DATABASES

INTRODUCTION TO DATA SCIENCE
TIM KRASKA
# DATA PROCESSING PIPELINES

<table>
<thead>
<tr>
<th>Colin Mallows:</th>
<th>Peter Huber:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify data to collect and its relevance to your problem</td>
<td>1. Inspection</td>
</tr>
<tr>
<td>2. Statistical specification of the problem</td>
<td>2. Error checking</td>
</tr>
<tr>
<td>5. Interpret results for non-statisticians</td>
<td>5. Modeling and model fitting</td>
</tr>
<tr>
<td>7. What-if analyses</td>
<td>7. What-if analyses</td>
</tr>
<tr>
<td>8. Interpretation</td>
<td>8. Interpretation</td>
</tr>
<tr>
<td>9. Presentation of conclusions</td>
<td>9. Presentation of conclusions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ben Fry:</th>
<th>Our Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Acquire</td>
<td>1. Preparing to run a model</td>
</tr>
<tr>
<td>2. Parse</td>
<td>2. Running the model</td>
</tr>
<tr>
<td>3. Filter</td>
<td>3. Communicating the results</td>
</tr>
<tr>
<td>4. Mine</td>
<td></td>
</tr>
<tr>
<td>5. Represent</td>
<td></td>
</tr>
<tr>
<td>6. Refine</td>
<td></td>
</tr>
<tr>
<td>7. Interact</td>
<td></td>
</tr>
</tbody>
</table>
How well do you know databases

A. What are they?

B. I used a relational database in the past, but don’t really know how they work.

C. I know SQL and tables

D. I know SQL, ER diagrams, and the relational algebra

E. I know normalization (e.g., 4th normal form) and, star and snowflake schemas
WHY DATABASES

Why not store everything in flat files?
WHY DATABASES

Why not store everything in flat files?

- Scalability → 100’s of nodes
WHY DATABASES

Why not store everything in flat files?
- Scalability → 100’s of nodes
- Data redundancy and inconsistency

<table>
<thead>
<tr>
<th>Name</th>
<th>Course</th>
<th>E-Mail</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>CS112</td>
<td><a href="mailto:jd@cs.brown.edu">jd@cs.brown.edu</a></td>
<td>B</td>
</tr>
<tr>
<td>C. Binnig</td>
<td>CS560</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>John Doe</td>
<td>CS560</td>
<td><a href="mailto:John_doe@brown.edu">John_doe@brown.edu</a></td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Course</th>
<th>E-Mail</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Doe</td>
<td>CS112</td>
<td></td>
<td><a href="mailto:jd@cs.brown.edu">jd@cs.brown.edu</a></td>
<td>B</td>
</tr>
<tr>
<td>Mike</td>
<td>Stonebaker</td>
<td>CS560</td>
<td><a href="mailto:stonebraker@uni.edu">stonebraker@uni.edu</a></td>
<td>A</td>
</tr>
<tr>
<td>Carsten</td>
<td>Binnig</td>
<td></td>
<td><a href="mailto:Carsten_binnig@brown.edu">Carsten_binnig@brown.edu</a></td>
<td>B</td>
</tr>
</tbody>
</table>

Why is this a problem?
WHY DATABASES

Why not store everything in flat files?

- Scalability → 100’s of nodes
- Data redundancy and inconsistency

<table>
<thead>
<tr>
<th>Name</th>
<th>Course</th>
<th>E-Mail</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>CS112</td>
<td><a href="mailto:jd@cs.brown.edu">jd@cs.brown.edu</a></td>
<td>B</td>
</tr>
<tr>
<td>C. Binnig</td>
<td>CS560</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>John Doe</td>
<td>CS560</td>
<td><a href="mailto:John_doe@brown.edu">John_doe@brown.edu</a></td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Courses</th>
<th>E-Mail</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Doe</td>
<td>CS112</td>
<td><a href="mailto:jd@cs.brown.edu">jd@cs.brown.edu</a></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>Mike Stonebaker</td>
<td>CS560</td>
<td><a href="mailto:stonebraker@uni.edu">stonebraker@uni.edu</a></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Carsten Binnig</td>
<td>CS112</td>
<td><a href="mailto:Carsten_binnig@brown.edu">Carsten_binnig@brown.edu</a></td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

Why is this a problem?

