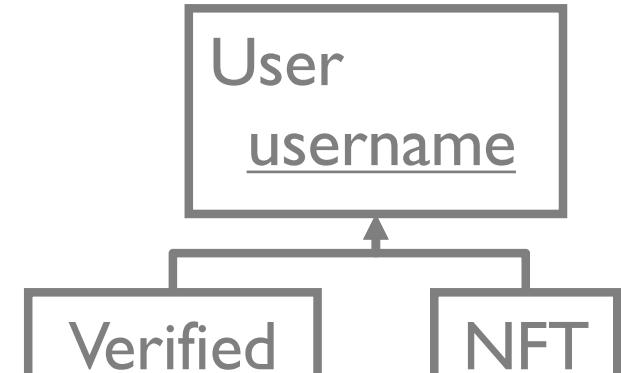
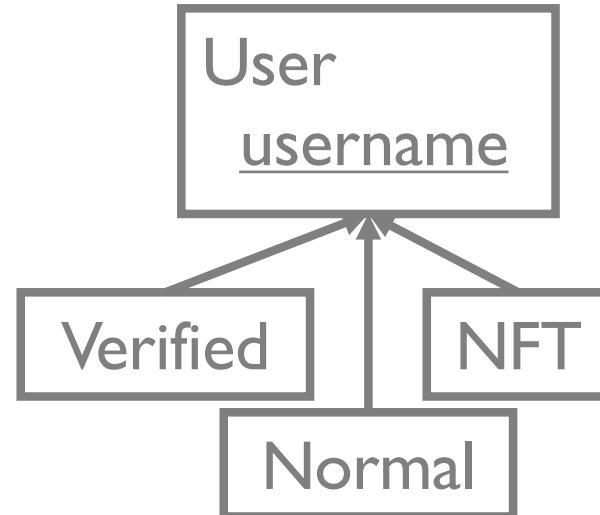
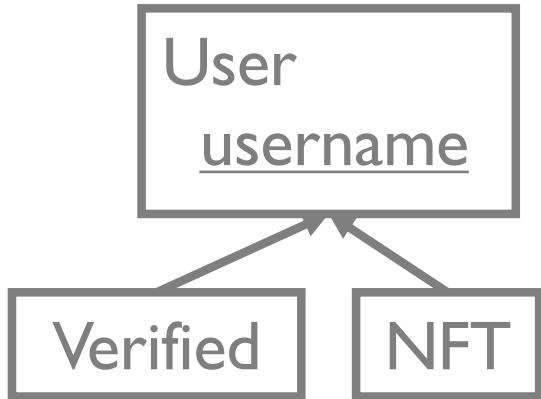


QUIZ TIME!!

<http://w4lll.github.io/quiz>

A twitter **user** is identified by username, and can be a **verified user**, an **NFT user**, a **normal** user, or any combination.



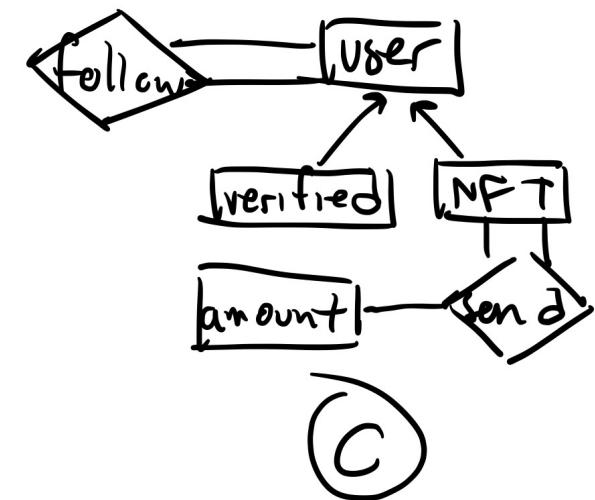
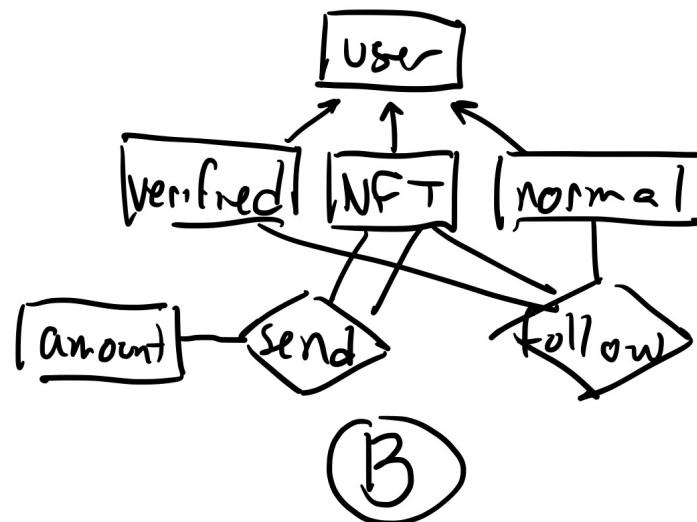
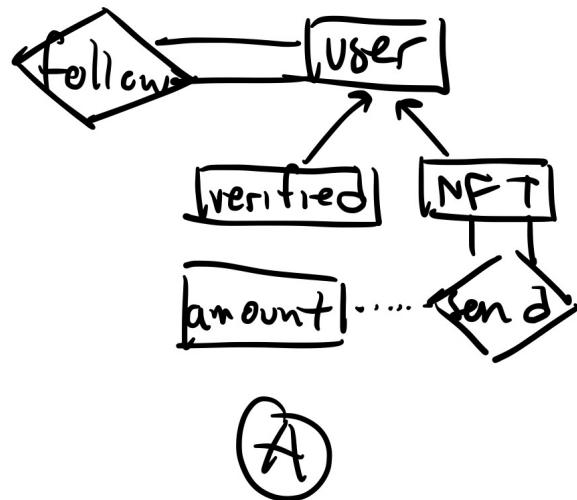
A

B

C

<http://w4lll.github.io/quiz>

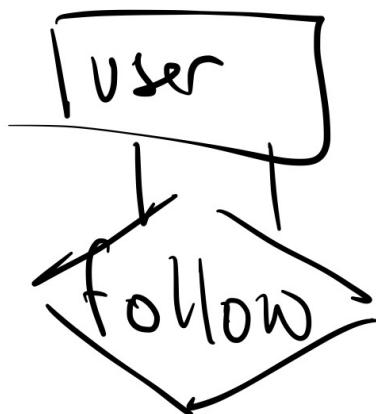
Any **user** can follow another **user**, but only **NFT** users can send **bitcoin** to each other.



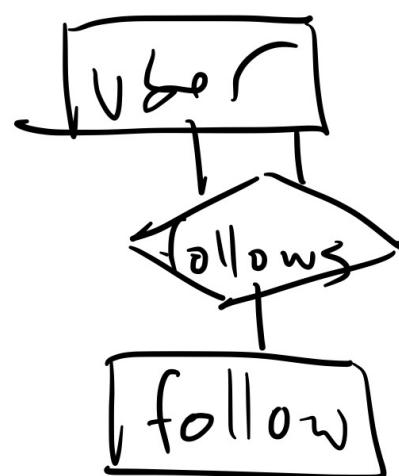
<http://w4lll.github.io/quiz>

Suppose Eugene follows BigDataMemes.

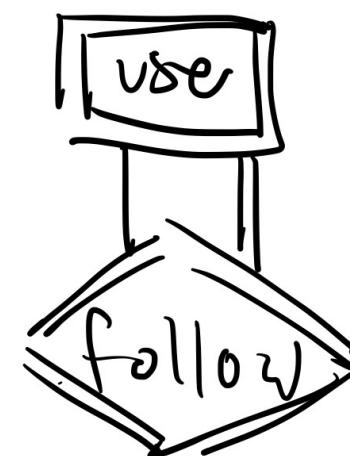
If Eugene deletes his account, BigDataMemes should not have Eugene as a follower.



(A)



(B)



(C)

Logistics

Project:

- teams can span 0I and H0I sections
- sign up for 15 min meetings (mandatory)

Looking for teammates?

- use megathread on discussion board
- meet people after class (outside classroom)

MON

7

TUE

8

WED

9

THU

10

FRI

11

<p>Rachel's Office Hours 7 – 9pm https://columbiauniversity.zoom.us/j/99563884586?pwd=TFpqc1MzUjd3TEpTb01</p>	<p>Jake's Office Hours 3 – 5pm https://columbiay.zoom.us/j/971181111?</p> <p>Project 1 Part 1 Meetings 3:30 – 5:30</p> <p>Project 1 Part 1 Meeting (Zach) 3 – 5pm</p> <p>Project 1 Part1 (Lucas) 3 – 5pm</p>	<p>Mausam's Office Hours 8 – 10am https://meet.google.com/doe-jdrw-xjn</p> <p>Kostis Kaffes (TT) 11:40am – 1pm Zoom Link below</p> <p>Eugene Wu's OH 2 – 3pm</p> <p>Project 1 Part 1 (Twisha) 5 – 7pm</p> <p>Twisha's Office Hours 5 – 7pm</p>	<p>Bochao Xie's Office Hours 10am – 1pm</p> <p>Project 1 Part 1 Meeting (Bochao) 11am – 1:30pm</p> <p>Project 1 Part 1 Meeting (Eugene) 11am – 1:30pm</p> <p>Lucas's OH 2 – 4pm</p>	<p>Project 1 Part 1 Meeting (Rachel) 12 – 2pm</p> <p>Project 1 Part 1 Meetings (Jake) 3 – 5pm</p>
--	---	---	---	---



Arnab Nandi liked



electronic meta-x @emax · 7h

my AI ethics lectures write themselves

...



bbc.co.uk

Supermarket cameras to guess age of alcohol buyers

Major chains begin testing automated age-verification, to avoid the wait for staff at self-checkouts.

