions
- Effortlessly update the database schema
- Support simultaneous updates (concurrency control)
Outline

- Course Syllabus
- What Is a Database?
- Life without Database Management Systems
- Features of a Good Data Management System (DBMS)
- Intro to the Relational Model
- Conclusion
New Goal: Data Abstraction

- *Data Abstraction*

  - Applications and users shouldn't need to care about how underlying data is stored on disk
    - e.g., humans can still choose to view data hierarchically, but on file, it can be stored in a comma-delimited format.
New Goal: Data Abstraction

- **Data Abstraction**
  - Goal: Decouple data model (logical) from storage of data (physical)

![Diagram of data abstraction levels]

- **Logical Level**
  - e.g., hierarchical view of accounts? Relational view?

- **Physical Level**
  - e.g., flat files? XML? CSV files?
Data independence is a nice goal, but how?
- Relational Model (Codd, 1970)
  - First described by Edgar (Ted) Codd at IBM Research
    - 1981 Turing Award
  - Based on Set Theory

Data is stored in *relations* (or *tables*) with a fixed schema

### Relation Name

**Students**

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>class</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>David</td>
<td>53 N. 19th St</td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td>34</td>
<td>Alan</td>
<td>12 Orange Rd.</td>
<td>JR</td>
<td>4.0</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

Values of attributes are "atomic." (Can't break up)
Characteristics of the Relational Model

- **Relation schema vs. Relation**
  - Schemas are the blueprint; relations are content that meet specification
  - The "Students" schema might be defined:

```plaintext
Students(
    ID    : int (>= 0),
    name  : string,
    addr  : string,
    class : string in {FR, SO, JR, SR},
    gpa   : double in [0.0, 4.0]
)
```
Characteristics of the Relational Model

- **Relation schema vs. Relation**
  - Schemas are the blueprint; relations are content that meet specification.
  - The “Students" schema might be defined:
    ```
    Students(
        ID    : int (>= 0),
        name  : string,
        addr  : string,
        class : string in \{FR, SO, JR, SR\},
        gpa   : double in [0.0, 4.0]
    )
    ```

- Examples of "Students" relations:
  ```
  Students
  ID | name     | addr           | class | gpa \
 ----|----------|----------------|-------|-----
  12 | David    | 53 N. 19th St | SO    | 3.0 \
  34 | Alan     | 12 Orange      | JR    | 4.0 \
  0  | James    | 392 Moon Ct    | FR    | 2.1 \
  58 | Julia    | 98 Rocky Rd    | SR    | 3.2 \
  43 | Sam      | 492 Sundial    | SO    | 2.7 \
  ```
  ```
  Students
  ID | name     | addr           | class | gpa \
  ----|----------|----------------|-------|-----
  12 | David    | 53 N. 19th St | SO    | 3.0 \
  34 | Alan     | 12 Orange      | JR    | 4.0 \
  0  | James    | 392 Moon Ct    | FR    | 2.1 \
  58 | Julia    | 98 Rocky Rd    | SR    | 3.2 \
  43 | Sam      | 492 Sundial    | SO    | 2.7 \
  ```
Characteristics of the Relational Model (Cont.)

- Tuples
  - Each tuple represents a record of the relation schema
  - Tuple (row) ordering does not matter

  Students

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>clas</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>David</td>
<td>53 N. 19th St</td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td>34</td>
<td>Alan</td>
<td>12 Orange</td>
<td>JR</td>
<td>4.0</td>
</tr>
<tr>
<td>0</td>
<td>James</td>
<td>392 Moon Ct</td>
<td>FR</td>
<td>2.1</td>
</tr>
<tr>
<td>58</td>
<td>Julia</td>
<td>98 Rocky Rd</td>
<td>SR</td>
<td>3.2</td>
</tr>
<tr>
<td>43</td>
<td>Sam</td>
<td>492 Sundial</td>
<td>SO</td>
<td>2.7</td>
</tr>
</tbody>
</table>

- Attributes
  - Attribute (column) ordering matters
  - Its values are atomic

- (How would you represent a student having more than one name or address?)
Quick Overview of Sets

- Some terms:
  
  - A **set** is a collection of elements
    - No duplicate elements are allowed
    - No implied ordering of elements
  