- Wasted space (?)
- Potential inconsistencies
  (e.g., multiple formats, John Smith vs Smith J.)
WHY DATABASES

Why not store everything in flat files?

- Scalability $\rightarrow$ 100’s of nodes
- Data redundancy and inconsistency
- Data retrieval
  - Find the student who took CS18
  - Find the student who took CS18 and has a GPA > 3.5
WHY DATABASES

Why not store everything in flat files?

- Scalability $\rightarrow$ 100’s of nodes
- Data redundancy and inconsistency
- Data retrieval
- Data-Independence
DATABASE OVERVIEW

<table>
<thead>
<tr>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Customer</strong></td>
</tr>
<tr>
<td><strong>Order</strong></td>
</tr>
<tr>
<td><strong>OrderLine</strong></td>
</tr>
</tbody>
</table>
DATABASE OVERVIEW

- View 1: Top 10 Customers
- View N: Highest Order

Logical Data Independence

- Schema:
  - Customer
  - Order
  - OrderLine

Data Models & Query Languages (Relational Model / SQL)
DATABASE OVERVIEW

<table>
<thead>
<tr>
<th>View 1</th>
<th>View N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 10 Customers</td>
<td>Highest Order</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Disk 1</th>
<th>Disk 3</th>
<th>Disk N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Order</td>
<td>OrderLine</td>
</tr>
</tbody>
</table>
```

Data Models & Query Languages (Relational Model / SQL)

Logical Data Independence

Physical Data Independence
WHY DATABASES

Why not store everything in flat files?

- Scalability → 100’s of nodes
- Data redundancy and inconsistency
- Data retrieval
- Data-Independence
- Concurrent access
- Security and privacy
COMPONENTS OF A DATABASE SYSTEM

- Logs
- Indexes
- DB
- Catalogue

External Storage

- Storage Manager
  - Logs
  - Indexes
  - DB
  - Catalogue

- DML-Compiler
  - Query Optimizer
    - Runtime
  - TA Management
    - Recovery

- DDL-Compiler
  - DBMS
    - Schema

- User
  - Application
- Ad-hoc Query

- App-Developer
  - Compiler
- Management tools

- DB-admin
DATABASES FOR DATA SCIENTIST

1. Requirement Engineering
2. Conceptual Modeling
3. Logical & Physical Modeling
4. Ask Questions

- Book of duty
- Conceptual Design (ER)
- Logical design (schema)
- Physical design (index, hints)
DATABASES FOR DATA SCIENTIST

- **Requirement Engineering**
  - Book of duty

- **Conceptual Modeling**
  - Conceptual Design (ER)

- **Logical & Physical Modeling**
  - Logical design (schema)
  - Physical design (index, hints)

- **Ask Questions**
Describe information requirements
- Objects used (e.g., student, professor, lecture)
- Domains of attributes of objects
- Identifiers, references / relationships

Describe processes
- E.g., examination, degree, register course

Describe processing requirements
- Cardinalities: how many students?
- Distributions: skew of lecture attendance
- Workload: how often a process is carried out
- Priorities and service level agreements
DATABASES FOR DATA SCIENTIST

Requirement Engineering → Conceptual Modeling → Logical & Physical Modeling → Ask Questions

- Book of duty
- Conceptual Design (ER)
- Logical design (schema)
- Physical design (index, hints)
ENTITY/RELATIONSHIP (ER) MODEL

Entity: Student

Relationship: attends

Attribute: Name

Key: Student-ID

Role: Attendant
ENTITY/RELATIONSHIP (ER) MODEL

Entity

Relationship

Attribute

Key

Role

Student

- Student-ID

Name

- Attendant

Course

Lecture

- Course-ID

- CP

- Title

CP

- Title

Student-attends-Lecture

Student-ID

Name

Attendant

Course

Lecture

Course-ID
WHY ER

Advantages

• ER diagrams are easy to create
• ER diagrams are easy to edit
• ER diagrams are easy to read (from the layman)
• ER diagrams express all information requirements

Other aspects

• Minimality
• Tools (e.g., Visio)
• Graphical representation

General

• Try to be concise, complete, comprehensible, and correct
• Controversy whether ER/UML is useful in practice
• No controversy that everybody needs to learn ER/UML
FUNCTIONALITIES

$R \subseteq E_1 \times E_2$

1:1

1:N

N:1

N:M
EXAMPLE: PROFESSOR <-> LECTURE

- Student
  - Student-ID
  - Name