L3

The Relational Model

Eugene Wu

Background

Relational model: most widely used data model in the world

Legacy Models

IMS hierarchical/Nested

CODASYL network

Other common models:

Key-Val, Matrix, Data frame

Overview of Key Relational Principles

Data redundancy (or how to avoid it)

Physical data independence

Can change physical storage w/out changing programs

Logical data independence

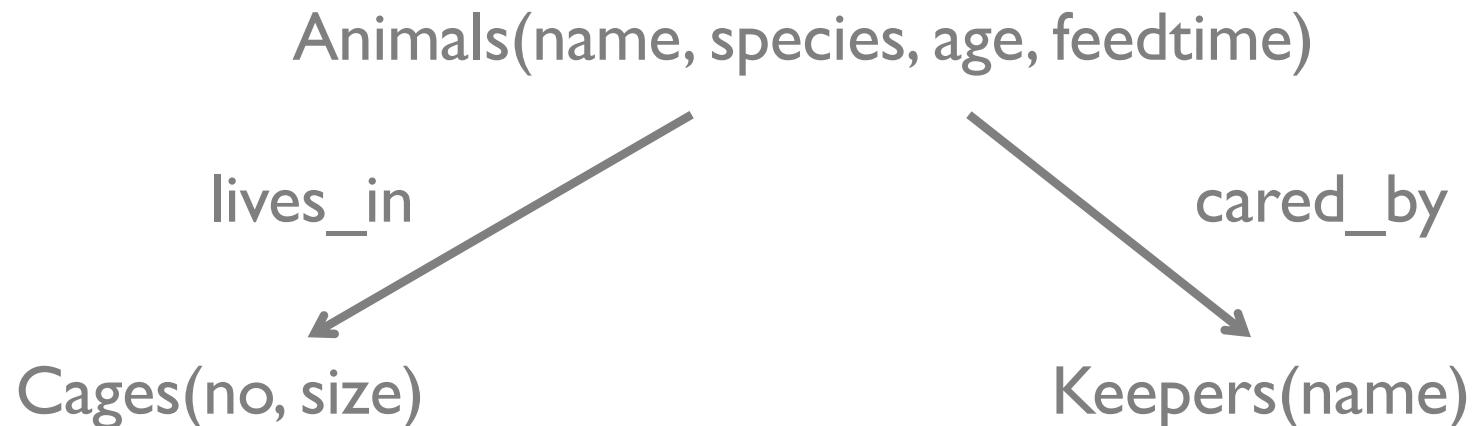
Can change schema without changing programs

High level languages

Historical Context (not on test)

Hierarchical model (IMS)
Network model (CODASYL)
Relational model (SQL/QUEL)

70s
80-90s

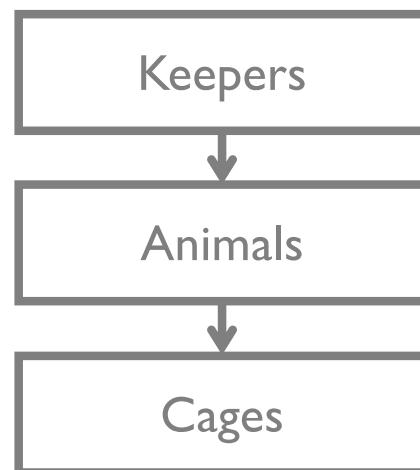


Hierarchical Model (IMS, circa 1968)

Segment types (objects)

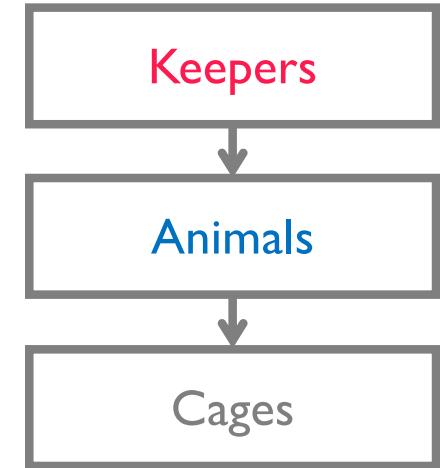
Segment instances (records)

Segment types form a tree



Data Stored Depth First

```
Jane          (HSK 1) { id: 1
  Bob, iguana, (2)   name: Jane
  1, 100ft2, (3)   animals: [
    Joe, student, (4)     { id: 2
    1, 100ft2, (5)       name: Bob
                           species: iguana,
                           cages: [
                             {id: 3,
                               no: 1,
                               size: 100
                             }]
                           }, ..
                           }, ..
```



What's repeated?

Inconsistencies possible, lack of protections

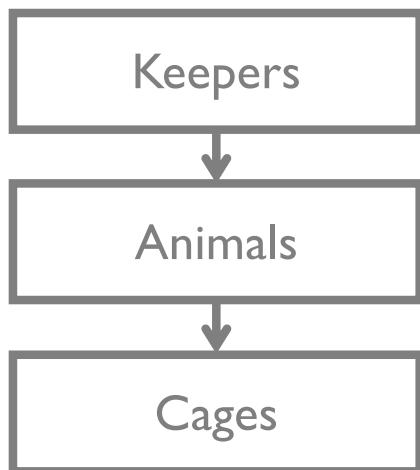
Similar to JSON

Can't Avoid Redundancy

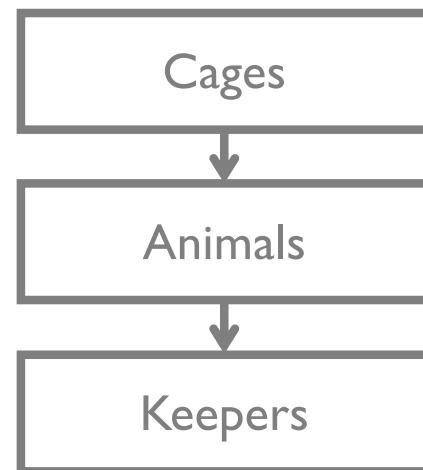
Segment types (objects)

Segment instances (records)

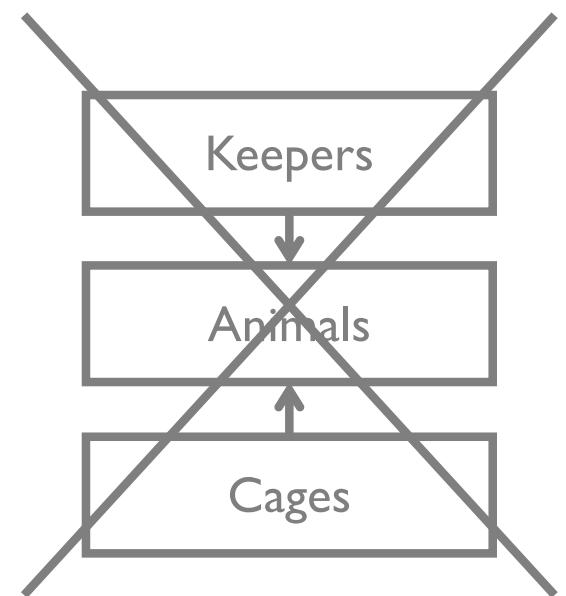
Segment types form a tree



Repeats cage data if
>1 animals in a cage



Repeats keeper data if
keeper cares for >1 animals



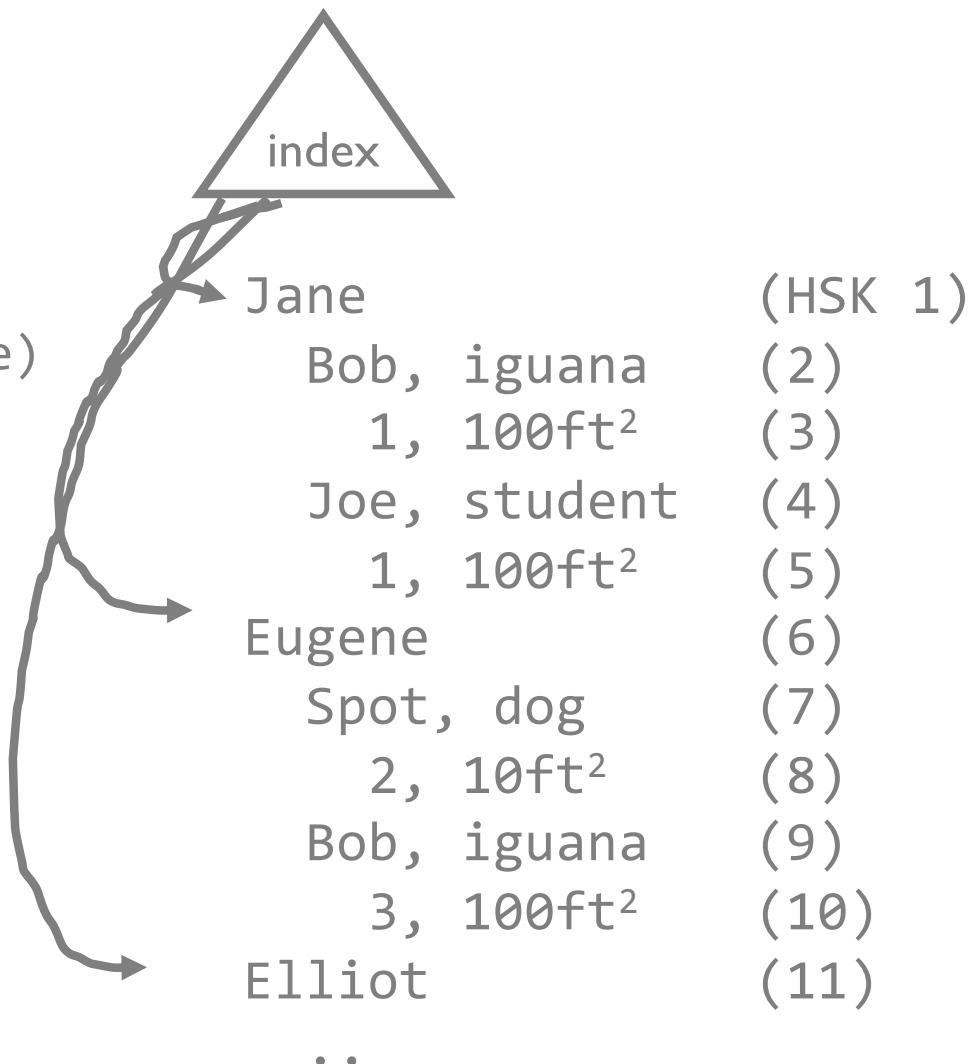
DAG != Tree

Hierarchical “Querying”