  - A **multiset (or bag)** is a set that allows duplicate elements
    - Dijkstra referred to multisets as "bags"
  
  - An **ordered list** imposes an order on elements and allows duplicates
  
  - A **tuple** is an **ordered list** of a fixed length
Set Expression

- Sets can be expressed in several ways

  - **Roster Notation**
    \[ A = \{0, 1, 3, 4, 6\} \]
    
    Note: \( \emptyset \) or \( \{\} \) is called the *empty set* (the set containing no elements)

  - **Set-Builder Notation**
    \[ B = \{x \mid (x \mod 2) = 0 \land (x \geq 0)\} \]
    
    \( B = \{x \mid x \text{ is a nonnegative even integer}\} \)

  - **Venn Diagram**

```
Set Operations

- **Cardinality**: $|S|$ returns the number of elements in the set

  $|\emptyset| = 0 \quad |\{a, b, c\}| = 3$
Set Operations

- **Cardinality:** $|S|$ returns the number of elements in the set

  $$|\emptyset| = 0 \quad |\{a, b, c\}| = 3$$

- **Membership:** $x \in S$ is *true* if element $x$ is found in $S$ and *false* otherwise

  $$a \in \{\} = \text{false} \quad 64 \in B = \text{true}$$

  $$a \notin \{\} = \text{true} \quad b \in \{a, b, c\} = \text{true}$$
Set Operations

- **Cardinality:** \(|S|\) returns the number of elements in the set

  \[ |\emptyset| = 0 \quad |\{a, b, c\}| = 3 \]

- **Membership test:** \(x \in S\) is **true** if element \(x\) is found in \(S\) and **false** otherwise

  \[ a \in \{\} = \text{false} \quad 64 \in B = \text{true} \]

  \[ a \notin \{\} = \text{true} \quad b \in \{a, b, c\} = \text{true} \]

- **Set existence test:** \(\exists S\) is **true** if the set \(S\) is not empty, and **false** otherwise

  \[ \exists \{x | x \text{ is prime } \land x \leq 1\} = false \]
Set Operations (Cont.)

- **Union:** $S_1 \cup S_2$

- Let $A = \{0, 1, 3, 4, 6\}$ and $B = \{0, 4, 6, 8, 14\}$

\[
A \cup B = \{0, 1, 3, 4, 6, 8, 14\}
\]
Set Operations (Cont.)

- **Intersection**: \( S_1 \cap S_2 \)

- Let \( A = \{0, 1, 3, 4, 6\} \) and \( B = \{0, 4, 6, 8, 14\} \)

\[
A \cap B = \{0, 6, 4\}
\]
Set Operations (Cont.)

- **Set Difference:** \( S_1 \setminus S_2 \) (or \( S_1 - S_2 \))

Let \( A = \{0, 1, 3, 4, 6\} \) and \( B = \{0, 4, 6, 8, 14\} \)

\[
A \setminus B = \{1, 3\}
\]
Set Operations (Cont.)

- **Subset:** $S_1 \subseteq S_2$

- Assume: $A = \{0, 1, 3, 4, 6\} \quad B = \{x \mid x \text{ is a nonnegative even integer}\}$

\[
\{0, 2\} \subseteq A = \text{true} \\
A \not\subseteq B = \text{true} \\
\{16, 32, 64, 128\} \subseteq B = \text{true}
\]
Set Operations (Cont.)

- **Set Equivalence:** $S_1 = S_2$

- **Proper Subset:** $S_1 \subset S_2$
  
  \[
  \{c\} \subset \{c, d\} = \text{true} \quad \{c, d\} \subset \{c, d\} = \text{false}
  \]

- **Power Set:** $\mathcal{P}(S)$
  
  - Suppose $S = \{a, b, c\}$
    
    \[
    \mathcal{P}(S) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}
    \]
  
  - Question:
    
    - What's the cardinality of $\mathcal{P}(S)$?
Set Operations (Cont.)

- **Cartesian Product:** $S_1 \times S_2$

- Examples

\[
\{a, b\} \times \{1, 2, 3\} = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}
\]

\[
\{(CS, 161), (CS, 261), (CS, 361)\} \times \{(David, C), (Brad, R)\} = ?
\]

- Question:
  - How is $S_1 \times S_2$ defined?
  - What is $|S_1 \times S_2|$
Relational Model: Connection to Sets

- A *relational schema*, \( R(A_1, A_2, \ldots, A_n) \), describes the structure of a relation.
  - Example: 
    - Relation Name: *Students*