  - attends
    - Lecture
      - Course-ID
      - Title
      - CP
SOMETIMES ALSO SHOWN AS

Student

Name

Student

Course-ID

Title

Lecture

CP

N

M

Student-ID
FUNCTIONALITIES OF N-ARY RELATIONSHIPS

$R : E_1 \times \ldots \times E_{k-1} \times E_{k+1} \times \ldots \times E_n \rightarrow E_k$
EXAMPLE SEMINAR

supervise : Professor × Student → Topic

supervise : Topic × Student → Professor
Should the *grade* be an entity or attribute?
Should test be an entity or relationship?
RULES OF THUMB

Attribute vs. Entity

• Entity if the concept has more than one relationship
• Attribute if the concept has only one 1:1 relationship

Partitioning of ER Models

• Most realistic models are larger than a page
• Partition by domains (library, research, finances, ...)

Good vs. Bad models

• Do not model redundancy or tricks to improve performance
• Less entities is better (the fewer, the better!)
• Remember the 5 C’s (clear, concise, correct, complete, compliant)
(Min, Max) - Notation
Weak Entities
Generalization (i.e., inheritance)
Modeling limitations
Enhanced ERM
...
CLICKER QUESTION

Model a music record database

• An album has a unique name and songs have unique titles
• An album contains several songs
• A playlist has a unique name and is created by one user with a unique login
• A playlist contains several songs from potential different albums
CLICKER QUESTION

A) Album
   \[\text{contains}\rightarrow\text{Playlist} \quad \text{contains}\rightarrow\text{Songs}\]
   Playlist
   \[\text{User}\rightarrow\text{login}\]
   Songs
   Title

B) Album
   \[\text{contains}\rightarrow\text{Playlist} \quad \text{contains}\rightarrow\text{Songs}\]
   Playlist
   \[\text{User}\rightarrow\text{creates}\times\text{Playlist}\]
   Songs
   Name
   Title
   User
   login

C) Album
   \[\text{contains}\rightarrow\text{Playlist}\]
   Playlist
   \[\text{User}\rightarrow\text{playList}\times\text{Playlist}\]
   Songs
   Name
   Title
   User
   login
DATABASES FOR DATA SCIENTIST

Requirement Engineering

Conceptual Modeling

Logical & Physical Modeling

Ask Questions

Book of duty

Conceptual Design (ER)

- Logical design (schema)
- Physical design (index, hints)
RELATIONAL MODEL - TERMS

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

Account =

Terms:
- **Tables** → Relations
- **Columns** → Attributes
- **Rows** → Tuples
- **Schema** (e.g.: Acct_Schema = (bname, acct_no, balance))
WHY ARE THEY CALLED RELATIONS?

Relation:

- $R \subseteq D_1 \times \ldots \times D_n$
- $D_1, D_2, \ldots, D_n$ are domains

Example: AddressBook $\subseteq$ string $\times$ string $\times$ integer
WHY ARE THEY CALLED RELATIONS?

Relation:

- \( R \subseteq D_1 \times \ldots \times D_n \)
- \( D_1, D_2, \ldots, D_n \) are domains

Example: AddressBook \( \subseteq \) string x string x integer

Tuple: \( t \in R \)

Example: \( t = ("Mickey Mouse", "Main Street", 4711) \)
WHY ARE THEY CALLED RELATIONS?

Relation:

- $R \subseteq D_1 \times \ldots \times D_n$
- $D_1, D_2, \ldots, D_n$ are domains

Example: AddressBook $\subseteq$ string x string x integer

Tuple: $t \in R$

Example: $t = ("Mickey Mouse", "Main Street", 4711)$

Schema: associates labels to domains

Example:

AddrBook: {[Name: string, Address: string, Tel#:integer]}
CSCI1270, Lecture 2

Relational database semantics are defined in terms of mathematical relations (i.e., sets)

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

Considered equivalent to...

\{ (Downtown, A-101, 500),
   (Brighton, A-201, 900),
   (Brighton, A-217, 500) \}
KEYS AND RELATIONS

Kinds of keys

• Superkeys:
  set of attributes of table for which every row has distinct set of values

• Candidate keys:
  “minimal” superkeys

• Primary keys:
  DBA-chosen candidate key (marked in schema by underlining)

<table>
<thead>
<tr>
<th>ISBN</th>
<th>Title</th>
<th>Author</th>
<th>Edition</th>
<th>Publisher</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0439708184</td>
<td>Harry Potter</td>
<td>J.K. Rowling</td>
<td>1</td>
<td>Scholastic</td>
<td>$6.70</td>
</tr>
<tr>