Fetch cages that Eugene manages

```
Goto(Keeper, name = Eugene)  
Until no more records  
    cage = NextInParent(Cage)  
    print cage.no
```

Output:



Hierarchical “Querying”

Fetch cages that Eugene manages

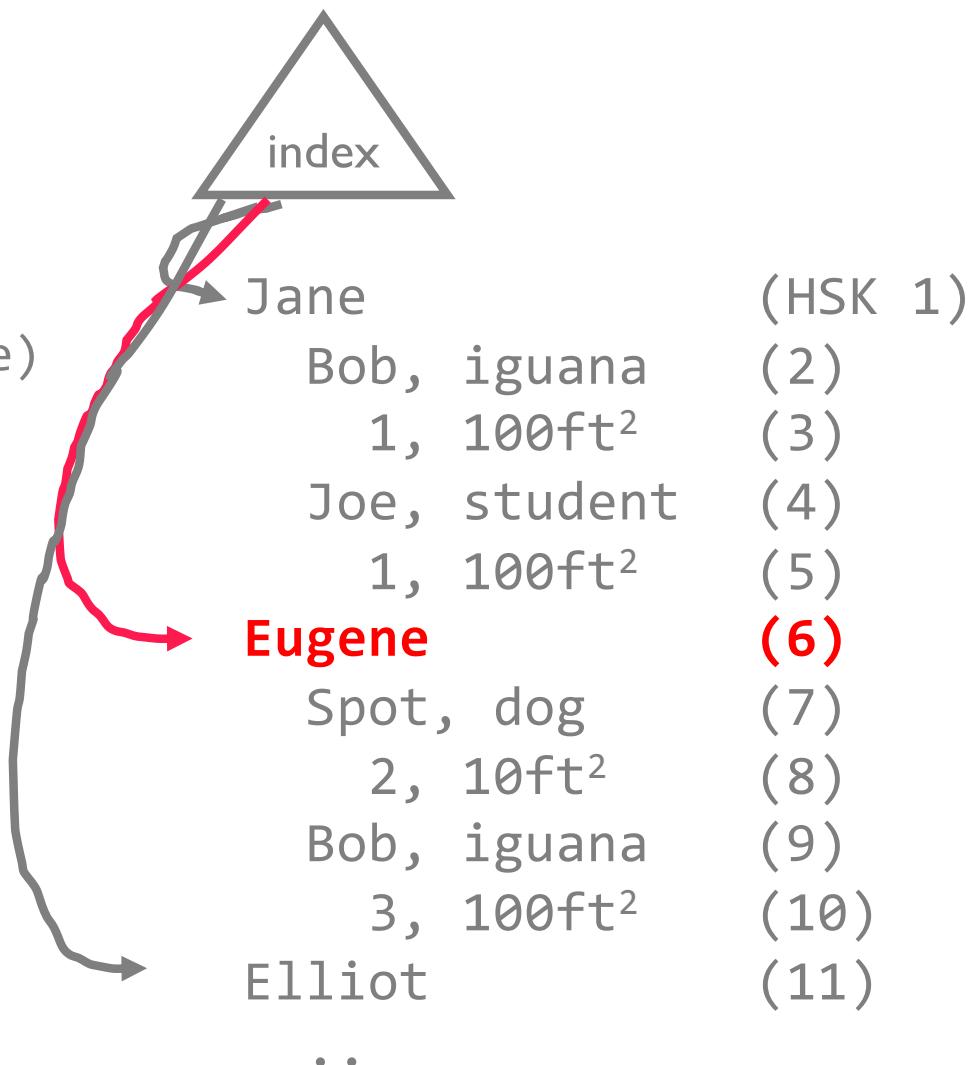
Goto(Keeper, name = Eugene)

Until no more records

 cage = NextInParent(Cage)

 print cage.no

Output:



Hierarchical “Querying”

Fetch cages that Eugene manages

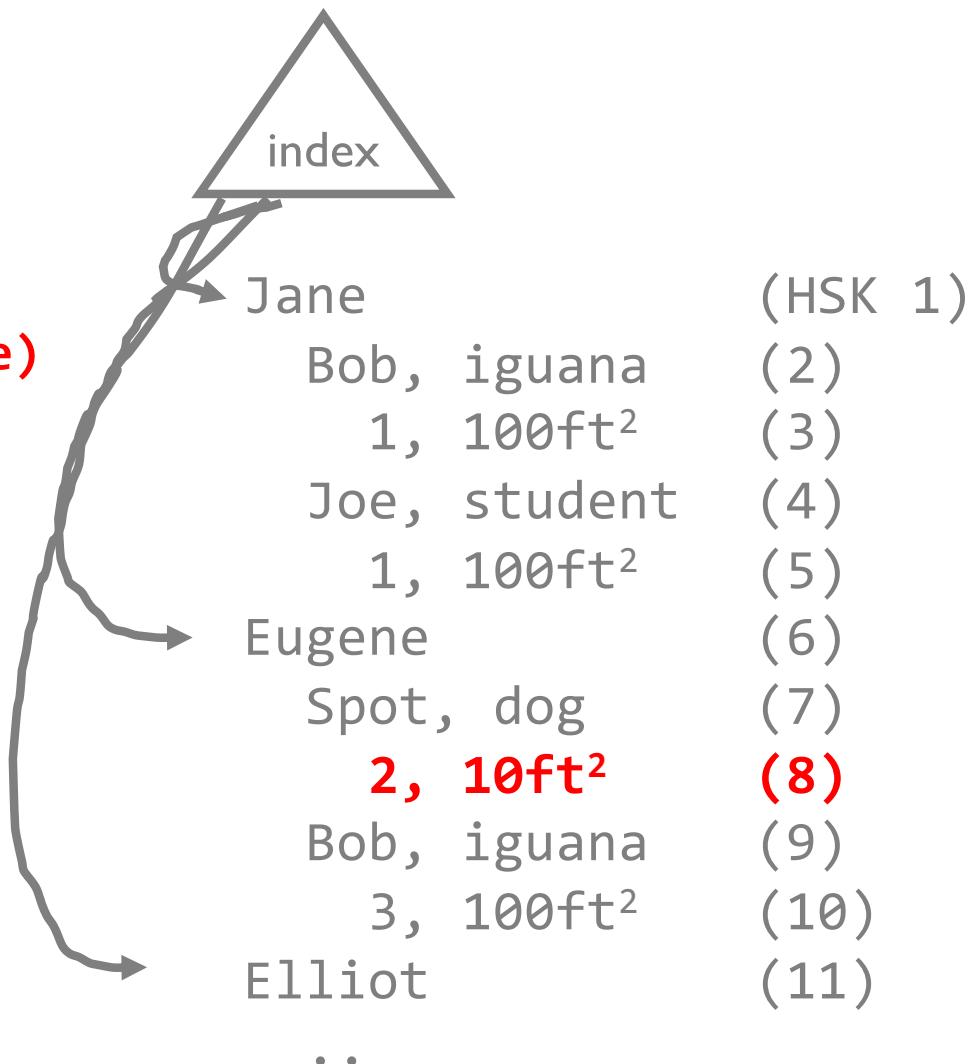
Goto(Keeper, name = Eugene)

Until no more records

cage = NextInParent(Cage)

print cage.no

Output:



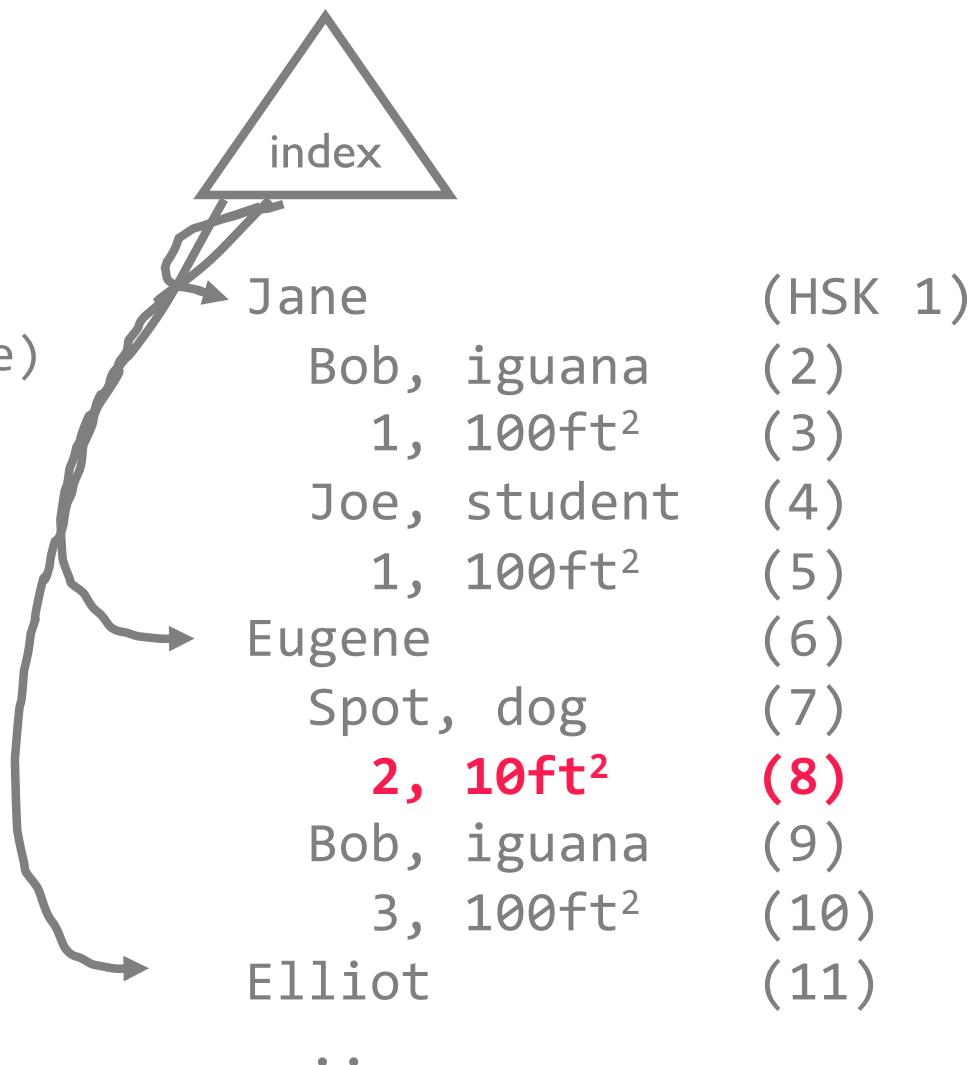
Hierarchical “Querying”

