Please use Piazza:
https://piazza.com/class/m06qshnnxi85if

http://dsg.csail.mit.edu/6.5830/

6.5830/6.5831
Introduction to Databases

6.5830 Lecture 1- 9/4/2024

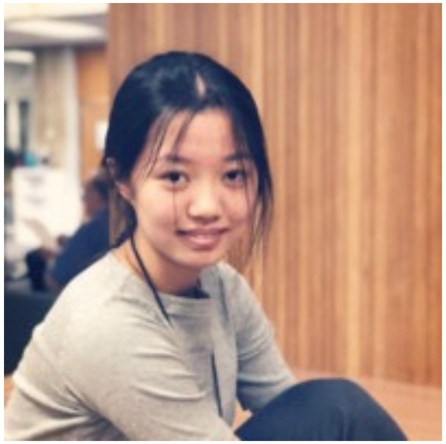
Mike Cafarella, Tim Kraska

michjc@csail.mit.edu, kraska@csail.mit.edu

Sam Madden

Tim Kraska

Mark Jabbour



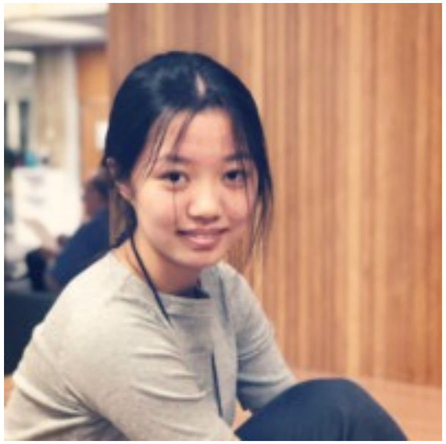
Sylvia Ziyu Zhang

The original cast for 2024

Michael Cafarella

Tim Kraska

Mark Jabbour

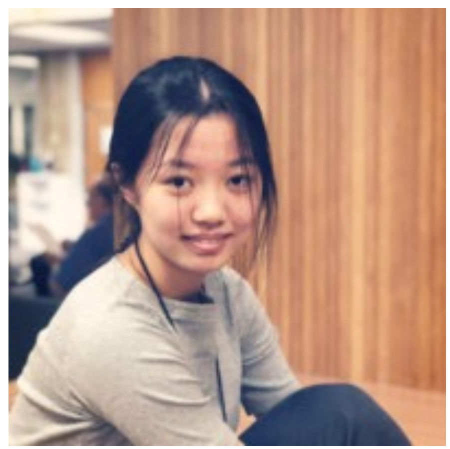


Sylvia Ziyu Zhang

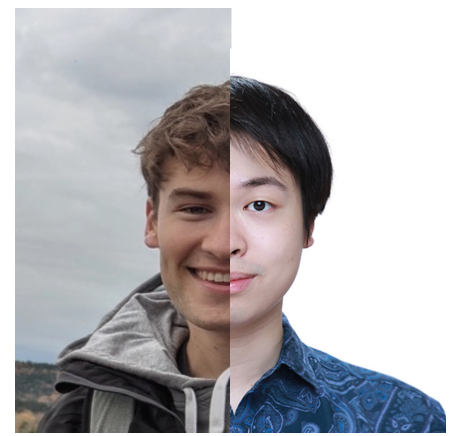
Michael Cafarella

Tim Kraska

Mark Jabbour



Sylvia Ziyu Zhang



Ferdi Kossmann

Ziniu Wu

Administrivia

http://dsg.csail.mit.edu/6.5830, Email: 6.5830-staff@mit.edu

Ask questions on Piazza! Use sign up link on website.

<u>Lecturers:</u>

Mike Cafarella (michjc@mit.edu)

Tim Kraska (kraska@mit.edu)

<u>TAs:</u>

Mark Jabbour (mjabbour@mit.edu)

Ferdi Kossmann (kossmann@mit.edu)

Ziniu Wu (ziniuw@mit.edu)

Sylvia Zhang (sylziyuz@mit.edu)

Office hours: see website

Note generative AI policy; 5 Late Days

Textbooks

- Readings in Database Systems
  - http://www.redbook.io
- Rest of readings will be drawn from literature (research papers and web pages)

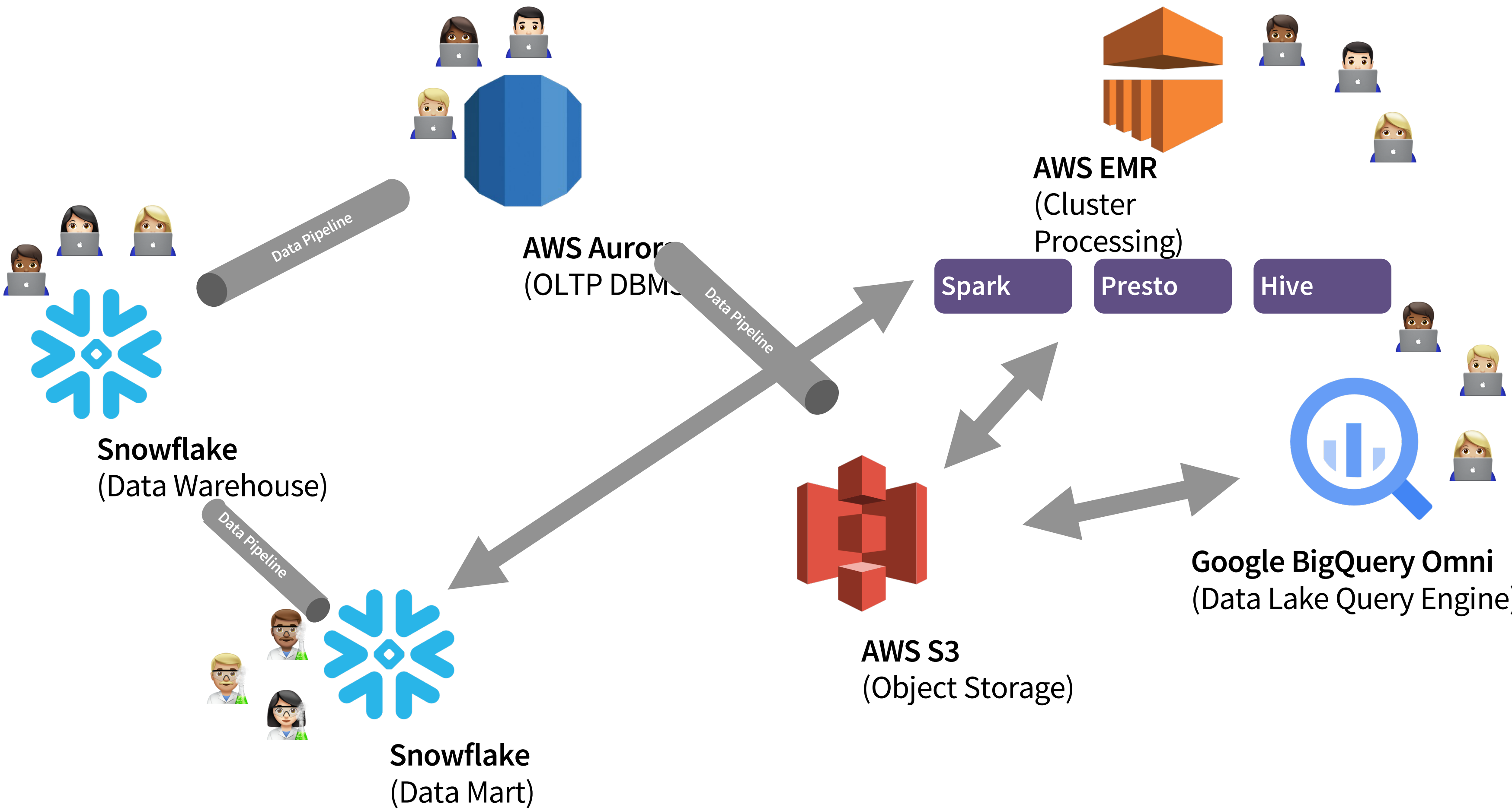
What is a Database?

- Structured Data Collection
  - Records
  - Relationships

- This class: Database Management Systems (DBMSs)

  Software systems for storing and querying databases

The modern cloud data mesh



Commercial Systems (even more if you include open-source

Source: mattturck.com

6.5830/1 Concepts

- Data modeling / layout

Quiz 1

- Declarative querying
  - Query processing
  - Algorithms for accessing and manipulating data

- Consistency / Transactions ("ACID")
- "Big Data" – scaling to massive volumes, many machines

Quiz 2

Two class flavors

6.5831 is an undergraduate class designed to satisfy the **AUS requirement** in the EECS curriculum (instead of 6.1800). The class does **not** fulfill the CI-M requirement.

6.5830 is a Grad-H class. It counts as an engineering concentration (EC) subject in Systems. **For Area II Ph.D. students in EECS, it satisfies the Systems TQE requirement and the AUS requirement.** 6.5830 requires the completion of a final open-ended research project

6.5830/1 Assignments

- 4 Labs: GoDB

- 3 Problem Sets
  - SQL
  - Quiz Prep 1
  - Quiz Prep 2
- 2 Quizzes

- 6.5830: **Final Project** – open ended research project of your choice

- 6.5831: **Final Project**: Extend GoDB **OR** open-ended research

6.5830/1 Grading

6.5830

- Assignment (Problem Sets and Labs): 35% total
  - PSET 1: 3.33%
  - PSET 2: 5.00%
  - PSET 3: 5.00%
  - Lab 0: 1.66%
  - Lab 1: 6.66%
  - Lab 2: 6.66%
  - Lab 3: 6.66%
- Quizzes: 15% each
- Course Project: 30%
- Class Participation: 5% (clicker, piazza, and general participation during class)

6.5831

- Assignments (Problem Sets and Labs): 65% total
  - PSET 1: 5%
  - PSET 2: 7.5%
  - PSET 3: 7.5%
  - Lab 0: 2.5%
  - Lab 1: 10%
  - Lab 2: 10%
  - Lab 3: 10%
  - Lab 4: 12.5%
- Quizzes: 15% each
- Class Participation: 5% (clicker, piazza, and general participation during class)

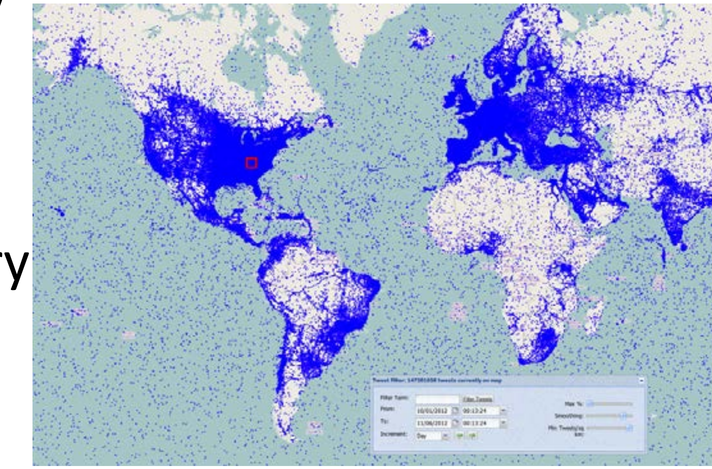
The System you will be working on:

Open-ended Project

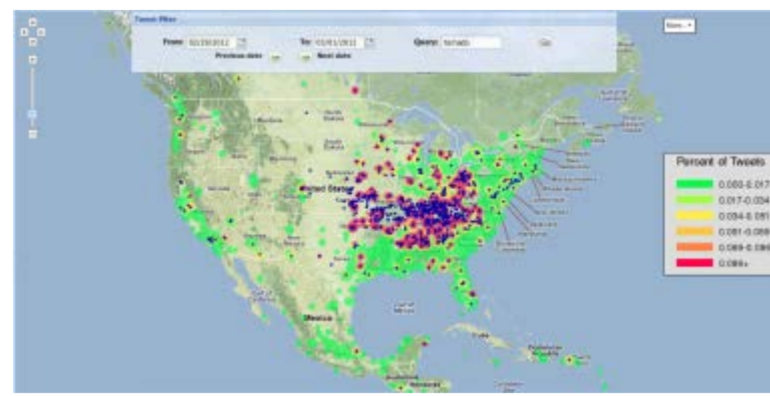
- We will release a list of potential project ideas, but you can also BYO.
- Projects needs to be related to data-centric systems
- Last projects included GPU-accelerated DBs, Database benchmarks, Learned Index Structures, Text2SQL, …
- Final deliverable: 5-min recorded presentation, in-class Q&A, written report

MapD: GPU Accelerated SQL Database

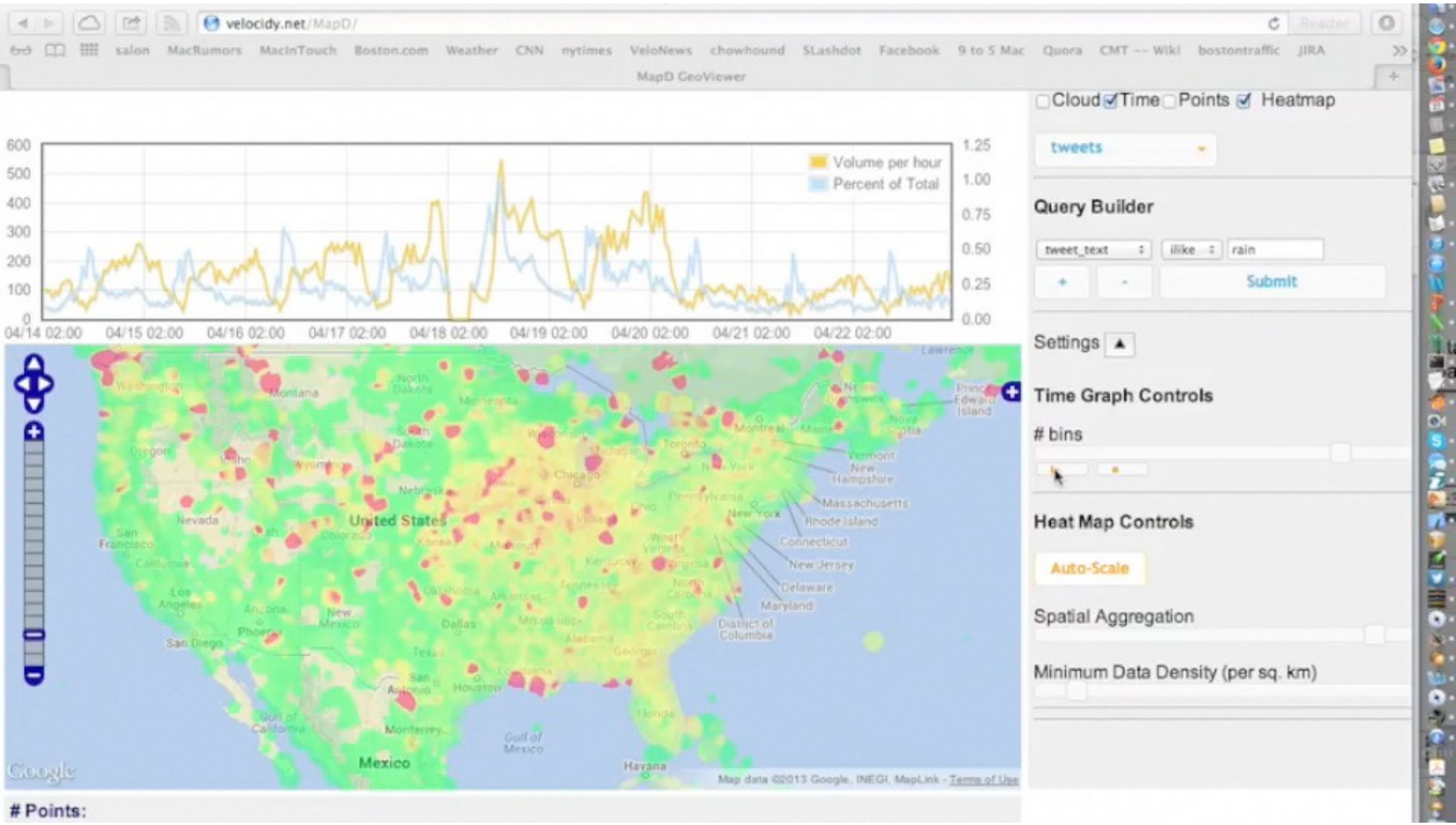
- *Key insight:* GPUs have enough memory that a cluster of them can store substantial amounts of data
- Not an accelerator, but a full blown query processor!
- Massive parallelism enables interactive browsing interfaces
  - 4x GPUs can provide > 1 TB/sec of bandwidth
  - 12 Tflops compute
  - Order of magnitude speedups over CPUs, when data is on GPU
- "Shared nothing" arrangement



147,201,658 tweets from Oct 1, 2012 to Nov 6, 2012



Relative intensity of "tornado" on Twitter (with point overlay) from Febuary 29, 2012 to March 1, 2012


velocidy.net/MapD/
salon MacRumors MacInTouch Boston.com Weather CNN nytimes VeloNews chowhound SLashdot Facebook 9 to 5 Mac Quora CMT -- Wiki bostontraffic JIRA
MapD GeoViewer
Cloud Time Points Heatmap
tweets
Query Builder
tweet_text
ilike
rain
+
-
Submit
Settings
Time Graph Controls
bins
Heat Map Controls
Auto-Scale
Spatial Aggregation
Minimum Data Density (per sq. km)
Volume per hour
Percent of Total
600
500
400
300
200
100
0
1.25
1.00
0.75
0.50
0.25
0.00
04/14 02:00
04/15 02:00
04/16 02:00
04/17 02:00
04/18 02:00
04/19 02:00
04/20 02:00
04/21 02:00
04/22 02:00
United States
Mexico
Map data ©2013 Google, INEGI, MapLink - Terms of Use
Points:

HEAVY.AI

Request Demo

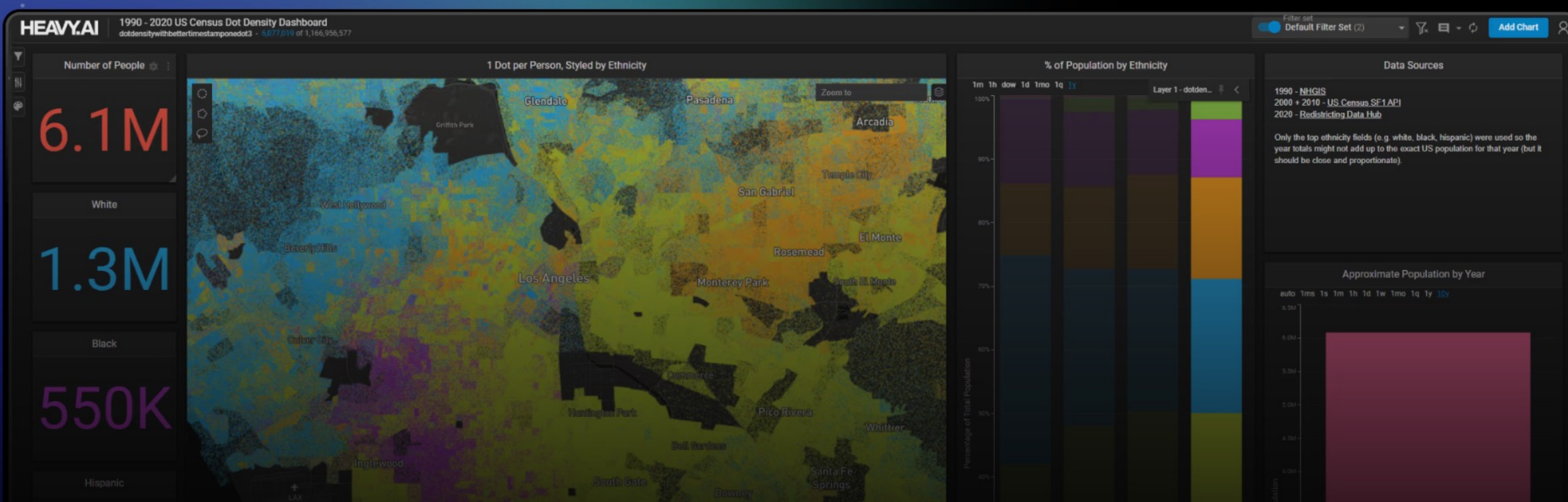
Get HEAVY.AI Free

A Revolutionary GPU-Accelerated **Analytics Platform**

Instant analytics on billions of records, including geospatial and time series data, for a complete view of what, when and where.

Get HEAVY.AI Free

Learn More →



Monday	Tuesday	Wednesday
Sep 2 *Labor Day*	Sep 3 *Registration Day*	Sep 4 *First Day of Classes* **Lec 1:** Introduction to Databases / Relational Model / SQL Part 1 **Assigned: Lab 0**
Sep 9 **Lec 2:** SQL Part 2 **Reading Assignment** **Assigned: PS 1**	Sep 10	Sep 11 **Lec 3:** Schema Design **Reading Assignment** **Assigned: Lab 1** **Due: Lab 0**
Sep 16 **Lec 4:** Intro to Database Internals **Reading Assignment**	Sep 17	Sep 18 **Lec 5:** Database Operators and Query Processing **Reading Assignment** **Due: PS 1**
Sep 23 **Lec 6:** Indexing and Access Methods **Reading Assignment** **Assigned: Lab 2**	Sep 24	Sep 25 **Lec 7:** Join Algorithms **Reading Assignment** **Due: Lab 1** **Due: Project teams (if doing final project)** **Assigned: PS 2**

Today

- Why database systems?
- User's perspective:
  - Modeling data
  - Querying data
- Data Models

Zoo Website Features

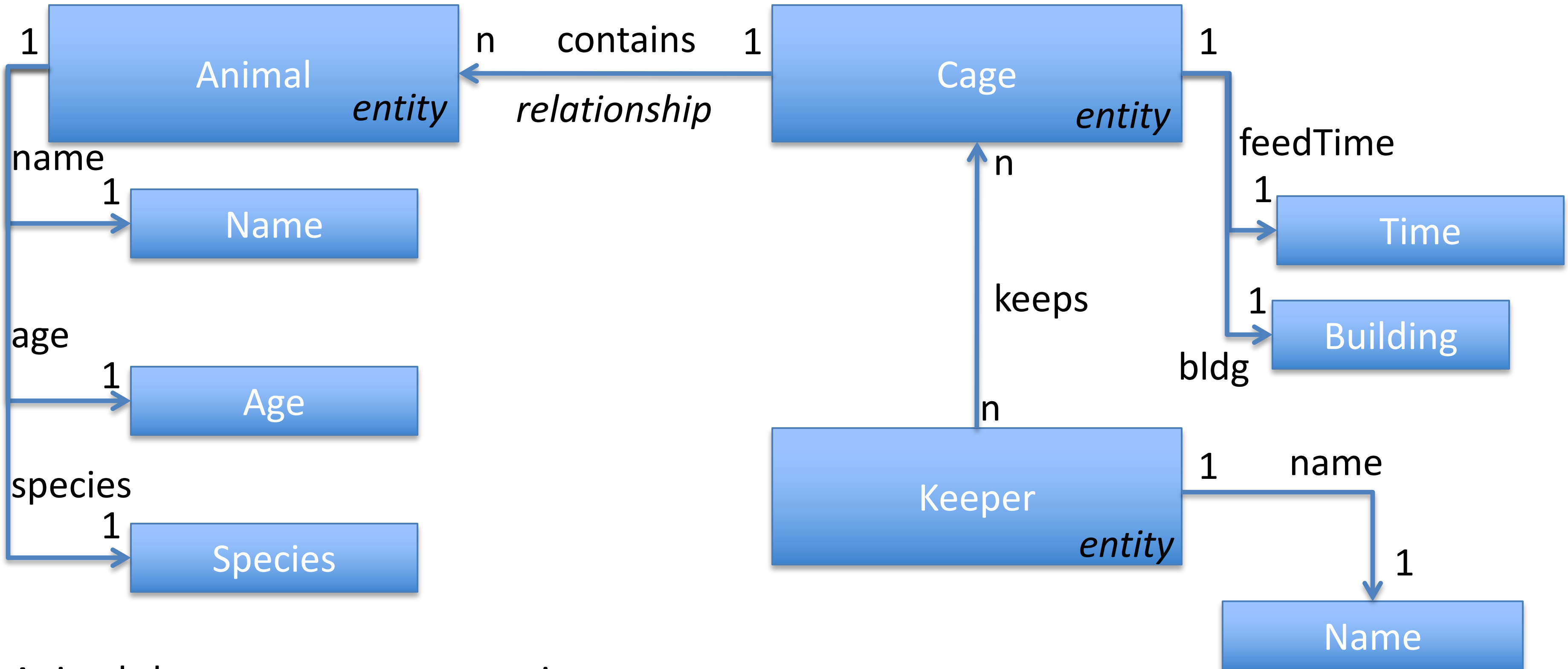
- Admin interface
  - Edit
  - Add an animal
- Public
  - Pictures & Maps
- Zookeeper
  - Feed times