    - Attributes:
      | ID | name | addr | class | gpa |
      |----|------|------|-------|-----|
      |    |      |      |       |     |
      |    |      |      |       |     |
      |    |      |      |       |     |
      |    |      |      |       |     |
A **relational schema**, \( R(A_1, A_2, \ldots, A_n) \), describes the structure of a relation.

- Example:

  - **Relation Name**
  - **Attributes**

<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students</td>
<td>ID, name, addr, class, gpa</td>
</tr>
</tbody>
</table>

Each **attribute** \( A \) has a **domain**, denoted by \( \text{dom}(A) \)

- \( \text{dom}(A) \) is the set of possible values for \( A \)
- \( \text{dom}(ID) = ? \quad \text{dom}(\text{class}) = ? \)
Relational Model: Connection to Sets (cont.)

- A *tuple* is a row in the relation
  - Also called a record

- Suppose we have $R(X, Y)$, where $\text{dom}(X) = \{0, 1\}$ and $\text{dom}(Y) = \{a, b, c\}$. List all possible tuples in $R$.  

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>class</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>David</td>
<td>53 N. 19th St</td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td>34</td>
<td>Alan</td>
<td>12 Orange Rd.</td>
<td>JR</td>
<td>4.0</td>
</tr>
<tr>
<td>90</td>
<td>Erin</td>
<td>4939 Sawmill</td>
<td>FR</td>
<td>3.9</td>
</tr>
</tbody>
</table>
- **Relations** are instances of the fixed schema.
  - Is a *set* of tuples
  - One instance of Students relation

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>class</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>David</td>
<td>53 N. 19th St</td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td>34</td>
<td>Alan</td>
<td>12 Orange Rd.</td>
<td>JR</td>
<td>4.0</td>
</tr>
<tr>
<td>90</td>
<td>Erin</td>
<td>4939 Sawmill</td>
<td>FR</td>
<td>3.9</td>
</tr>
</tbody>
</table>

- Other instances of Students relation

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>class</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>David</td>
<td>53 N. 19th St</td>
<td>SO</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Returning to Data Abstraction Goal

Logical data independence

Physical data independence

Physical Level

Logical Level

Application

Physical Data Storage

- Physical data independence
  - e.g., flat files? XML? CSV files?

- Logical data independence
  - e.g., hierarchical view of accounts? Relational view?
**Logical Data Independence**

- Changes to the logical level don't affect applications
  - Adding/changing attributes, tables, tuples should not require re-coding the application

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>class</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>David</td>
<td>53 N. 19th St</td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td>34</td>
<td>Alan</td>
<td>12 Orange Rd.</td>
<td>JR</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>
Relational model enables the use of a *system catalog*, which can store (among other things) attribute positions and offsets.

**DB System Catalog**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Start</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>name</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>addr</td>
<td>18</td>
<td>32</td>
</tr>
<tr>
<td>class</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>gpa</td>
<td>42</td>
<td>4</td>
</tr>
</tbody>
</table>

**Application**