Fetch cages that Eugene manages

```
Goto(Keeper, name = Eugene)  
Until no more records  
    cage = NextInParent(Cage)  
print cage.no
```

Output:

2



Hierarchical “Querying”

Fetch cages that Eugene manages

Goto(Keeper, name = Eugene)

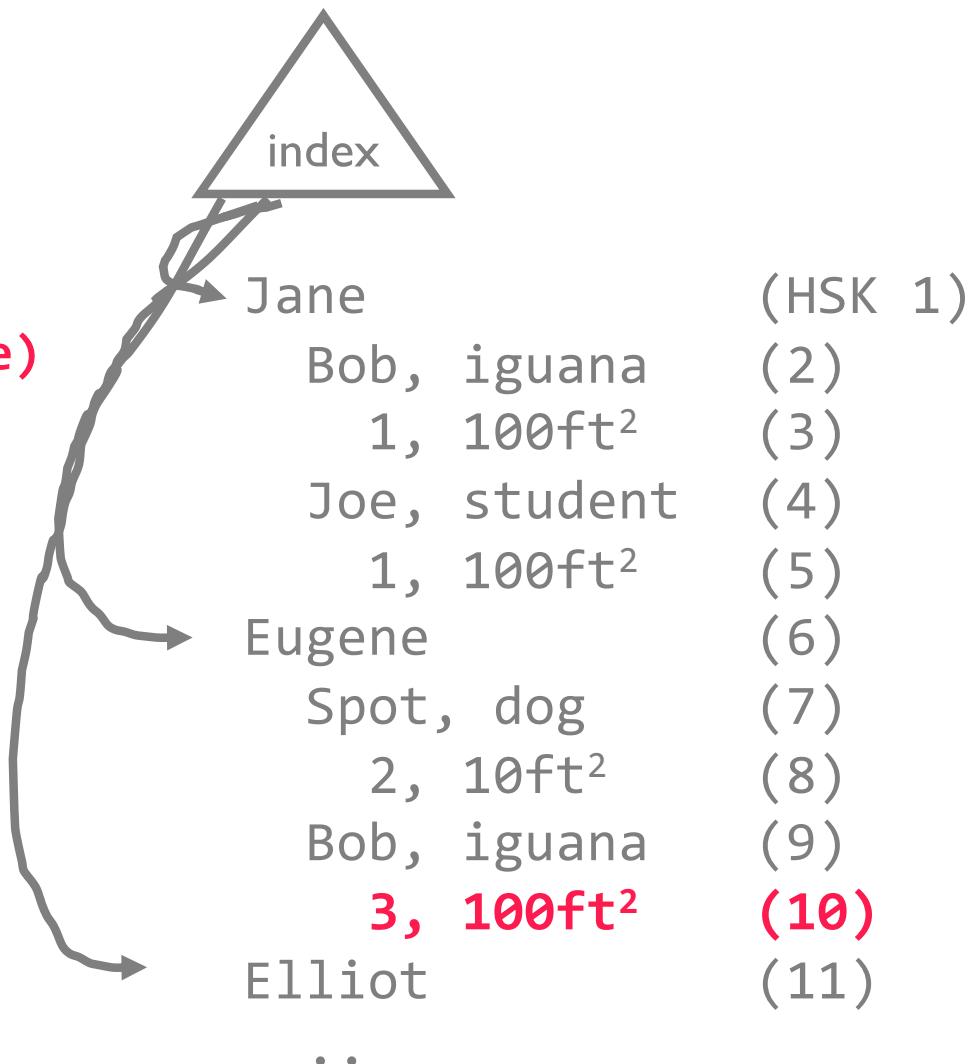
Until no more records

cage = NextInParent(Cage)

print cage.no

Output:

2



Hierarchical “Querying”

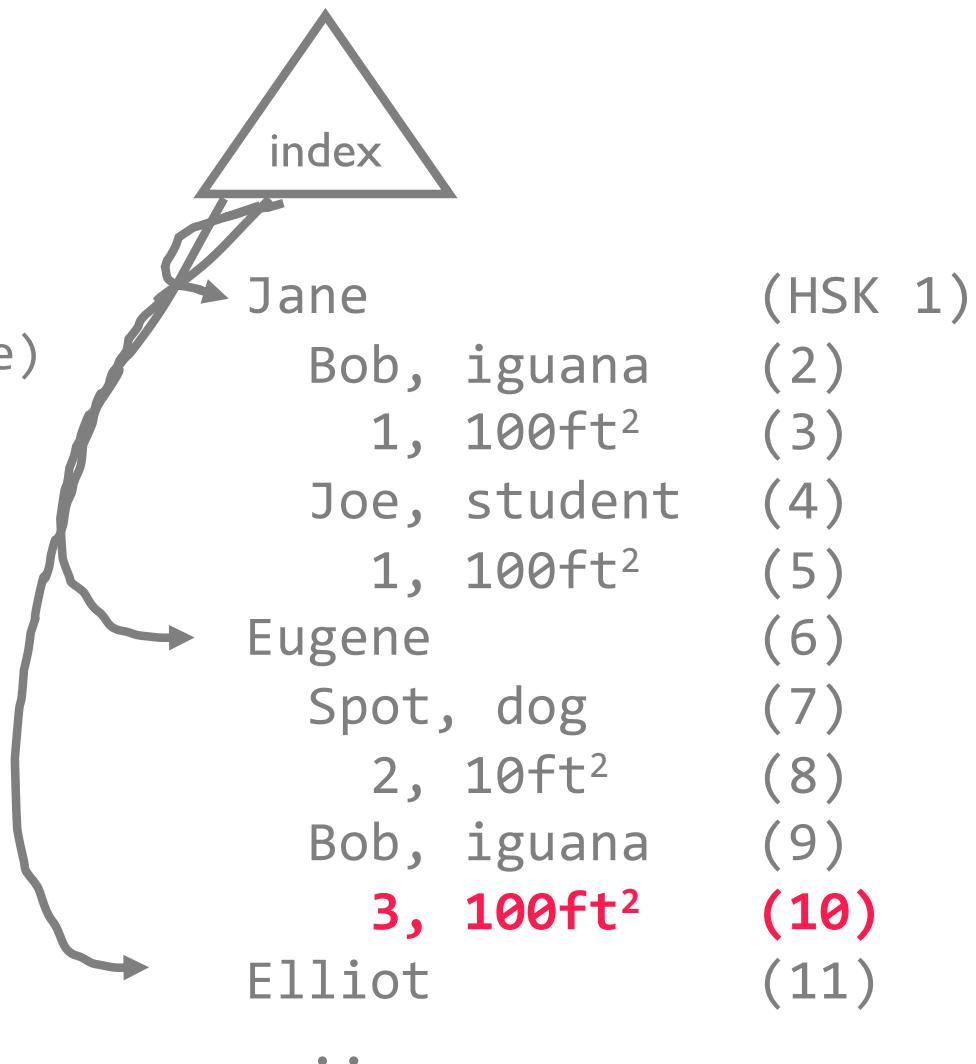
Fetch cages that Eugene manages

```
Goto(Keeper, name = Eugene)  
Until no more records  
    cage = NextInParent(Cage)  
print cage.no
```

Output:

2

3



Hierarchical “Querying”

Fetch cages that Eugene manages

Goto(Keeper, name = Eugene)

Until no more records

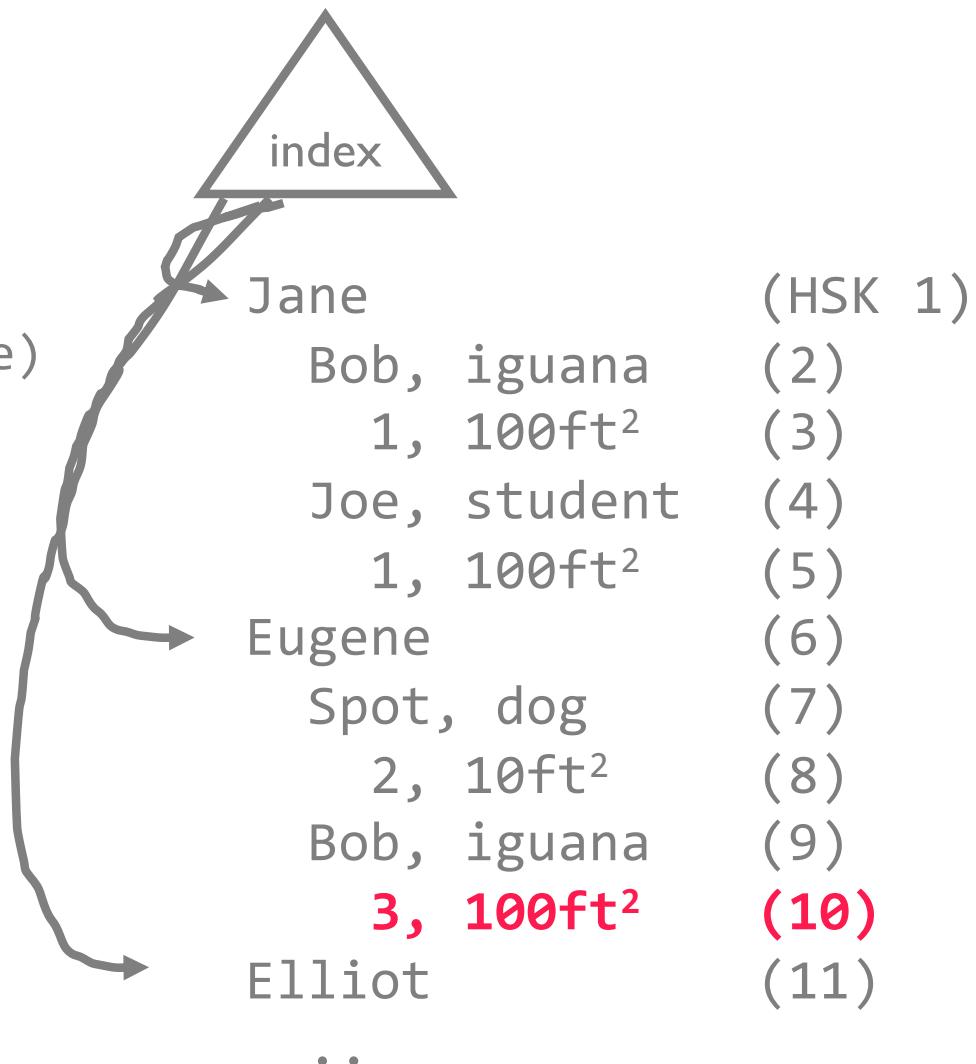
cage = NextInParent(Cage)