San Diego Zoo

- 1K animals, 5K URLs, 10 admins, 200 keepers

Zoo Data Model

Entity Relationship Diagram



Animals have names, ages, species
Keepers have names
Cages have cleaning times, buildings
Animals are in 1 cage; cages have multiple animals
Keepers keep multiple cages, cages kept by multiple keepers

Our Zoo

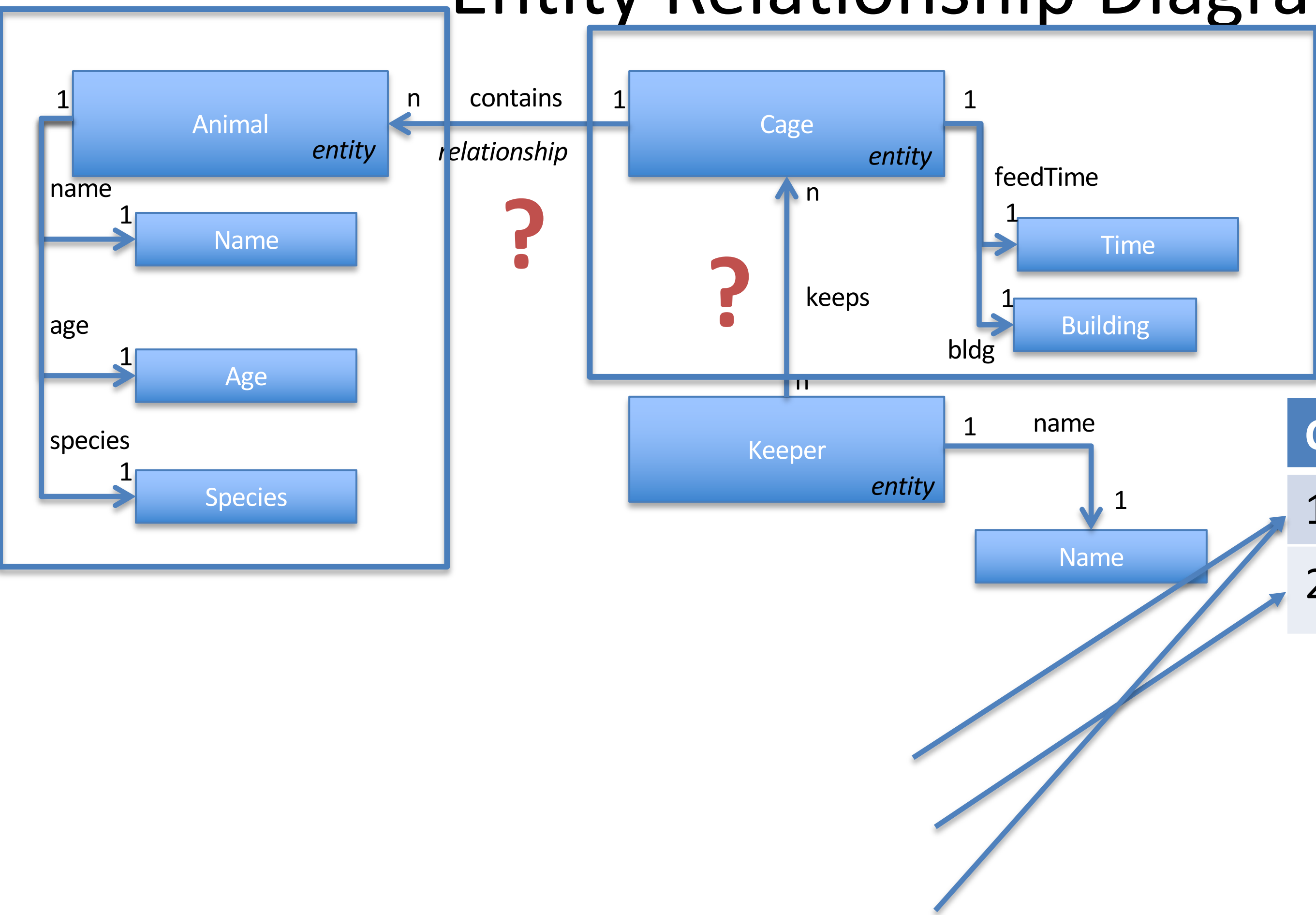
Mike the Moose

Tim the Giraffe

Sally the Student

Zoo Data Model

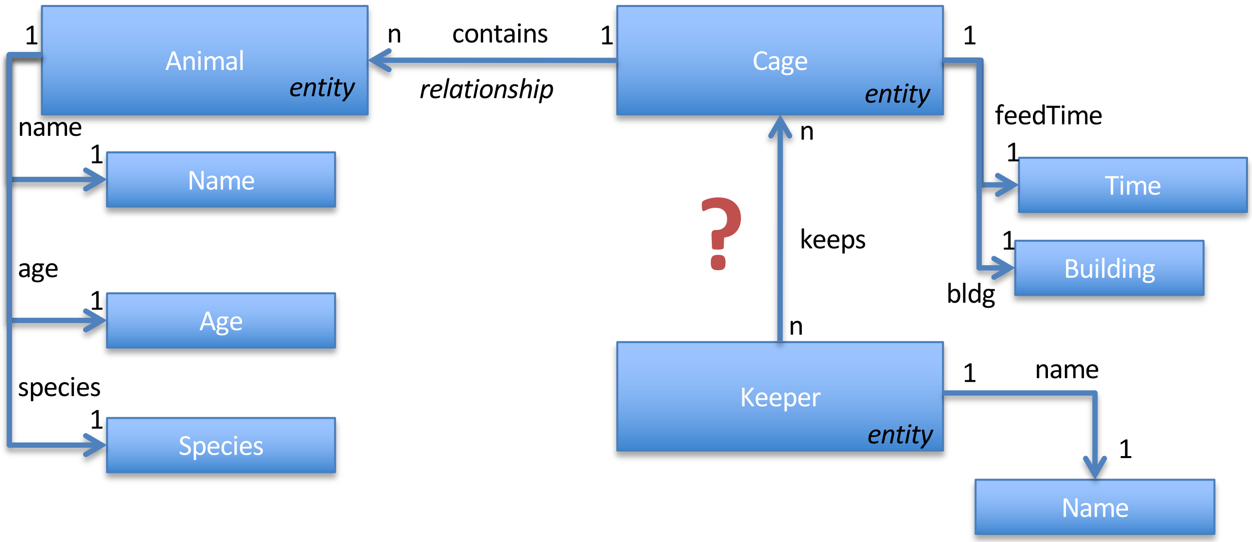
Entity Relationship Diagram



Cageid	Feedtime	Bldg
1	1:30	1
2	2:30	2

Zoo Data Model

Entity Relationship Diagram



Cageid	Feedtime	Bldg
1	1:30	1
2	2:30	2

keeperid	name
1	jenny
2	joe

keeperid	cageid
1	1
1	2
2	1

Study Break #1

- Questions
  - Are there other ways to represent this zoo data than a collection of tables?
  - What are tradeoffs in different representations?

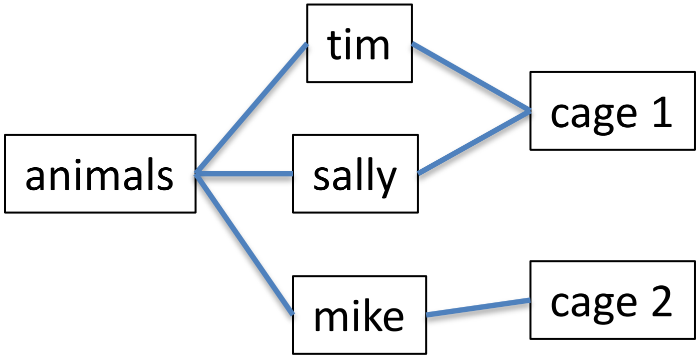
Alternatives to Relations

Hierarchy

cage 1
- tim
  - giraffe
  - 13 yrs
- sally
  - student
  - 1 yr

cage 2
- mike
  - moose
  - 3 yrs

Graph



Multiple Tabular Representations Are Possible

name	age	species	cageno	feedtime	bldg
tim	13	giraffe	1	1:30	1
mike	3	moose	2	2:30	2
sally	1	student	1	1:30	1

Is this a good representation? Why or why not?

Not "Normalized" – repeats data. More in later lectures!

SQL – Structured Query Language

```
SELECT  field1, …, fieldM
FROM table1, …
WHERE condition1, …

INSERT INTO table VALUES (field1, …)

UPDATE table SET field1 = X, …
WHERE condition1,…
```

Names of Giraffes

- Imperative

```
for each row r in animals
   if r.species = 'giraffe'
      output r.name
```

- Declarative

```
SELECT r.name FROM animals
WHERE r.species = 'giraffe'
```

Cages in Building 32

NESTED LOOPS

- Imperative

```
for each row a in animals
   for each row c in cages
      if a.cageno = c.no and c.bldg = 32
         output a
```

- Declarative

JOIN

```
SELECT a.name FROM animals AS a, cages AS c
WHERE a.cageno = c.no AND c.bldg = 32
```

Average Age of Bears

- Declarative

```
SELECT AVG(age) FROM animals
WHERE species = ‘bear’
```

Complex Queries

Find pairs of animals of the same species and different genders older than 1 year:

```
SELECT a1.name,a2.name
FROM animals as a1, animals as a2
WHERE a1.gender = M and a2.gender = F
AND a1.species = a2.species
AND a1.age > 1 and a2.age > 1
```

"self join"

Find cages with salamanders fed later than the average feedtime of any cage:

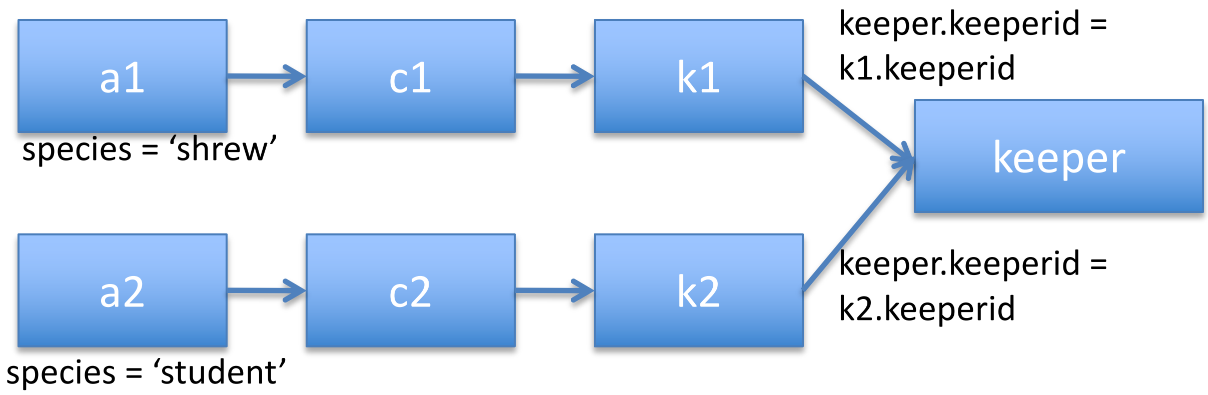
```
SELECT cages.cageid FROM cages, animals
WHERE animals.species = 'salamander'
AND animals.cageid = cages.cageid
AND cages.feedtime >
    (SELECT AVG(feedtime) FROM cages )
```

"nested queries"

Complex Queries 2

Find keepers who keep both students and salamanders:

```
SELECT keeper.name
FROM keeper, cages as c1, cages as c2,
        keeps as k1, keeps as k2, animals as a1, animals as a2
WHERE c1.cageid = k1.cageid AND keeper.keeperid = k1.keeperid
AND c2.cageid = k2.cageid AND keeper.keeperid = k2.keeperid
AND a1.species = ’student' AND a2.species = ’salamander'
AND c1.cageid = a1.cageid AND c2.cageid = a2.cageid
```

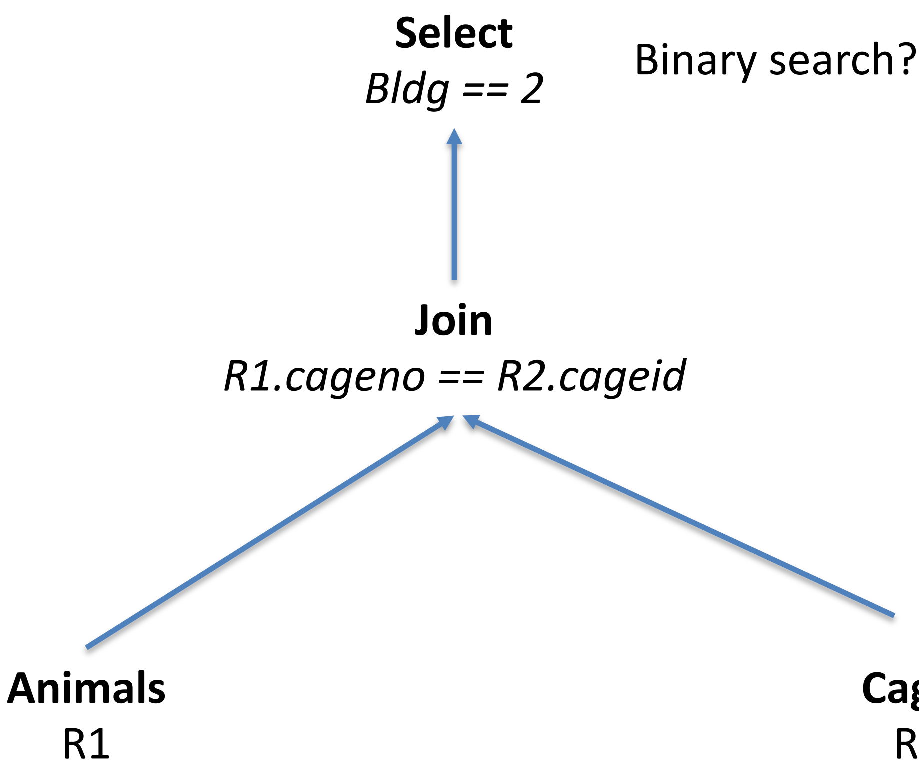


Declarative Queries: **What,** not **How**

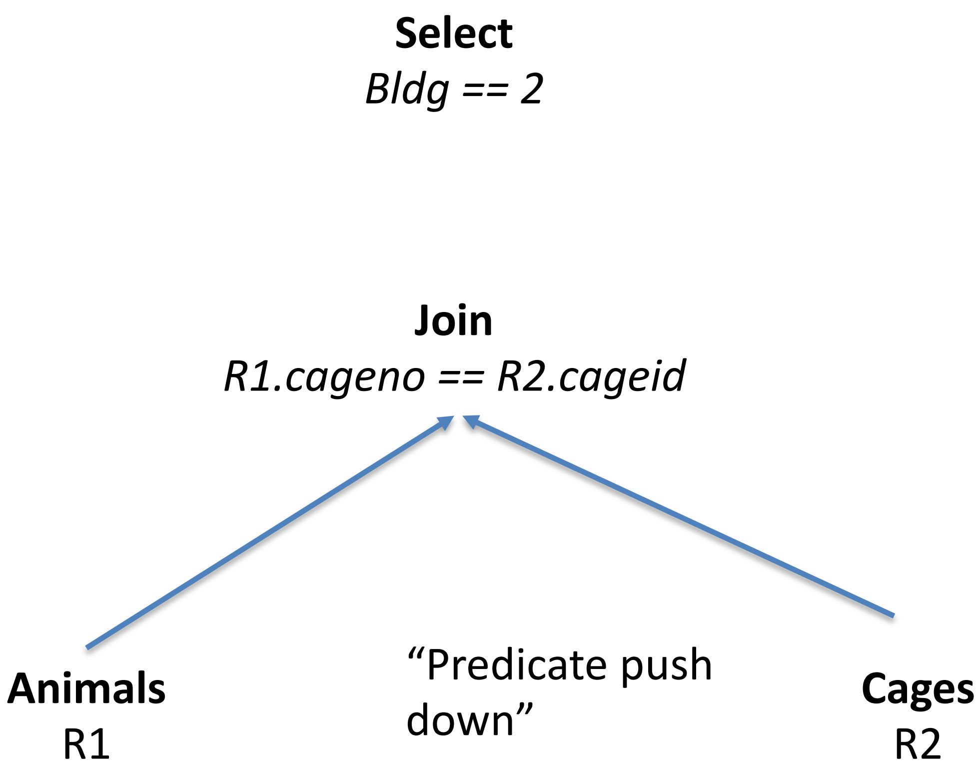
- Many possible proc
  query

- Besides looping thr
  could we do?
  - Sort animals on type
    - + good for "bears" query
    - - Inserts are slower
  - Store animals table in a hash table or tree ("index")

SQL →Procedural Plan → Optimized Plan → Compiled Program



SQL → Procedural Plan → <u>Optimized Plan</u> → Compiled Program



SQL programmer just thinks in terms of table operations, not the order or implementation!

Summary: Database Systems

- Relational Model + Schema Design
- Declarative Queries
- Query Optimization
- Efficient access and updates to data
  - Recoverability
  - Consistency

Relational Model

“Those who cannot remember the past are doomed to repeat it”

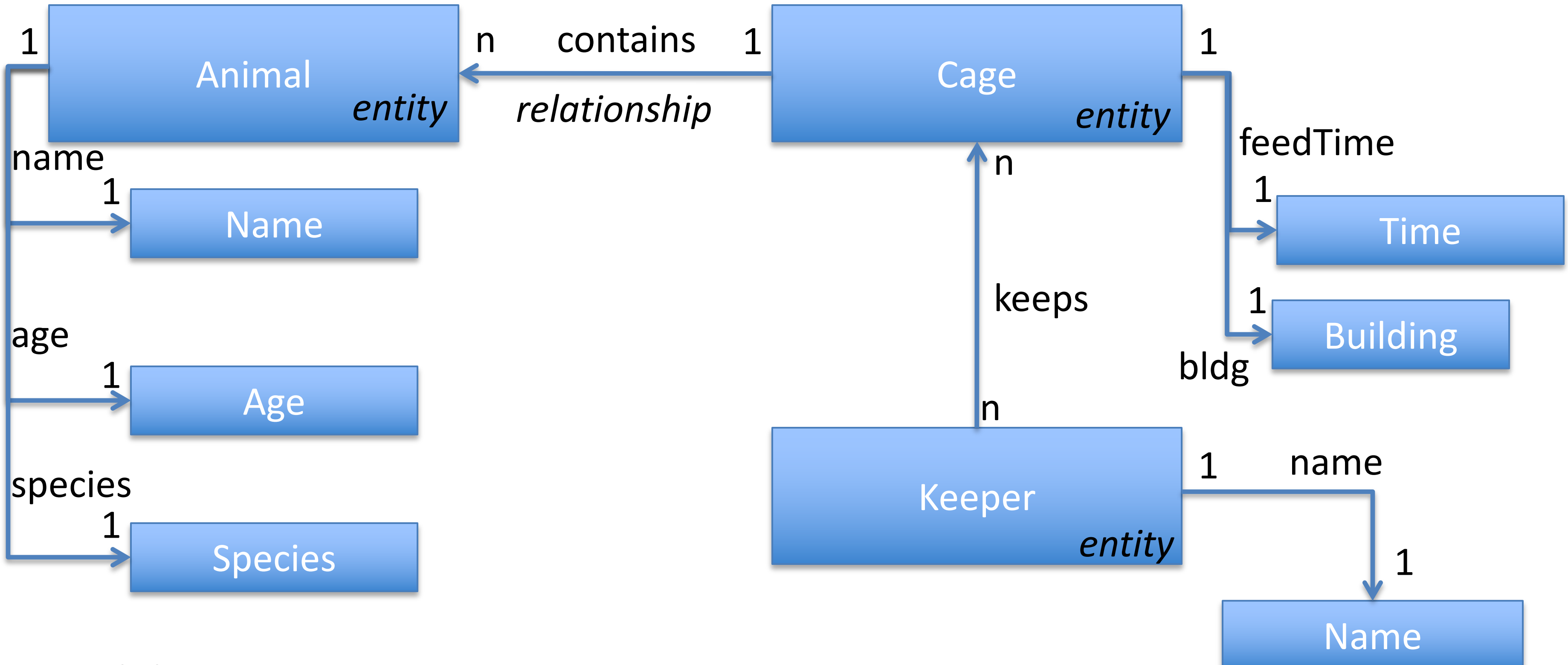
A Short History Lesson

- **Different Data Models**
  - Hierarchical (IMS/DL1) – 19*60’s*
  - Network (CODASYL) – 1970’s
  - Relational – 1970’s and beyond

- **Key ideas**
  - Data redundancy (and how to avoid it)
  - Physical and logical data independence
  - Relational algebra and axioms

Recap: Zoo Data Model

Entity Relationship Diagram



Animals have names, ages, species

Keepers have names

Cages have cleaning times, buildings

Animals are in 1 cage; cages have multiple animals

Keepers keep multiple cages, cages kept by multiple keepers

Zoo Tables (aka Relations)

Animals

id	name	age	species	cageno
1	Mike	3	Moose	1
2	Tim	12	Giraffe	1
3	Sally	1	Student	2