<td>0545663261</td>
<td>Mockingjay</td>
<td>Suzanne Collins</td>
<td>1</td>
<td>Scholastic</td>
<td>$7.39</td>
</tr>
</tbody>
</table>
KEYS AND RELATIONS

Kinds of keys

• Superkeys:
  set of attributes of table for which every row has distinct set of values

• Candidate keys:
  “minimal” superkeys

• Primary keys:
  DBA-chosen candidate key (marked in schema by underlining)

Act as Integrity Constraints
  i.e., guard against illegal/invalid instance of given schema

e.g., Branch = (bname, bcity, assets) ✗

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brighton</td>
<td>Brooklyn</td>
<td>5M</td>
</tr>
<tr>
<td>Brighton</td>
<td>Boston</td>
<td>3M</td>
</tr>
</tbody>
</table>
HOW TO TRANSLATE ERM TO RELATIONS

- **Professor**
  - Person-ID
  - Name
  - Course-ID
  - Title
  - CP

- **Student**
  - Student-ID
  - Name
  - Lecture
  - Course-ID

- **Grade**
  - tests

- **Lecture**
  - attend

- **tests**
  - 1

- **attend**
  - N

- **Professor**
  - gives

- **Lecture**
  - M

- **Student**
  - M

- **Professor**
  - M

- **Student**
  - N

- **Grade**
  - N
RULE #1: ENTITIES

Professor(Person-ID:integer, Name:string)
Student(Student-ID:integer, Name:string)
Lecture(Course-ID:string, Title:string, CP:float)
RULE #2: RELATIONSHIPS

R: \{ [A_{11}, \ldots, A_{1k_1}], A_{21}, \ldots, A_{2k_2}, \ldots, A_{n1}, \ldots, A_{nk_n}, A_1^R, \ldots, A_{k_R}^R \}
RULE #2: RELATIONSHIPS

Professor(Person-ID:integer, Name:string)
Student(Student-ID:integer, Name:string)
Lecture(Course-ID:string, Title:string, CP:float)
Gives(Person-ID:integer, Course-ID:string)
Attends(Student-ID:integer, Course-ID:string)
Tests(Student-ID:integer, Course-ID:string, Person-ID:integer, Grade:StringLength)

What about keys?
RULE #2: RELATIONSHIPS

Professor(Person-ID:integer, Name:string)
Student(Student-ID:integer, Name:string)
Lecture(Course-ID:string, Title:string, CP:float)
Gives(Person-ID:integer, Course-ID:string)
Attends(Student-ID:integer, Course-ID:string)
Tests(Student-ID:integer, Course-ID:string, Person-ID:integer, Grade:string)
### INSTANCE OF ATTENDS

<table>
<thead>
<tr>
<th>Student</th>
<th>Attends</th>
<th>Lecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student-ID</td>
<td>Student-ID</td>
<td>Course-ID</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>CS1951a</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>CS167</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>CS1951a</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>CS167</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>CS18</td>
</tr>
<tr>
<td>10</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Diagram:**

- **Student** (N) attend **Attends** (1..*M*)
- **Attends** (1..*N*) attend **Lecture** (1..*M*)

**Key:**
- Student-ID
- Course-ID
**RULE #3: MERGE RELATIONS WITH THE SAME KEY**

Professor\((\text{Person-ID:integer, Name:string})\)
Lecture\((\text{Course-ID:string, Title:string, CP:float})\)
Gives\((\text{Person-ID:integer, Course-ID:string})\)

\(\Rightarrow\)

Professor\((\text{Person-ID:integer, Name:string})\)
Lecture\((\text{Course-ID:string, Title:string, CP:float, Person-ID:integer})\)
Why didn’t we merge **Attends** and **Tests**?
SQL: CREATE TABLE

Professor(Person-ID:integer, Name:string)
Lecture(Course-ID:string, Title:string, CP:float, Person-ID:integer)
Student(Student-ID:integer, Name:string)

CREATE TABLE Student (Student-ID INT, Name VARCHAR(45));
CREATE TABLE Professor (Person-ID INT, Name VARCHAR(45));
CREATE TABLE Lecture (Course-ID INT, Title VARCHAR(45), CP REAL, Person-ID INT);
SQL: CREATE TABLE

Professor(Person-ID:integer, Name:string)
Lecture(Course-ID:string, Title:string, CP:float, Person-ID:integer)
Student(Student-ID:integer, Name:string)

CREATE TABLE Student (Student-ID INT, Name VARCHAR(45));
CREATE TABLE Professor (Person-ID INT, Name VARCHAR(45));
CREATE TABLE Lecture (Course-ID INT, Title VARCHAR(45), CP REAL, Person-ID INT);
INTEGRITY CONSTRAINTS IN CREATE TABLE

not null
primary key \( (A_1, \ldots, A_n) \)

Example: Declare \( ID \) as the primary key for instructor