print cage.no

Output:

2

3



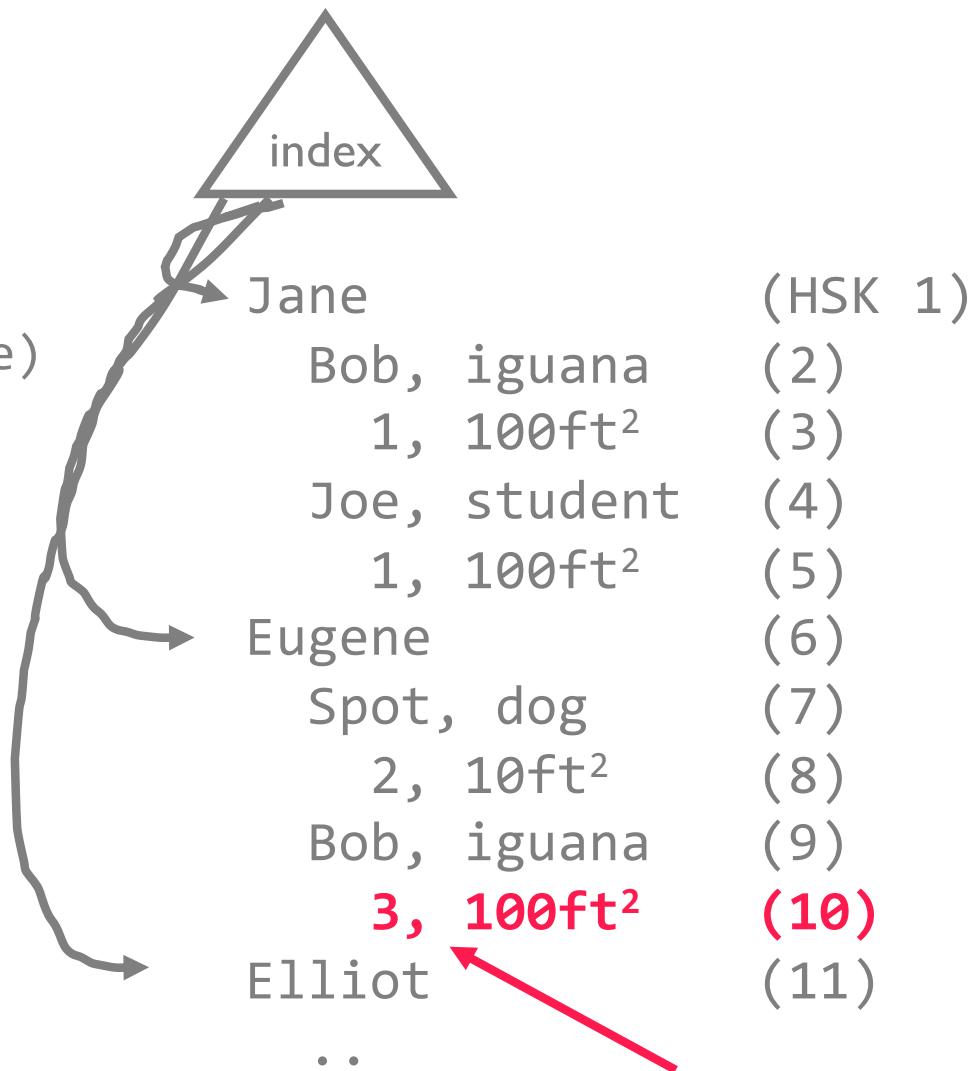
Hierarchical “Querying”

Fetch cages that Eugene manages

```
Goto(Keeper, name = Eugene)  
Until no more records  
    cage = NextInParent(Cage)  
    print cage.no
```

Output:

2
3



Notice the redundant and inconsistent data

Hierarchical “Querying”

Find Keepers of Cage 1

```
keeper = Goto(Keeper)
```

```
NextInParent(Cages, no=1)
```

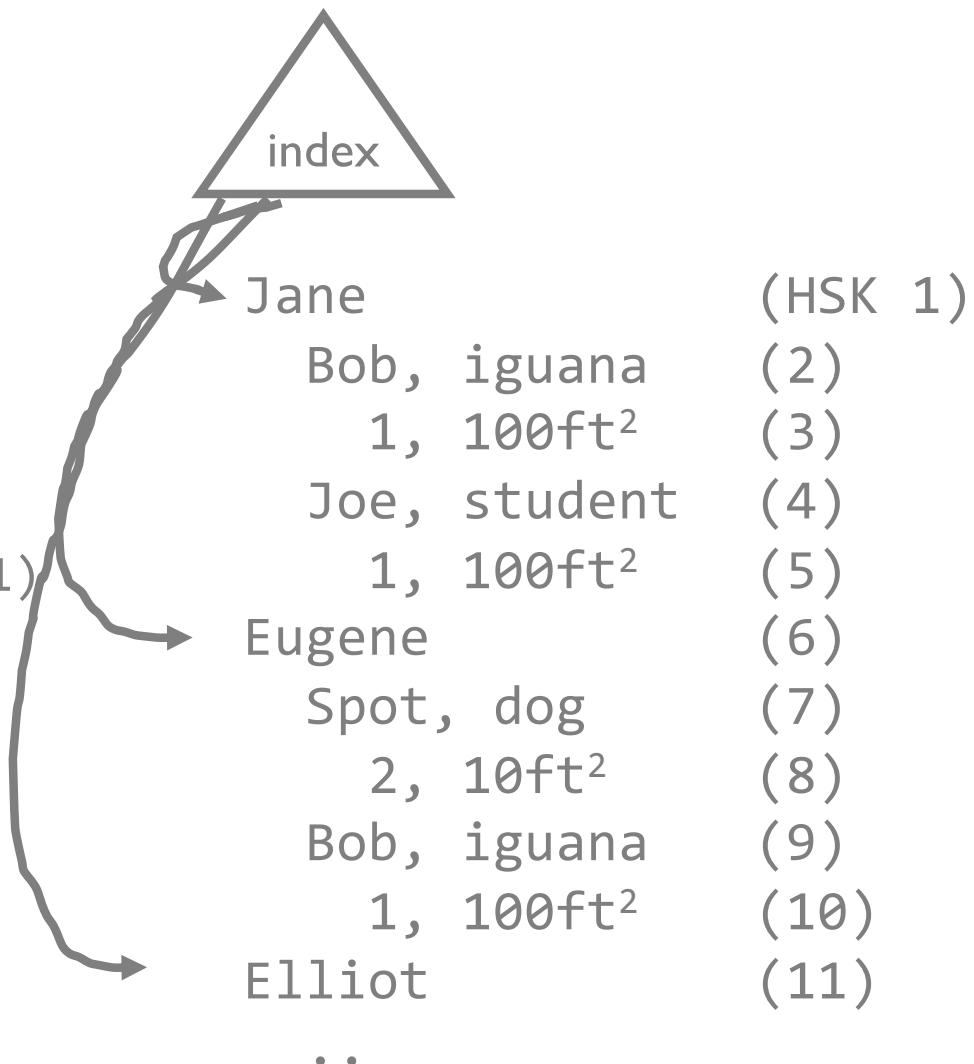
```
print keeper
```

```
Until no more records
```

```
    keeper = Next(Keeper)
```

```
    NextInParent(Cages, no=1)
```

```
    print keeper
```



Problems

Duplicates data

Low level programming interface

Almost no physical data independence

Change root from tree to hash index causes programs with GN
on root to *fail*

Inserts into sequential root structures disallowed

Lacks logical data independence

Changing schema requires changing program

Violates many desirable properties
of a proper DBMS

More Problems

Schema changes require program changes because
pointers after GetNext() calls now different