"Schema": Field names & types

Rows, records, or tuples

Cages

no	feedtime	building
1	12:30	1
2	1:30	2

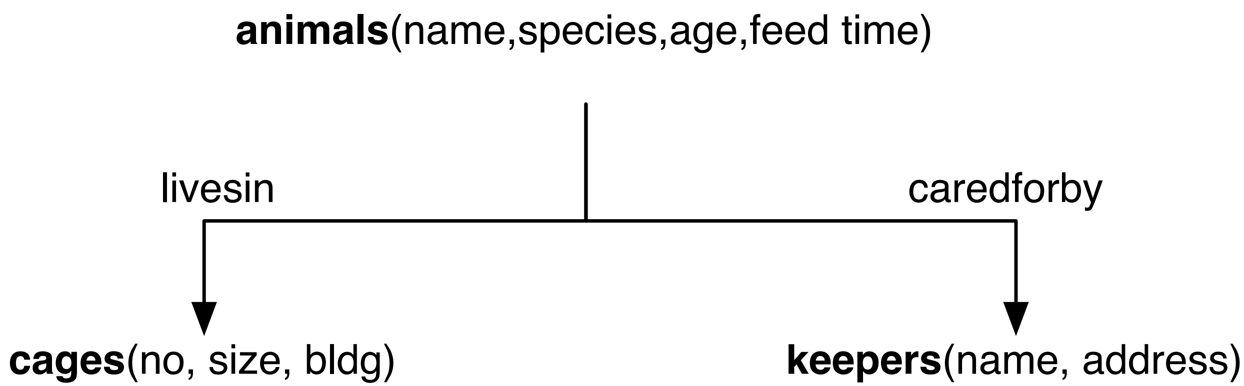
Keepers

id	name
1	Jane
2	Joe

Keeps

kid	cageno
1	1
1	2
2	1

Modified Zoo Data Model



Slightly different than last time:

- Each animal in 1 cage, multiple animals share a cage
- Each animal cared for by 1 keeper, keepers care for multiple animals

IMS (Hierarchical Model)

- Data organized as *segments*
  - Collection of records, each with same *segment type*
  - Arranged in a tree of segment types, e.g.:

Keepers	Keepers
Animals	Cages
Cages	Animals

- Segments have different physical representations
  - Unordered
  - Indexed
    - Sorted
    - Hashed

Example Hierarchy

- Jane (keeper) (HSK 1)
  - Mike, moose, … (2)
    - 1, 100sq ft, … (3)
  - Tim, giraffe, … (4)
    - 2, 1000sq ft, … (5)
  - Sally, student, … (6)
    - 1, 100sq ft, … (7)
- Joe (keeper) (8)

Repeated information!

IMS Physical Represenation

Keepers segment

A1 Segment	**A2 Segment**	**A3 Segment**
C1 Segment	**C2 Segment**	**C3 Segment**

Segment Structure

- Each segment has a particular physical representation
  - Chosen by database administrator
  - E.g., ordered, hashed, unordered…
- Choice of segment structure affects which operations can be applied on it

IMS / DL/1 Operations

- **GetUnique** (seg type, pred)
  - Get first record satisfying pred
  - Only supported by hash / sorted segments
- **GetNext** (seg type, pred)
  - Get first or next key in hierarchical order
  - Starts from last GetNext/GetUnique call
- **GetNextParent** (seg type, pred)
  - Same as GetNext, but will not move up hierarchy to next parent
- **Delete, Insert**

Example PL/1 Program #1

Find the cages that Jane keeps

```
GetUnique(Keepers, name = "Jane")
Until done:
    cageid = GetNextParent (cages).no
    print cageid
```

Jane (keeper) (HSK 1)
    Mike, moose, … (2)
        1, 100sq ft, … (3)
    Tim, giraffe, … (4)
        2, 1000sq ft, … (5)
    Sally, student, … (6)
        1, 100sq ft, … (7)
Joe (keeper) (8)

This iterates through data underneath Jane

Implicitly, now navigating from the Jane record in keepers

Example PL/1 Program #2

Find the keepers that keep cage 6

```
keep = GetUnique(keepers)

Until done:
    cage = GetNextParent(cages, id = 6)
    if (cage is not null):
            print keep
    keep = GetNext(keepers)
```

What's Bad About IMS/PL1?

- Duplication of data w/ non-hierarchical data
- Painful low level programming interface – have to program the search algorithm
- Limited **physical data independence**
  - Change root from indexed to hash --- programs that do GetNext on the root segment will fail
  - Change root from keepers to animals? Also fails.
  - Cannot do inserts into sequential root structure
- Limited **logical data independence**
  - Schemas change, do programs have to?

Logical Data Independence

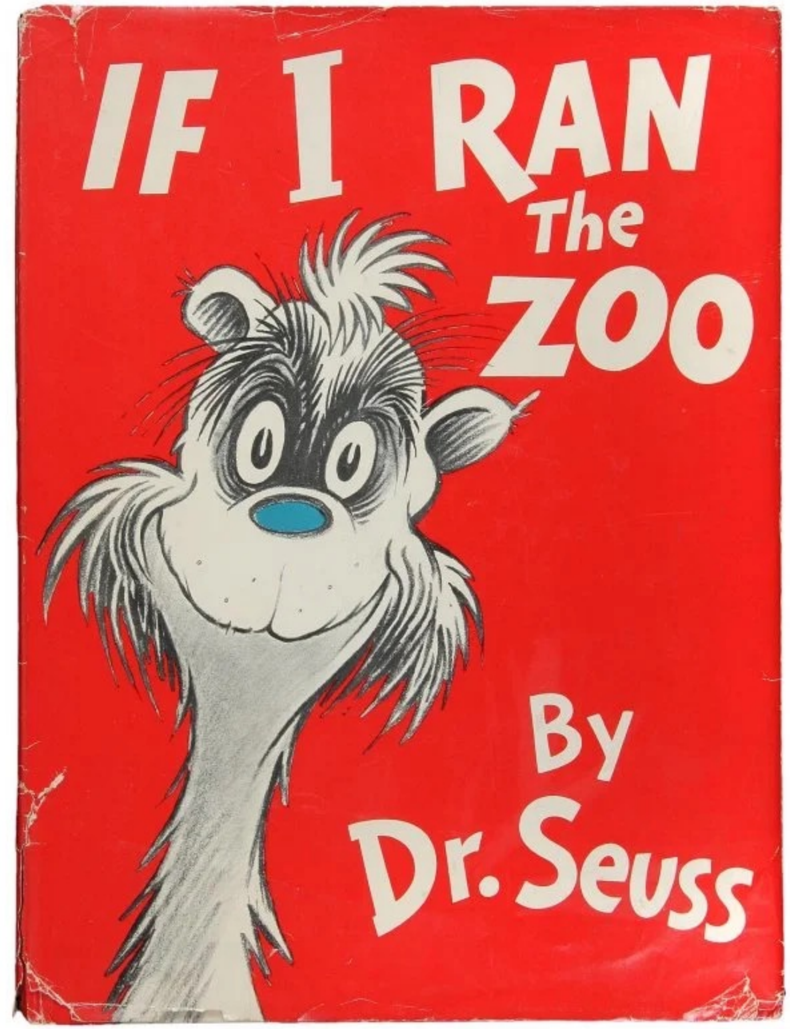
- Suppose as a cost cutting measure, Zoo management decides a keeper will be responsible for a cage – and all the animals in that cage.

Programs have to change, because the position in the database after a GN/GNP call may not be the same anymore!

Will see how SQL addresses this

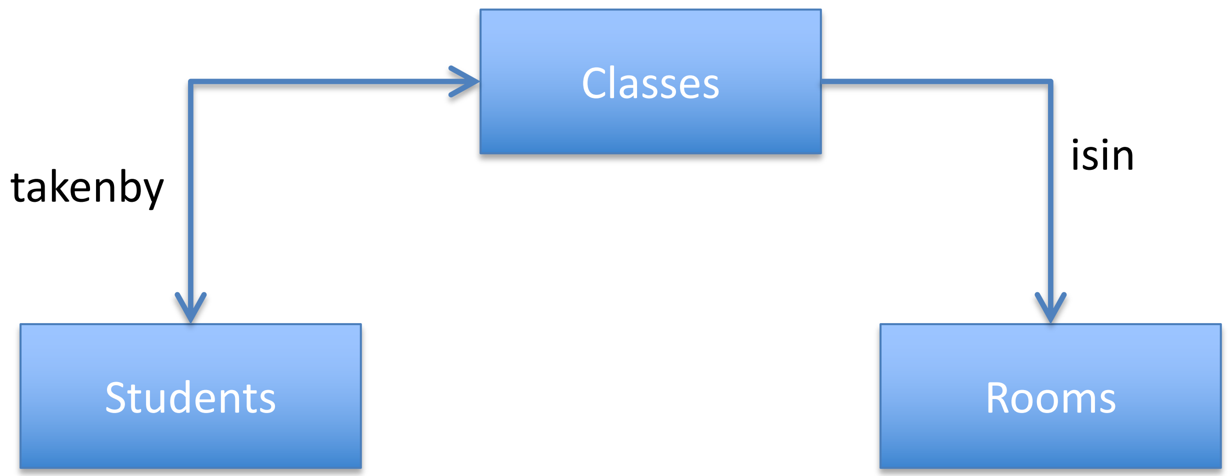
Schemas Change for Many Reasons

- Management decides to have “patrons” who buy cages
  - Need to add a patronid column
- Feds change the rules (OSHA)
  - Keepers can keep at most 2 cages
- Tax rules change (IRS)
- Merge with another zoo



Study break #2

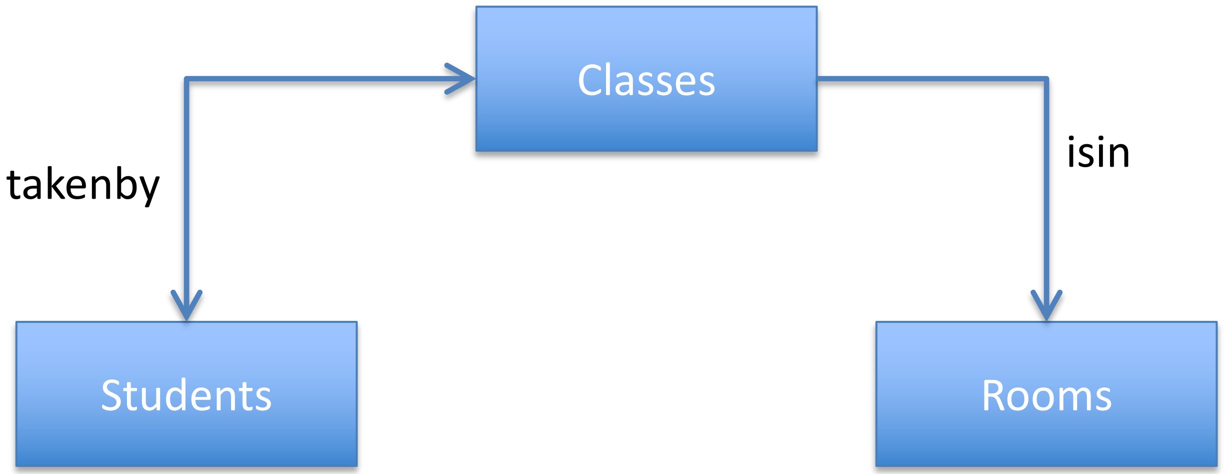
- Consider a course schema with students, classes, rooms (each has a number of attributes)



Classes in exactly one room
Students in zero or more classes
Classes taken by zero or more students
Rooms host zero or more classes

Questions

1. Describe one possible hierarchical schema for this data
2. Is there a hierarchical representation that is free of redundancy?