```
select all from Students where gpa >= 3.0;
```
Logical Data Independence (Cont.)

- DB updates the attribute mappings.
  - Any update is therefore insulated from the application programmer!

---

### DB System Catalog

**Name: Students**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Start</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>name</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>addr</td>
<td>44</td>
<td>32</td>
</tr>
<tr>
<td>class</td>
<td>34</td>
<td>2</td>
</tr>
<tr>
<td>gpa</td>
<td>36</td>
<td>4</td>
</tr>
<tr>
<td>nick</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>meal</td>
<td>40</td>
<td>4</td>
</tr>
</tbody>
</table>

---

**Logical Level**

```sql
select all from Students
where gpa >= 3.0;
```

**(No updates here in Application Level!!)**
### Physical Data Independence

- Changes to the physical level don't affect logical view of data (relations).
  - e.g., File formats, splitting up files, ordering the tuples in file, adding data structures for faster retrieval, etc.

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>class</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>David</td>
<td>53 N. 19th St</td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td>34</td>
<td>Alan</td>
<td>12 Orange Rd.</td>
<td>JR</td>
<td>4.0</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

Physical data independence through the adherence to set theory
Physical Data Independence (Cont.)

- Currently:

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>class</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>David</td>
<td>53 N. 19th St.</td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td>34</td>
<td>Alan</td>
<td>12 Orange Rd.</td>
<td>JR</td>
<td>4.0</td>
</tr>
<tr>
<td>13</td>
<td>Julia</td>
<td>219 W St.</td>
<td>SR</td>
<td>3.2</td>
</tr>
</tbody>
</table>

DBMS extracts data from 1 large file

**Students.txt**

12, David, 53 N. 19th St, SO, 3.0
13, Julia, 219 W St, SR, 3.2
34, Alan, 12 Orange Rd, JR, 4.0
...

Students.txt
Split up files by rows

- Sets impose no ordering and retains no duplicate tuples

Students

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>class</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>David</td>
<td>53 N. 19th St</td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td>34</td>
<td>Alan</td>
<td>12 Orange Rd.</td>
<td>JR</td>
<td>4.0</td>
</tr>
<tr>
<td>13</td>
<td>Julia</td>
<td>219 W St</td>
<td>SR</td>
<td>3.2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

DBMS easily reconstructs data using: Students1 U Students2

(No updates here in Logical Level!)
Could also **split** the files by **columns**

---

### Logical Level

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>class</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>David</td>
<td>53 N. 19th St</td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td>34</td>
<td>Alan</td>
<td>12 Orange Rd.</td>
<td>JR</td>
<td>4.0</td>
</tr>
<tr>
<td>13</td>
<td>Julia</td>
<td>219 W St</td>
<td>SR</td>
<td>3.2</td>
</tr>
</tbody>
</table>

(No updates here in Logical Level!)

---

### Physical Level

- 12, David, SO, 3.0
- 13, Julia, SR, 3.2
- 34, Alan, JR, 4.0

---

DBMS reconstructs data using:

- Students1
- Students2

Called a "join" Operation (modification of cartesian product)
For true data abstraction:

- Users tell DB what they want, not how to process it
  - Departure from where you wrote a function for each query

**IBM eventually adopted Codd's relational model in IBM System-R.**

- Language support?
  - **QUEL** (by Stonebraker) for IngresDB
  - **Alpha** (by Codd)
    - Based on relational algebra and set theory
  - **SEQUEL** (by Chamberlin and Boyce, also @ IBM)
    - Used to stand for *Structured English QUEry Language*
    - Today: Just called SQL (Structured Query Language)
DBMS Architecture

- **Query Processor**
  - DDL Compiler
  - DML Compiler
  - Query Optimizer
  - Query Executor

- **Storage Manager**
  - Buffer Manager
  - File Manager
  - Transaction Manager

- **Disk**
  - Data Dictionary
  - Indices
  - Data Files

**User Groups**
- Database Admins (DBAs)
- Casual Users

**Statement Types**
- DDL Statements
- DML Statements
Outline

- Course Syllabus
- What Is a Database?
- Life without Database Management Systems
- Features of a Good Data Management System (DBMS)
- Intro to the Relational Model
- Conclusion
Database Systems in Use

- Relational database management systems (RDBMS)
  - IBM establishes System-R: the first RDBMS (1974)
  - Larry Ellison founds Oracle (1977)
    - Inspired by Codd's paper and learnings of System-R prototype
  - SQL is standardized in 1987

- RDBMS now ubiquitous
Also in Use Today: NoSQL Databases

- Not Only SQL (NoSQL)
  - Object Model (1980's)
    - We know this one. Schema defined as classes. Records are instances
  - Semi-Structured Data Model: XML and JSON (early 2000's - now)
    - Flat files making a come back as well (big push by Big Data)
  - Distributed Key-Value stores (2010s - now)

- Relation model is the focus of this course (next lecture)
Reminders:
- DB team formation emails due tonight!
- CS/Math tutoring applications due tonight!
- CS 295 Problem Seminar (zero-unit pass/fail)

Last time…
- DB is a collection of interrelated data
- Challenges of using flat-files and APIs for data management