But...schemas change all the time

Keepers now responsible for a whole cage

Hummingbirds require multiple feedings

Merge with another zoo

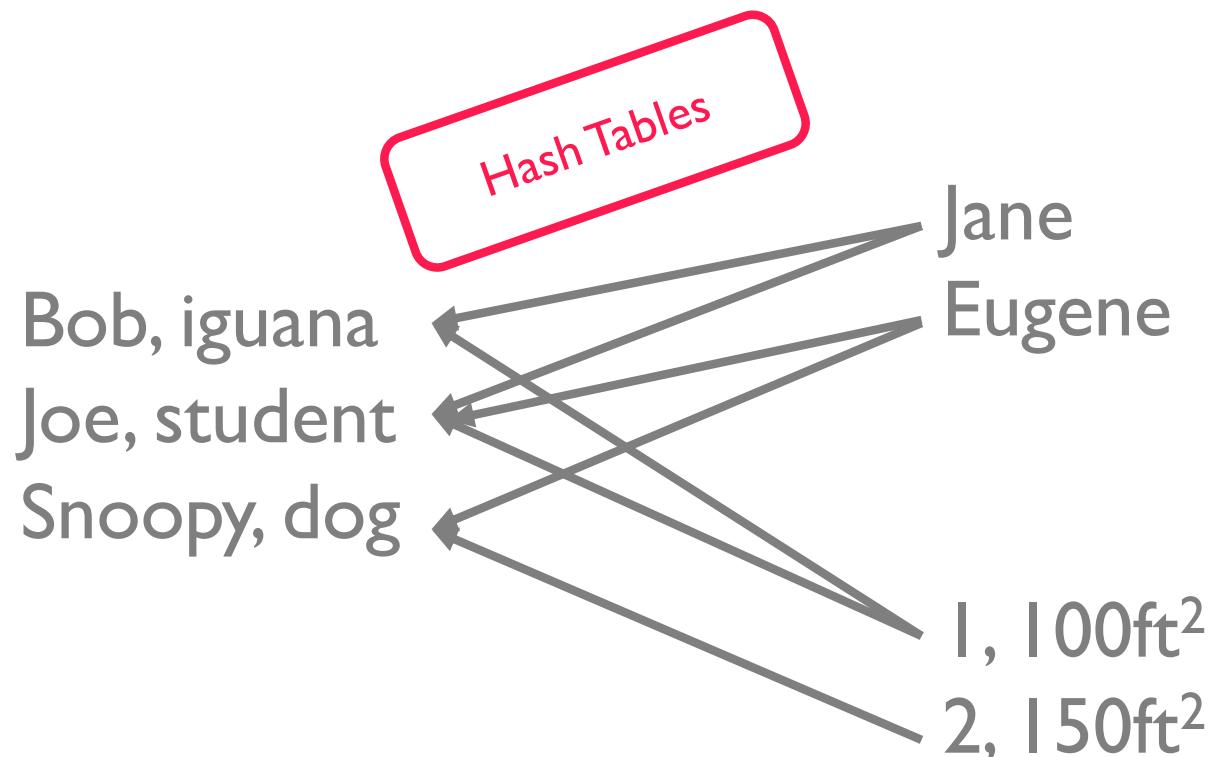
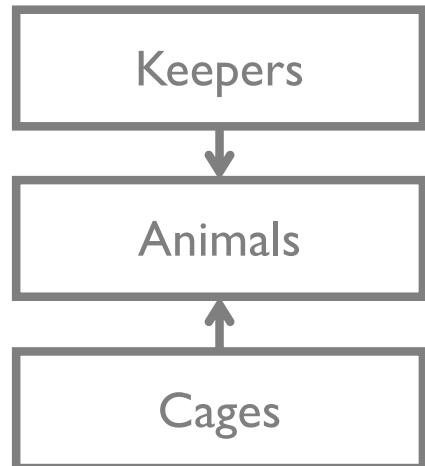
Network Models (CODASYL, 1969)

Abstraction

Types of Records

Connected by named sets (1-to-N relationships)

Modeled as a graph



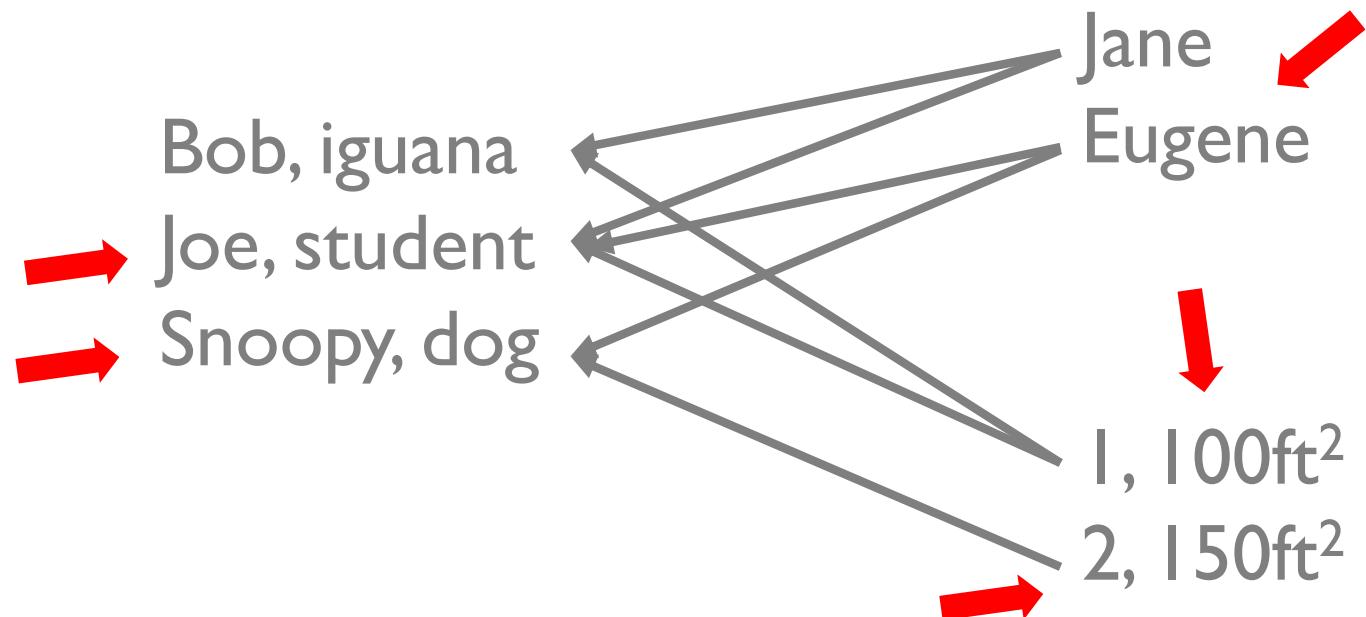
Network Models (CODASYL, 1969)

Find Keeper (name = ‘Eugene’)
until no more

 Find next Animal in cares_for

 Find Cage in lives_in

 Get current record



Network Models: Problems

Very complex due to navigational programming
(hard to hire these programmers!)

No data independence

Poor implementations e.g., main memory only

Bad trade off: increased programmer pain in order to model non-hierarchical data

Relational Model (1970)

Ted Codd, 1970

Reaction to CODASYL

Key properties

1. simple representation
2. set oriented model
3. no physical data model needed

Information Retrieval

A Relational Model of Data for
Large Shared Data Banks

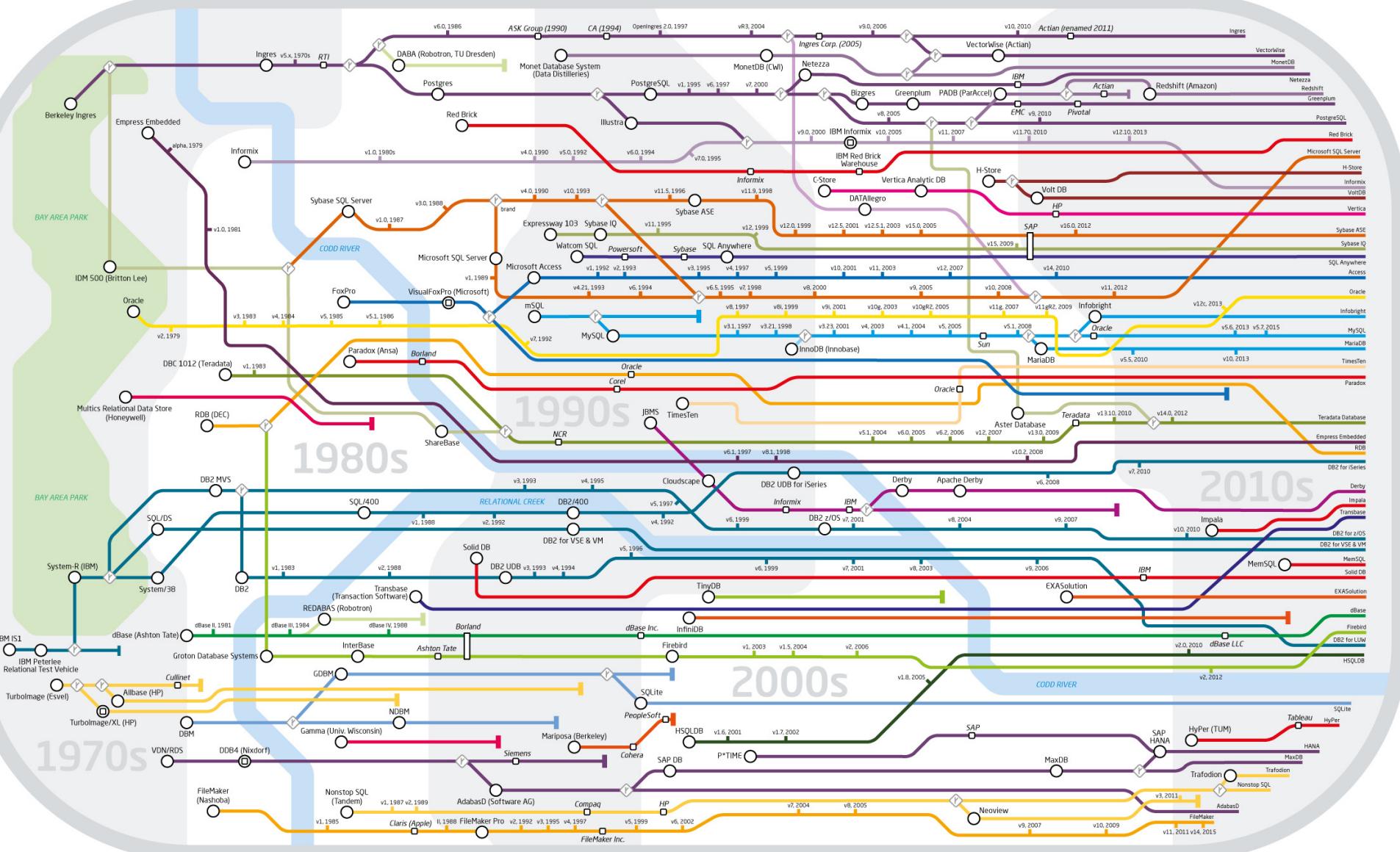
E. F. CODD

IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models

Genealogy of Relational Database Management Systems





Franck Pachot @FranckPachot · Sep 10

Replying to @MongoDB

Real friends will patiently explain to n00b friends why relational model was invented. #consistency #security #abstraction #encapsulation #normalization #joins #multi-purpose model.

seas.upenn.edu/~zives/03f/cis...