Solution

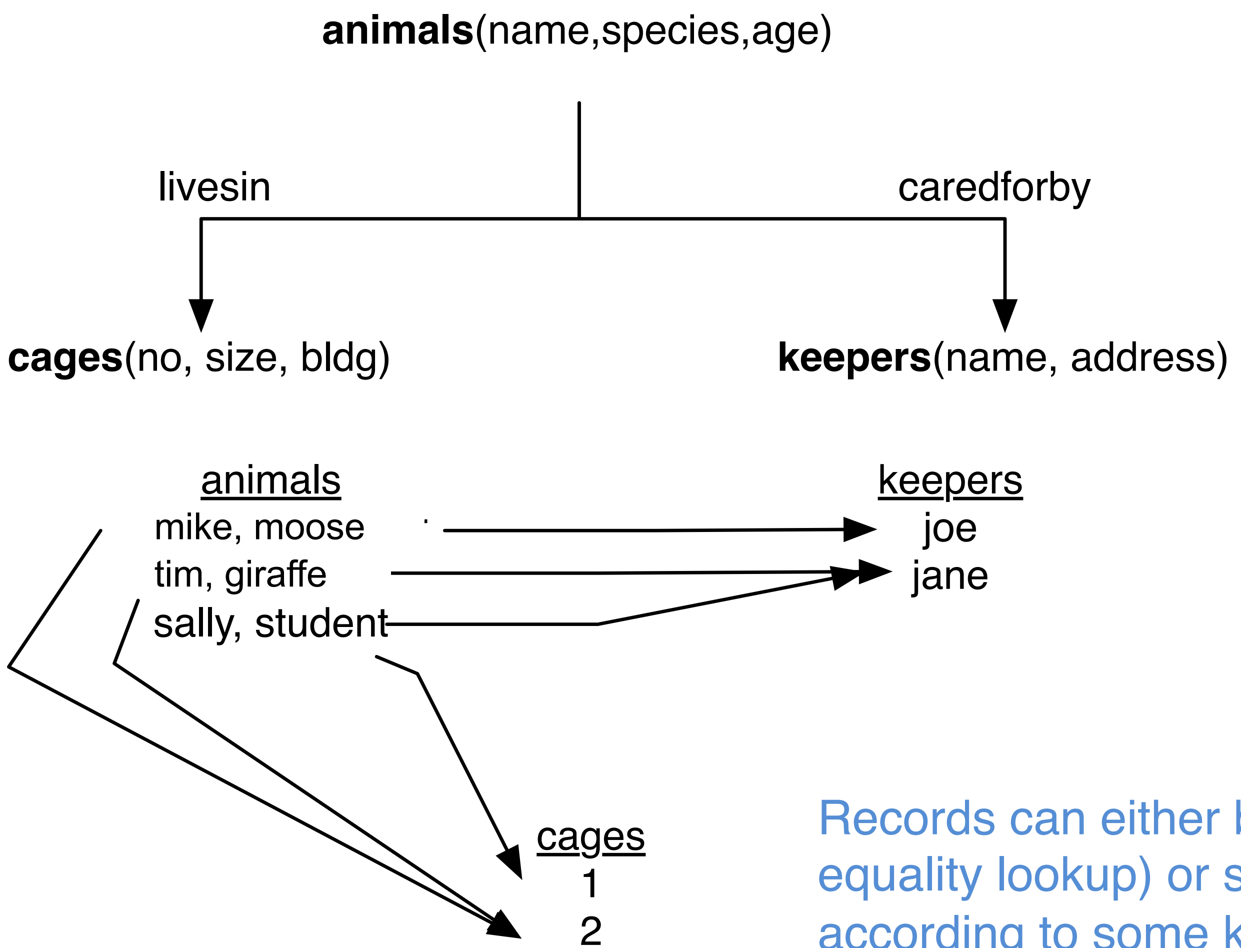
- Many are possible; one example:
  - Classes
    - Students
      - Rooms
- Duplicates data about students,
  - Students take multiple classes, rooms host multiple classes
- Any other arrangement also duplicates data

CODASYL

- **Conference/Committee on Data Systems Languages**
  - Responsible for COBOL
- CODASYL data model developed by consortium of large companies in the 70’s
- Designed to address limitations of IMS/PL1
- Graph or network-based data model

The Computer Museum in Boston celebrated COBOL’s 25th anniversary on May 16, 1985. The COBOL tombstone sent to Charles Phillips (see his adjoining article) was presented to the museum. Here surrounding the tombstone (left to right): Ron Hamm, Jack Jones, Jan Prokop, Oliver Smoot, Tom Rice, Donald Nelson, Grace Hopper, Michael O’Connell, and Howard Bromberg (photo by Lilian Kemp). At the museum’s celebration, Bromberg told the following tale of the tombstone.

Example CODASYL Network



Records can either be hashes (allowing equality lookup) or sorted ("clustered") according to some key (allowing a range lookup).

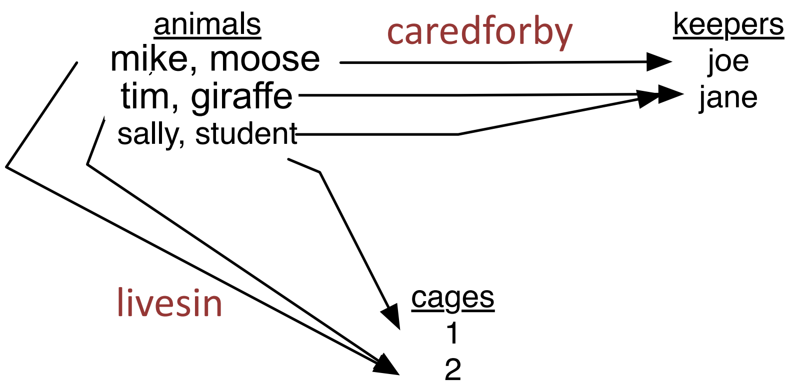
Example: Find Cages Joe Keeps

Find keepers (name = 'Joe')

Until done:

Find next animal in caredforby

Find cage in livesin



- Programming is finding an entry point and navigating around in multidimensional space
  - Each line of code is implicitly at some location in this structure
  - Have to remember where you are

Codasyl Problems

- Incredibly complex —
  "Navigational Programming"
- Programs lack physical or logical data independence
  - Can't change schema w/out changing programs;
  - Can't change physical representation either b/c different index types might or might not support different operations
- Some of this could have been fixed by adding a high-level language to CODASYL
- Relational model was a clean-slate approach designed to fix this

Relational Principles

- Simple representation
- Set-oriented programming model that doesn't require "navigation"
- No physical data model description required(!)
  - E.g., no specification of sort orders, hashes, etc

Relational Data Model

- All data is represented as tables of records (*tuples)*
- Tables are unordered sets (no duplicates)
- Database is one or more tables
- Each relation has a *schema* that describes the types of the columns/fields
- Each field is a primitive type -- not a set or relation
- Physical representation/layout of data is not specified (no index types, nestings, etc)

Zoo Tables

Foreign Keys

Animals

Primary Key

id	name	age	species	cageno	keptby	feedtime
1	Mike	3	Moose	1	1	10:00 am
2	Tim	12	Giraffe	1	2	11:00 am
3	Sally	1	Student	2	1	1:00 pm

Cages

Primary Key

no	building
1	1
2	2

Keepers

Primary Key

id	name
1	Jane
2	Joe

Schema: Animals
(id: int,
name: string,
age: int,
species: string,
cageno: int **references** cages.no,
keptby: int **references** keepers.id.
feedtime: time)

Zoo Tables (original schema)

Animals

id	name	age	species	cageno
1	Mike	3	Moose	1
2	Tim	12	Giraffe	1
3	Sally	1	Student	2

Primary Key (id) — Foreign Key (cageno)

Cages

no	feedtime	building
1	12:30	1
2	1:30	2

Primary Key (no)

Keepers

id	name
1	Jane
2	Joe

Primary Key (id)

Keeps

kid	cageno
1	1
1	2
2	1

Foreign Key (kid) — Foreign Key (cageno)

Relational Algebra

- **Projection** (π(T,c1, …, cn))
  - select a subset of columns c1 .. cn
- **Selection** (σ(T, pred))
  - select a subset of rows that satisfy pred
- **Cross Product** (T1 x T2)
  - combine two tables
- **Join** ($\bowtie$(T1, T2, pred)) = σ(T1 x T2, pred)
  - combine two tables with a predicate
- Plus set operations (UNION, DIFFERENCE, etc)
- "Algebra" – Closed under its own operations
  - Every expression over relations produces a relation

Join as Cross Product

Animals

name	cageno
Mike	1
Tim	1
Sally	2

Cages

no	bldg
1	32
2	36

cageno	no	name	bldg
1	1	Mike	32
1	2	Mike	36
1	1	Tim	32
1	2	Tim	36
2	1	Sally	32
2	2	Sally	36

Find animals in bldg. 32

σ (
    ⋈(
        animals,
        cages,
        animals.cageno = cages.no
    ),
    bldg = 32
)

Real implementations do not ever materialize the cross product

Join as Cross Product

Animals

name	cageno
Sam	1
Tim	1
Sally	2

Cages

no	bldg
1	32
2	36

cageno	no	name	bldg
1	1	Mike	32
~~1~~	~~2~~	~~Mike~~	~~36~~
1	1	Tim	32
~~1~~	~~2~~	~~Tim~~	~~36~~
~~2~~	~~1~~	~~Sally~~	~~32~~
2	2	Sally	36

Find animals in bldg. 32

```
σ (
    ⋈(
        animals,
        cages,
        animals.cageno = cages.no
    ),
    bldg = 32
)
```

1. animals.cageno = cages.no

Join as Cross Product

Animals

name	cageno
Sam	1
Tim	1
Sally	2

Cages

no	bldg
1	32
2	36

cageno	no	name	bldg
1	1	Mike	32
~~1~~	~~2~~	~~Mike~~	~~36~~
1	1	Tim	32
~~1~~	~~2~~	~~Tim~~	~~36~~
~~2~~	~~1~~	~~Sally~~	~~32~~
~~2~~	~~2~~	~~Sally~~	~~36~~

Find animals in bldg. 32

```
σ (
    ⋈(
        animals,
        cages,
        animals.cageno = cages.no
    ),
    bldg = 32
)
```

1. animals.cageno = cages.no
2. **bldg = 32**

Do you think this is how databases actually execute joins?

Relational Identities

- **Join reordering**
  - $A \bowtie B = B \bowtie A$
  - $(A \bowtie B)$ join $C = A \bowtie (B \bowtie C)$
- **Selection reordering**
  - $\sigma_1(\sigma_2(A)) = \sigma_2(\sigma_1(A))$
- **Selection push down**
  - $\sigma(A \bowtie_{pred} B) = \sigma(A) \bowtie_{pred} \sigma(b)$
  - σ may only apply to one table
- **Projection push down**
  - $\pi(\sigma(A)) = \sigma(\pi(A))$
  - As long as π doesn't remove fields used in σ
  - Also applies to joins

Push Down Example

```
σ (
  ⋈ (
    animals,
    cages,
    animals.cageno = cages.no
  ),
  bldg = 32
)
```

```
⋈ (
  animals,
  σ (
      cages,
      bldg = 32
  )
  animals.cageno = cages.no
)
```

Join Ordering Example

- Find buildings Joe keeps
- SQL

```
SELECT building
FROM cages JOIN keeps ON no = cageno
JOIN keepers on kid = id
WHERE name = 'Joe'
```

SQL query executor free to choose either ordering!
Text of SQL query is not an ordering

```
⋈ (
    ⋈ (
        cages,
        keeps,
        no = cageno
    ),
    σ (
        keepers,
        name = 'Joe'
    ),
    kid = id
)
```

```
⋈ (
    cages,
    ⋈ (
        σ (
            keepers,
            name = 'Joe'
        ),
        keeps,
        kid=id
    ),
    no = cageno
)
```

Best ordering depends on sizes of tables

Filtered keepers may be *much* smaller

Study Break # 2

Schema:

classes: (<u>cid</u>, c_name, *c_rid*, …)
rooms: (<u>rid</u>, bldg, …)
students: (<u>sid</u>, s_name, …)
takes: (*t_sid*, *t_cid*)

SELECT s_name FROM student,takes,classes
WHERE t_sid=sid AND t_cid=cid
AND c_name='6.830'

Questions

- Write an equivalent relational algebra expression for this query
- Are there other possible expressions?
- Do you think one would be more "efficient" to execute? Why?

```
SELECT s_name FROM student,takes,classes
WHERE t_sid=sid AND t_cid=cid
AND c_name='6.830'
```