```
CREATE TABLE Attends (  
    Student-ID INT,  
    Course-ID VARCHAR(6),  
    PRIMARY KEY (Student-ID, Course-ID));
```

primary key declaration on an attribute automatically ensures not null
CREATE TABLE course (  
course_id VARCHAR(8) PRIMARY KEY,  
title VARCHAR(50),  
cp NUMERIC(1,1));

Primary key declaration can be combined with attribute declaration as shown above
CREATE TABLE `Attends` (  
`Student_ID` INT NOT NULL,  
`Course-ID` VARCHAR(6) NOT NULL,  
PRIMARY KEY (`Student_ID`, `Course-ID`),  
CONSTRAINT `fk_attend_Student`  
FOREIGN KEY (`Student_ID`)  
REFERENCES `Student` (`ID`)  
ON DELETE NO ACTION  
ON UPDATE NO ACTION,  
CONSTRAINT `fk_attend_lecture`  
FOREIGN KEY (`Lecture_Course-ID`)  
REFERENCES `Lecture` (`Course-ID`)  
ON DELETE NO ACTION  
ON UPDATE NO ACTION;
INDEX

CREATE INDEX `fk_Student_has_Lecture_Lecture1_idx` ON `Attends` (`Course-ID` ASC);

CREATE INDEX `fk_Student_has_Lecture_Student_idx` ON `Attends` (`Student-ID` ASC);
PROBLEM

• You are the new Data Scientist at Evil Market
• Evil Market is tracking all customer purchases with their membership card or credit card
• They also have data about their customers (estimated income, family status, …)
• Recently, they are trying to improve their image for young mothers
• As a start they want to know the following information for mothers under 30 for 2013:
  • How much do they spend?
  • How much do they spend per state?
  • How does this compare to all customers under 30?
  • What are their favorite products?
  • How much do they spend per year?

Your first project: Design the schema for Evil Market!
OTHER ANNOUNCEMENTS

Want to get involved in research?

We are offering several independent studies and summer research internship. 

*Sign-up available on:* [http://database.cs.brown.edu/](http://database.cs.brown.edu/)

*or directly:* [http://tinyurl.com/zxznf92](http://tinyurl.com/zxznf92)

Possible Topics:

- Infiniband
- Tupleware
- Interactive Data Exploration
Do you want to drink from the fire hose??? Then take my CS227 class