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, is presented.

Roadmap

History lesson

DDLs: Data definition language

Create schemas and constraints

DMLs: Data Manipulation Language Selection

Select, insert, delete, update data

ER → Relational Model

Basic Defs

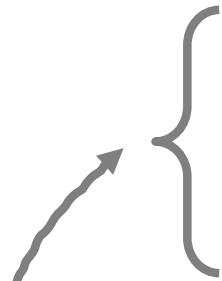
Relation defined by schema and contains set of tuples

Schema: relation name and **ordered list** of (attribute,type) pairs

an attribute/column



<u>sid</u>	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@cs	33	3.9



tuples: value for each attribute

Basic Defs

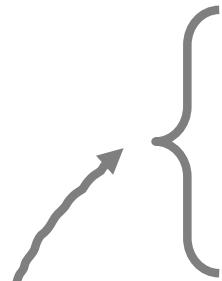
Relation defined by schema and contains set of tuples

`students(sid int, name text, login text, age int, gpa float)`

an attribute/column



<u>sid</u>	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@cs	33	3.9

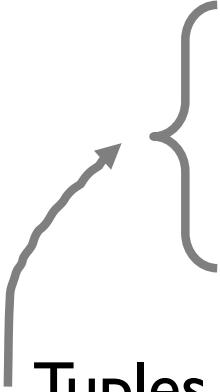


tuples: value for each attribute

Basic Defs

Relation defined by schema and contains set of tuples

Schema is metadata



<u>sid</u>	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@cs	33	3.9

Tuples are data

Basic Defs

Relation defined by schema and contains set of tuples

<u>sid</u>	name	login
1	eugene	ewu@cs
2	luis	luis@cs
3	ken	ken@cs

=?

<u>sid</u>	login	name
1	ewu@cs	eugene
2	luis@cs	luis
3	ken@cs	ken

Not Same

Basic Defs

Relation defined by schema and contains set of tuples

“Relation instance”: schema and a **set** of specific tuples
unordered, no duplicates

Database set of relations

“Database instance”: set of relation instances

sid	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@cs	33	3.9

relation instance

Can the values of two columns be the same?

Can the values of two rows be the same?

Going Back to the Examples

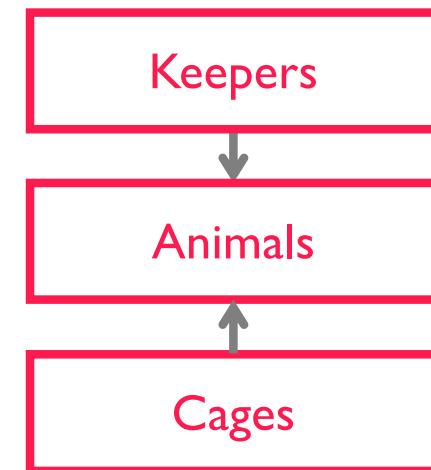
Keepers(id int, name text)

caresfor(keeper_id int, animal_id int)

Animals(id int, name text, species text)

livesin(animal_id int, cage_no)

Cages(no int, size int)



Terminology

Formal Name	Synonyms
Relation	Table
Tuple	Row, Record
Attribute	Column, Field
Domain	Type

DDL: CREATE TABLE

Create the Students Relation

Note: attribute domains are defined & enforced by DBMS

Create the Enrolled relation

```
CREATE TABLE Students(  
    sid int,  
    name text,  
    login text,  
    age int,  
    gpa float  
)
```

```
CREATE TABLE Enrolled(  
    sid int,  
    cid int,  
    grade char(2)  
)
```

Integrity Constraints (ICs)

def: a condition that is true for *any* instance of the database

Often specified when defining schema
DBMS enforces ICs at all times

An instance of a relation is **legal** if it satisfies all declared ICs
Programmer doesn't have to worry about data errors!
e.g., data entry errors

PostgreSQL documentation great resource
<https://www.postgresql.org/docs/current/ddl-constraints.html>

Domain Constraints (attr types)

```
CREATE TABLE Students(  
    sid int,  
    name text,  
    login text,  
    age int,  
    gpa float  
)
```

NULL Constraints

```
CREATE TABLE Students(  
    sid int NOT NULL,  
    name text,  
    login text,  
    age int,  
    gpa float  
)
```

Candidate Keys

Set of fields is a *candidate key (or just Key)* for a relation if:

1. No two distinct tuples have same values in all key fields
2. This is untrue for any subset of the key
3. Key fields cannot be NULL

If (2) is false, called a *superkey* what's a trivial superkey?

If >1 candidate keys in relation, DBA picks one as *primary key*

sid is key for Students

is name a key?

what is (sid, gpa)?

Primary and Candidate Keys

UNIQUE & PRIMARY KEY key words

Be careful with ICs:

Each student can enroll in
a course only once

What does this say?

```
CREATE TABLE Enrolled(  
    sid: int,  
    cid: int,  
    grade: char(2),  
    PRIMARY KEY (sid, cid)  
)
```

```
CREATE TABLE Enrolled(  
    sid: int,  
    cid: int,  
    grade: char(2),  
    PRIMARY KEY (sid),  
    UNIQUE (cid, grade)  
)
```

Primary and Candidate Keys

UNIQUE + NOT NULL == field is a key

PRIMARY KEY is shorthand for UNIQUE NOT NULL

```
CREATE TABLE Students(  
    sid int NOT NULL UNIQUE,  
    name text,  
    login text,  
    age int,  
    gpa float,  
    PRIMARY KEY (sid)  
)
```

Default Candidate Keys

sqlite, duckdb, oracle

rowid: “hidden” unique id column

postgreSQL

ctid: location of tuple. May change arbitrarily!

Primary Keys IRL

rt peteskomoroch Retweeted



Jay Kreps @jaykreps · 2h

100x this. You can't build a reliable system without a primary key for your users. If you are presenting a utility bill as a form of ID something is completely broken. No real privacy is gained by this (the government has you on a list), just inefficiency.

 **The Economist**  @TheEconomist · 4h

Britain and America have long resisted introducing a national identity system. The pandemic has shown why that is a mistake
econ.trib.al/CPGp5pi

4

5

25

↑

▼

Foreign Keys

def: set of fields in Relation R_i used to refer to tuple in R_j via R_j 's keys (a logical pointer)

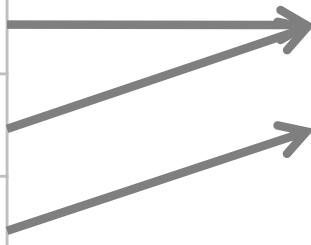
```
CREATE TABLE Enrolled(  
    sid: int, cid: int, grade: char(2),  
    PRIMARY KEY (sid, cid),  
    FOREIGN KEY (sid) REFERENCES Students(sid)  
)
```

Enrolled

sid	cid	grade
1	2	A
1	3	B
2	2	A+

Students

sid	name
1	eugene
2	luis



Foreign Keys

def: set of fields in Relation R_i used to refer to tuple in R_j via R_j 's keys (a logical pointer)

```
CREATE TABLE Donation(
    companyid int,
    classid int,
    amount float,
    PRIMARY KEY(companyid, classid)
)
CREATE TABLE Donation_Admin(
    daid int PRIMARY KEY,
    companyid int,
    classid int,
    FOREIGN KEY (companyid, classid) REFERENCES
        Donation(companyid, classid)
)
```

Referential Integrity

A database instance has *referential integrity* if all foreign key constraints are enforced no dangling references

What are where referential integrity is not enforced?

HTML links

Yellow page listing

Restaurant menus

CHECK Constraints

Boolean constraint expression added to schema

Very powerful mechanism.

Evaluated over each tuple, rejects tuple if false

```
CREATE TABLE Enrolled(  
    sid int,  
    cid int,  
    grade char(2),  
    CHECK (  
        grade IN ('A', 'B', 'C', 'D', 'F')  
    )  
)
```

How to Enforce Integrity Constraints

Run checks anytime database changes

On INSERT

What if new Enrolled tuple refers to non-existent student?

Reject insertion

On DELETE (many options)

What if Students tuple is deleted?

Delete dependent Enrolled tuples

Reject deletion

Set Enrolled.sid to default value or **null**

(null means ‘unknown’ or ‘inapplicable’ in SQL)

Where do ICs come from?

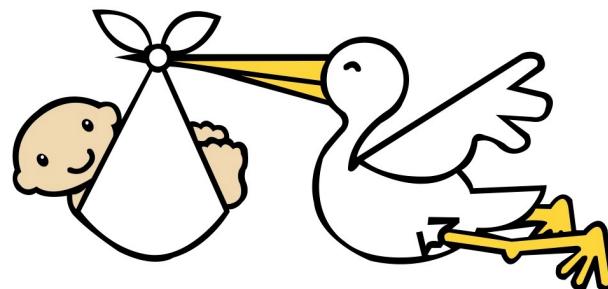
Based on application semantics and use cases

Can check if database instance satisfies ICs

IC is a statement over *all possible instances*

Can't infer ICs by staring at an instance

Key and foreign key ICs are most common, more general table and database constraints possible as well.



(mostly) More Powerful than ER Constraints

Functional dependencies

A dept can't order two distinct parts from the same supplier.