Solution

SELECT s_name FROM student,takes,classes
WHERE t_sid=sid AND t_cid=cid
AND c_name='6.830'

```
⋈ (
    student,
    ⋈ (
        σ (
            classes,
            c_name = '6.830'
        ),
      takes,
      t_cid=cid
    ),
    t_sid=sid
)
```

Filtering first is probably a good idea

Filtered table is small, so do join with it and classes first

Will formalize this intuition in a few classes

IMS v CODASYL v Relational

	IMS	CODASYL	Relational
Many to many relationships without redundancy	✗	✓	✓
Declarative, non "navigational" programming	✗	✗	✓

IMS v CODASYL v Relational

	IMS	CODASYL	Relational
Many to many relationships without redundancy	✗	✓	✓
Declarative, non "navigational" programming	✗	✗	✓
Physical data independence	✗	✗	✓

Physical Independence

Can change representation of data without needing to change code

Example:

```
SELECT a.name FROM animals AS a, cages AS c
WHERE a.cageno = c.no AND c.bldg = 32
```

- Nothing about how animals or cages tables are represented is evident
  - Could be sorted, stored in a hash table / tree, etc
  - Changing physical representation will not change SQL
- No specification of implementation
- Both CODASYL and IMS expose representation-dependent operations in their query API

IMS v CODASYL v Relational

	IMS	CODASYL	Relational
Many to many relationships without redundancy	✗	✓	✓
Declarative, non "navigational" programming	✗	✗	✓
Physical data independence	✗	✗	✓
Logical data independence	✗	✗	✓

Logical Data Independence

- What if I want to change the schema without changing the code?
- No problem if just adding a column or table
- *Views* allow us to map old schema to new schema, so old programs work
  - *Even when changing existing fields*

Key Idea: View

- View is a logical definition of a table in terms of other tables
- E.g., a view computing animals per cage

```
CREATE VIEW cage_count as
(SELECT cageno, count(*)
FROM animals JOIN cages ON cageno=no
GROUP by cageno
)
```

This view can be used just like a table in other queries

Views Example

- Suppose I want to add multiple feedtimes?
- How to support old programs?
  - Rename existing animals table to animals2
  - Create feedtimes table
  - Copy feedtime data from animals2
  - Remove feedtime column from animals2
  - Create a view called animals that is a query over animals2 and feedtimes

```
CREATE VIEW animals as (
SELECT name, age, species, cageno,
     (SELECT feedtime FROM feedtimes WHERE animalid = id  LIMIT 1)
     FROM animals2
)
```

Summary: IMS v CODASYL v Relational

	IMS	CODASYL	Relational
Many to many relationships without redundancy	✗	✓	✓
Declarative, non "navigational" programming	✗	✗	✓
Physical data independence	✗	✗	✓
Logical data independence	✗	✗	✓

Next time: Fancy SQL

Today

- Why database systems?

- User's perspective:
  - Modeling data
  - Querying data

- Data Models

Zoo Website Features

- Admin interface
  - Edit
  - Add an animal
- Public
  - Pictures & Maps

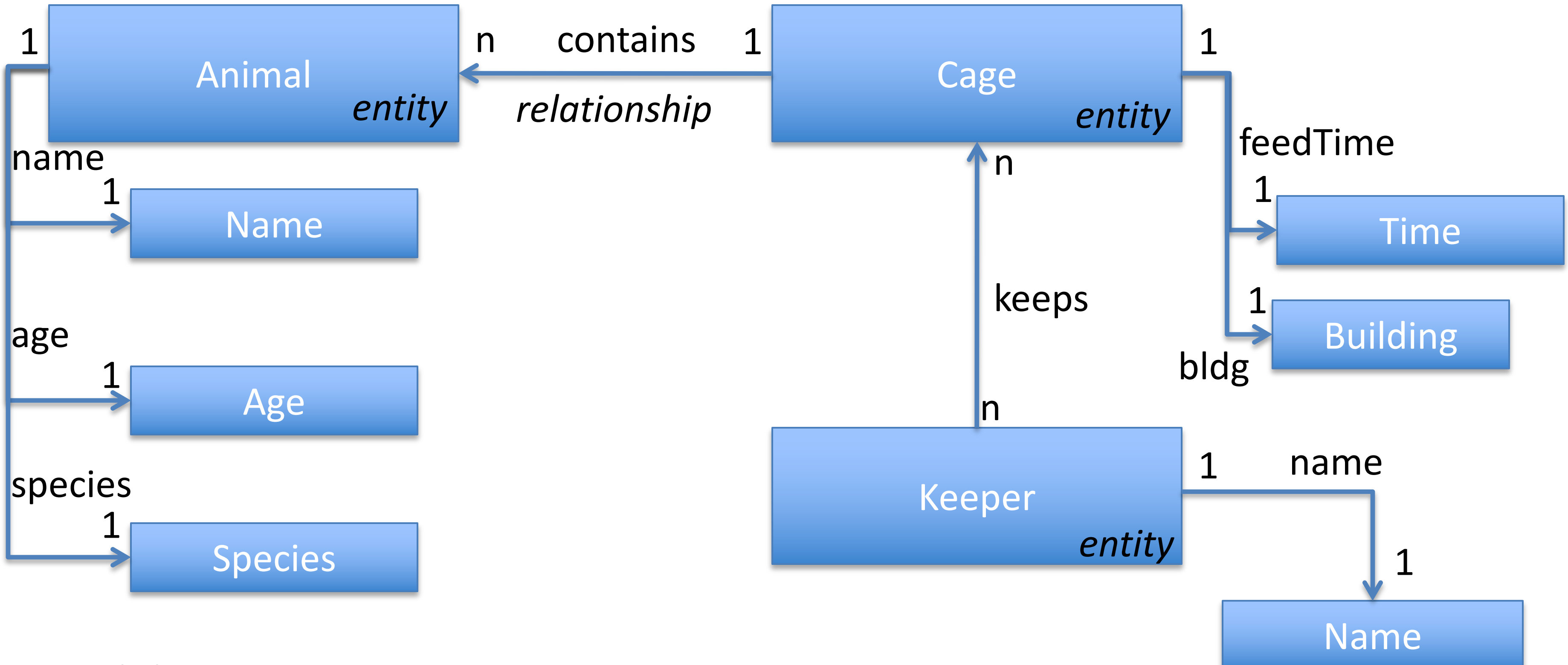
San Diego Zoo

- Zookeeper
  - Feed times

- 1K animals, 5K URLs, 10 admins, 200 keepers

Zoo Data Model

Entity Relationship Diagram



Animals have names, ages, species
Keepers have names
Cages have feeding times, buildings
Animals are in 1 cage; cages have multiple animals
Keepers keep multiple cages, cages kept by multiple keepers

Our Zoo

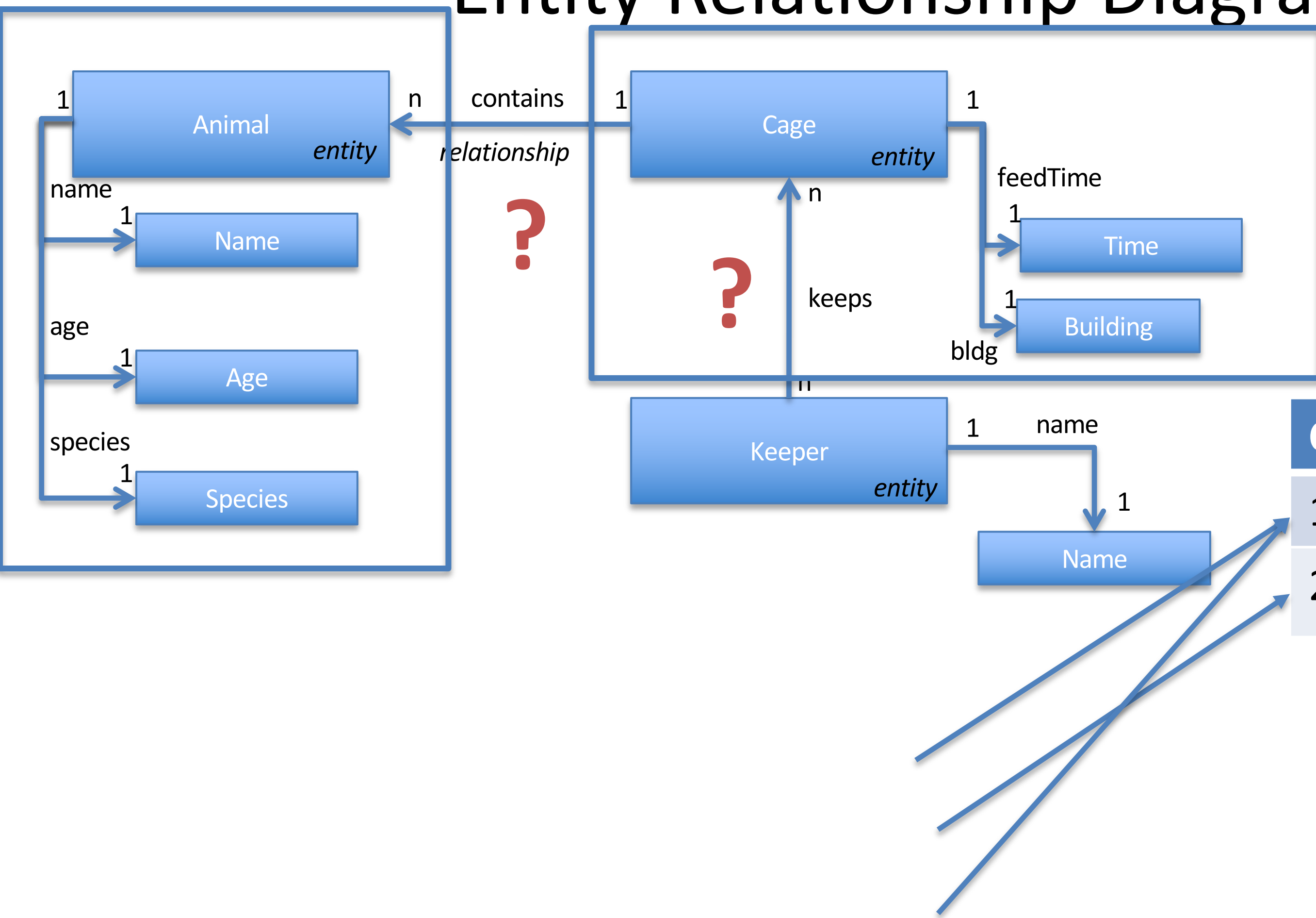
Mike the Moose

Tim the Giraffe

Sally the Student

Zoo Data Model

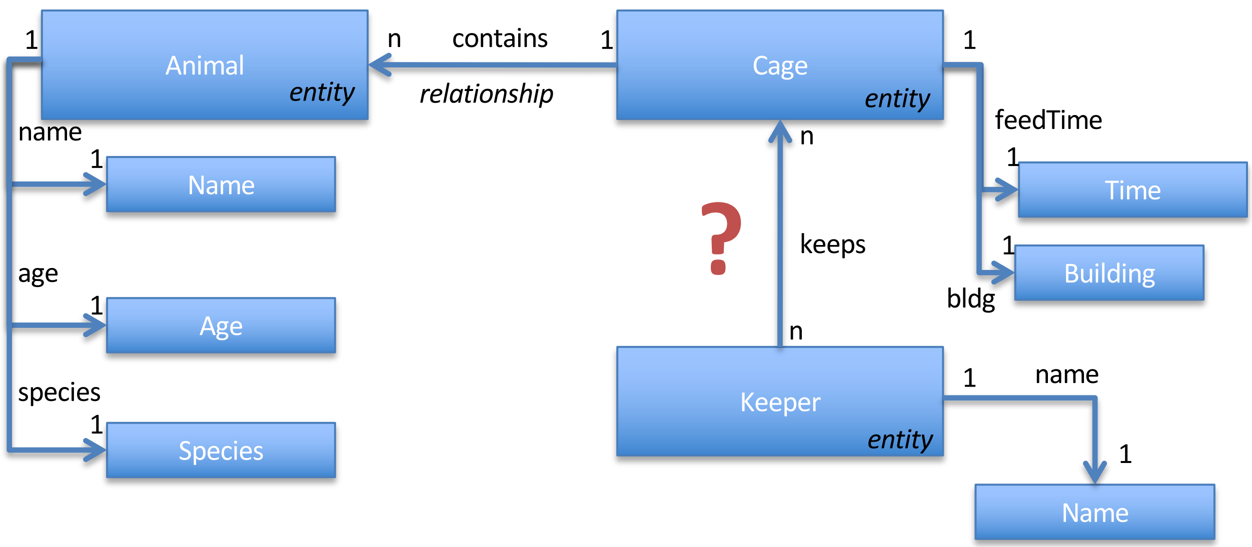
Entity Relationship Diagram



Cageid	Feedtime	Bldg
1	1:30	1
2	2:30	2

Zoo Data Model

Entity Relationship Diagram



Cageid	Feedtime	Bldg
1	1:30	1
2	2:30	2

keeperid	cageid
1	1
1	2
2	1

keeperid	name
1	jenny
2	joe

Study Break #1

- Questions
  - Are there other ways to represent this zoo data than a collection of tables?
  - What are tradeoffs in different representations?

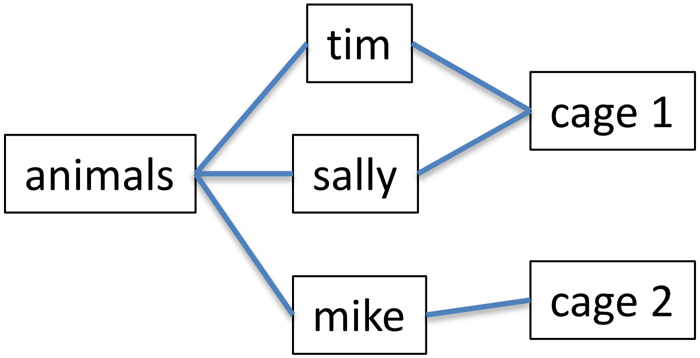
Alternatives to Relations

Hierarchy

cage 1
- tim
  - giraffe
  - 13 yrs
- sally
  - student
  - 1 yr

cage 2
- mike
  - moose
  - 3 yrs

Graph



Multiple Tabular Representations Are Possible

name	age	species	cageno	feedtime	bldg
tim	13	giraffe	1	1:30	1
mike	3	moose	2	2:30	2
sally	1	student	1	1:30	1

Is this a good representation? Why or why not?

Not “Normalized” – repeats data. More in later lectures!

SQL – Structured Query Language

```
SELECT  field1, …, fieldM
FROM table1, …
WHERE condition1, …

INSERT INTO table VALUES (field1, …)

UPDATE table SET field1 = X, …
WHERE condition1,…
```

Names of Giraffes

- Imperative

```
for each row r in animals
   if r.species = 'giraffe'
      output r.name
```

- Declarative

```
SELECT r.name FROM animals
WHERE r.species = 'giraffe'
```

Cages in Building 32

- Imperative

```
for each row a in animals
   for each row c in cages
      if a.cageno = c.no and c.bldg = 32
         output a
```

NESTED LOOPS

- Declarative

JOIN

```
SELECT a.name FROM animals AS a, cages AS c
WHERE a.cageno = c.no AND c.bldg = 32
```

Average Age of Bears

- Declarative

```
SELECT AVG(age) FROM animals
WHERE species = 'bear'
```

Complex Queries

Find pairs of animals of the same species and different genders older than 1 year:

```
SELECT a1.name,a2.name
FROM animals as a1, animals as a2
WHERE a1.gender = M and a2.gender = F
AND a1.species = a2.species
AND a1.age > 1 and a2.age > 1
```

"self join"

Find cages with salamanders fed later than the average feedtime of any cage:

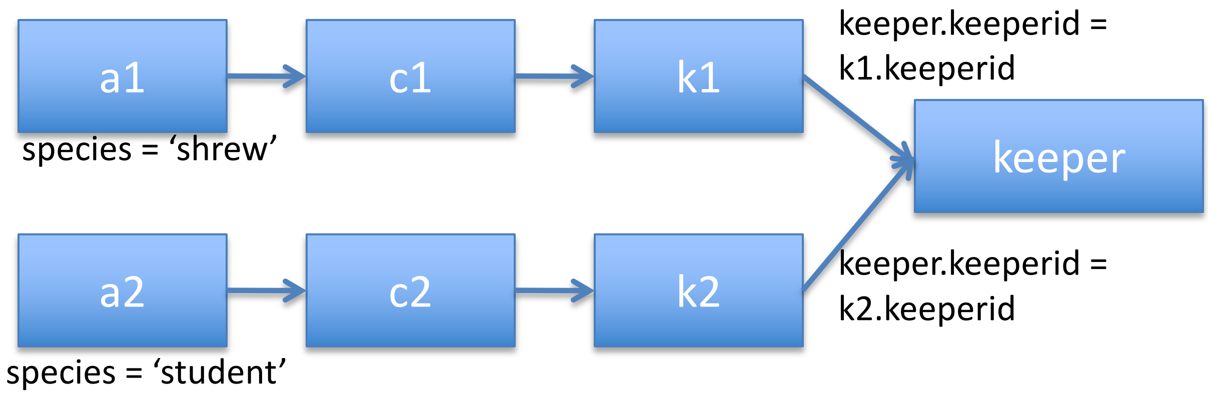
```
SELECT cages.cageid FROM cages, animals
WHERE animals.species = ’salamander'
AND animals.cageid = cages.cageid
AND cages.feedtime >
    (SELECT AVG(feedtime) FROM cages )
```

"nested queries"

Complex Queries 2

Find keepers who keep both students and salamanders:

```
SELECT keeper.name
FROM keeper, cages as c1, cages as c2,
        keeps as k1, keeps as k2, animals as a1, animals as a2
WHERE c1.cageid = k1.cageid AND keeper.keeperid = k1.keeperid
AND c2.cageid = k2.cageid AND keeper.keeperid = k2.keeperid
AND a1.species = ’student' AND a2.species = ’salamander'
AND c1.cageid = a1.cageid AND c2.cageid = a2.cageid
```

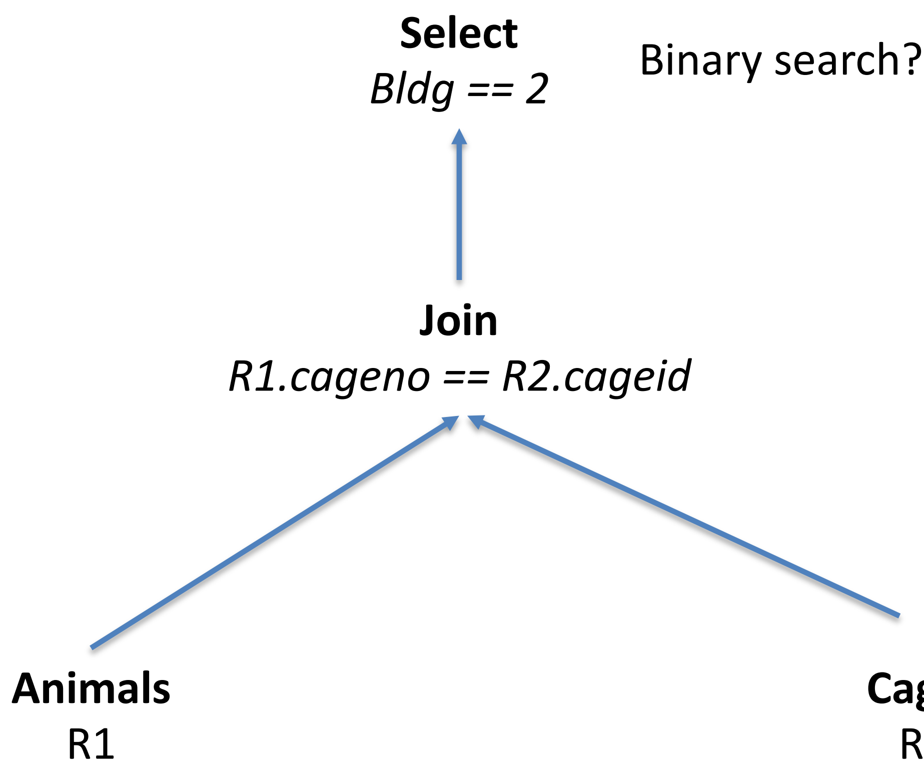


Declarative Queries: **What,** not **How**

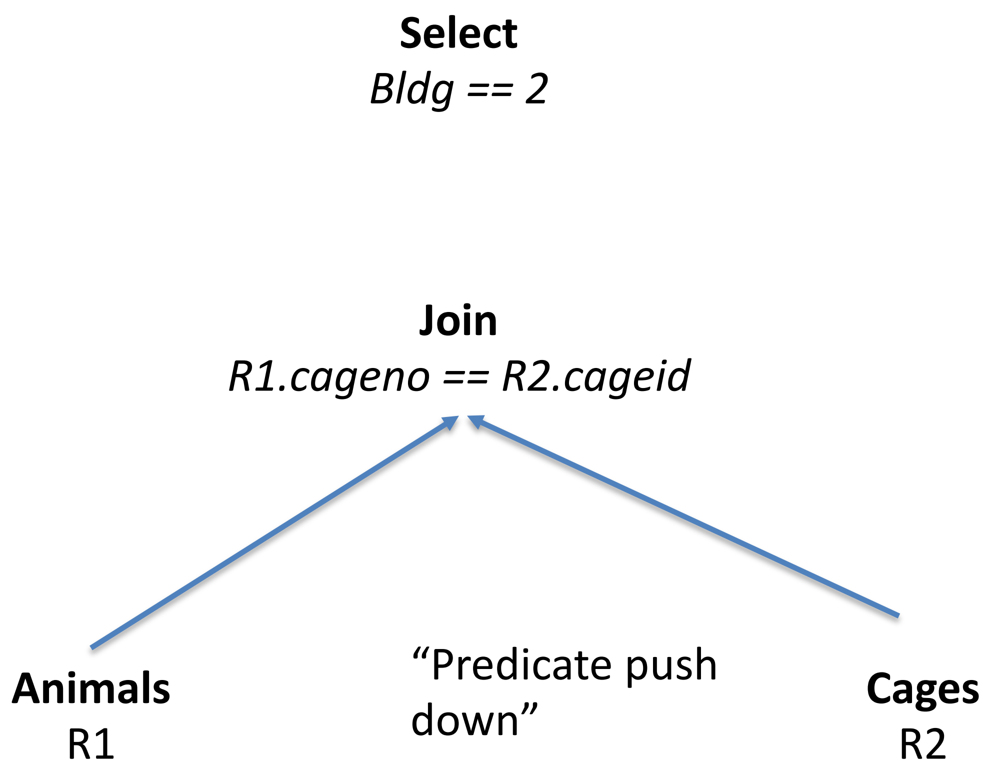
- Many possible proc
  query

- Besides looping thr
  could we do?
  - Sort animals on type
    - \+ good for "bears" query
    - \- Inserts are slower
  - Store animals table in a hash table or tree ("index")

SQL → Procedural Plan → Optimized Plan → Compiled Program



SQL → Procedural Plan → Optimized Plan → Compiled Program



SQL programmer just thinks in terms of table operations, not the order or implementation!

Summary: Database Systems

- Relational Model + Schema Design
- Declarative Queries
- Query Optimization
- Efficient access and updates to data
  - Recoverability
  - Consistency