Can't express this wrt ternary Contracts relationship.

Used to further refine ER models (called normalization)

Inclusion dependencies

Special case: ER model can express Foreign keys

At least 1 person must report to each manager.

General constraints

Each donation is less than 10% of the combined donations to all humanities courses. (but can be expensive to enforce)

What Can ER Express?

Key constraints,
participation constraints,
overlap/covering constraints

Some foreign key constraints as part of relationship set
Some constraints require general CHECK stmts
ER cannot express e.g., function dependencies at all
Constraints help determine best database design

Roadmap

History lesson

DDLs: Data definition language

Create schemas and constraints

DMLs: Data Manipulation Language Selection

Select, insert, delete, update data

ER → Relational Model

DML: Introduction to Queries

Key strength of relational model
declarative querying of data

Queries are high level, readable
DBMS makes it fast, user don't need to worry

Precise semantics for relational queries
Lets DBMS choose different ways to run query while
ensuring answer is the same

INSERT/DELETE

Add a tuple

```
INSERT INTO Students(sid, name, login, age, gpa)  
VALUES (4, 'wu', 'wu@cs', 20, 5)
```

Delete tuples satisfying a predicate (condition)

```
DELETE FROM Students S  
WHERE S.name = 'wu'
```



Boolean predicate

Bulk Imports

varies by database system, read their docs

Postgresql

```
COPY into Students  
FROM '/users/ewu/students.csv' WITH CSV
```

SQLite

```
.mode csv  
.import /users/ewu/students.csv Students
```

DuckDB

```
CREATE TABLE Students AS  
SELECT * FROM read_csv_auto('/users/ewu/students.csv')
```

Basic SELECT

Get all attributes of
students younger than 21

```
SELECT *  
FROM Students S  
WHERE S.age < 21
```

Get only names

```
SELECT S.name  
FROM Students S  
WHERE S.age < 21
```

sid	name	login	age	gpa
1	eugene	ewu@cs	20	2.5
2	luis	luis@cs	20	3.5
3	ken	ken@math	33	3.9

Single Table Semantics

A *conceptual evaluation method* for previous query:

```
for row in Students           // FROM clause
  if predicate(row):          // WHERE clause
    emit desired fields       // SELECT clause
```

Actual evaluation will be *much* more efficient, but must produce the same answers.

Multi-table SELECT

What does this return?

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid = E.sid AND  
E.grade = "A"
```

Enrolled

sid	cid	grade
1	2	A
1	3	B
2	2	A+

Students

sid	name
1	eugene
2	luis

Result

name	cid
eugene	2

Multi-Table Semantics

Modify the FROM clause evaluation

I. FROM clause: compute *cross-product* of Students and Enrolled

Enrolled

sid	cid	grade
1	2	A
1	3	B
2	2	A+

Cross-product

sid	cid	grade	sid	name
1	2	A	1	eugene
1	3	B	1	eugene
2	2	A+	1	eugene
1	2	A	2	luis
1	3	B	2	luis
2	2	A+	2	luis

Students

sid	name
1	eugene
2	luis

Multi-Table Semantics

Modify the FROM clause evaluation

1. FROM clause: compute *cross-product* of Students and Enrolled
2. WHERE clause: check conditions, discard tuples that fail
3. SELECT clause: keep desired fields

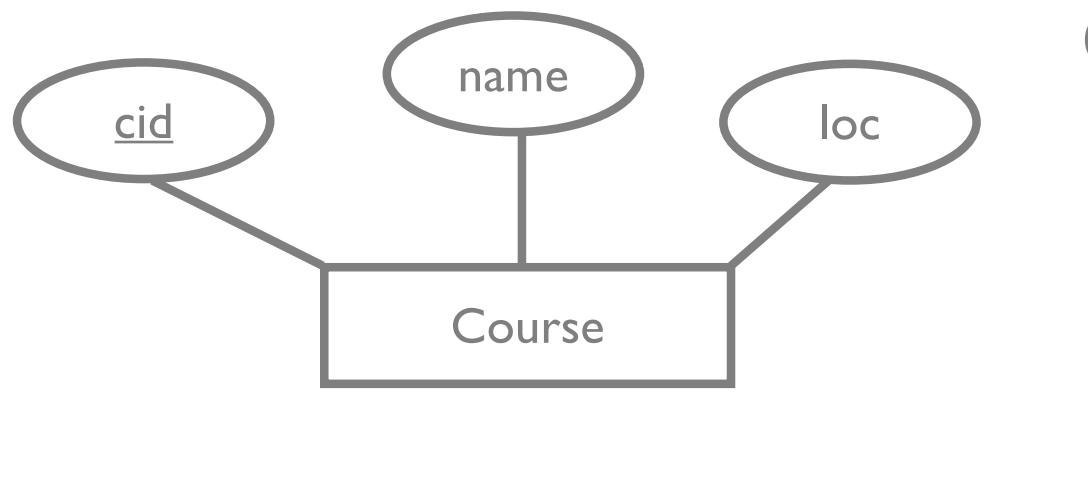
Multi-Table Semantics

```
SELECT a1, a2, a3  
FROM A, B, C  
WHERE A.a1 = B.b1
```

```
for row1 in A  
  for row2 in B  
    for row3 in C  
      if (row1.a1 == row2.b1) {  
        yield [row1.a1, row1.a2, row1.a3]  
      }
```

Translating ER → Relational Models

Entity Set → Relation



```
CREATE TABLE Course(  
    cid int,  
    name text,  
    loc text,  
    PRIMARY KEY (cid)  
)
```

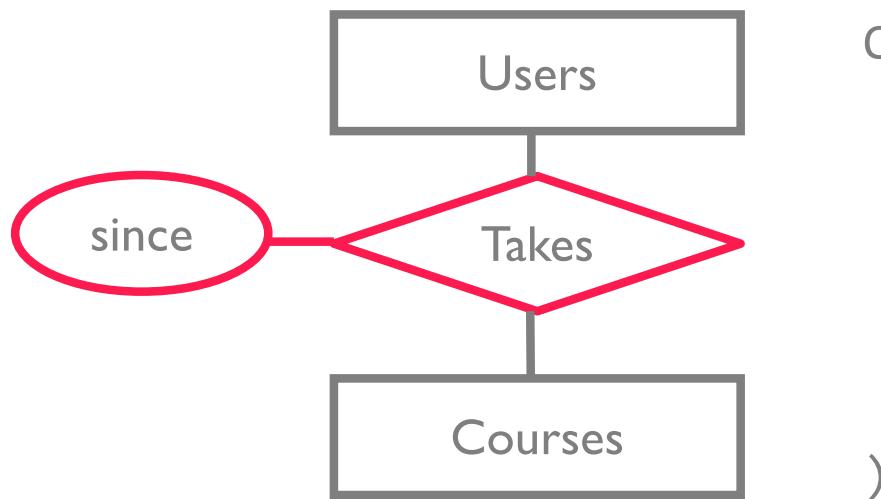
Relationship Set w/out constraint → Relation

Relation must include

Keys for each entity set as foreign keys

these form superkey for relation

All attributes of the relationship set



```
CREATE TABLE Takes(  
    uid int,  
    cid int,  
    since date,  
    PRIMARY KEY (uid, cid),  
    FOREIGN KEY (uid) REFERENCES Users,  
    FOREIGN KEY (cid) REFERENCES Courses  
)
```

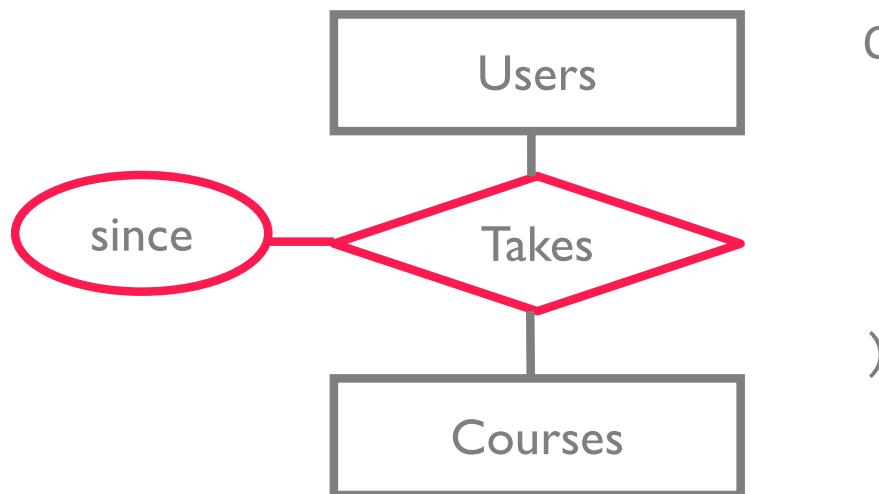
Relationship Set w/out constraint → Relation

Relation must include

Keys for each entity set as foreign keys

these form superkey for relation

All attributes of the relationship set



```
CREATE TABLE Takes(  
    uid int REFERENCES Users(uid),  
    cid int REFERENCES Courses(cid),  
    since date,  
    PRIMARY KEY (uid, cid)  
)
```

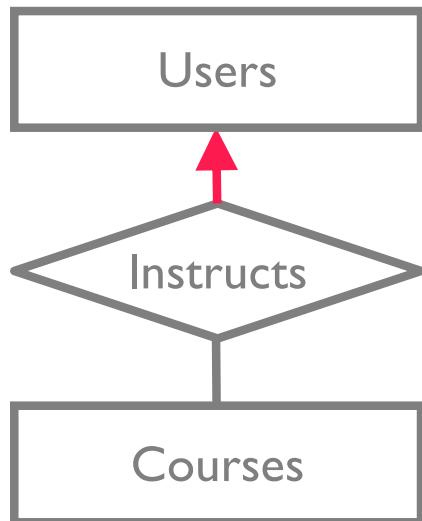
Short hand

Key Constraint → Relation

Note only cid is a Key

Why don't we have UNIQUE(uid, cid)?

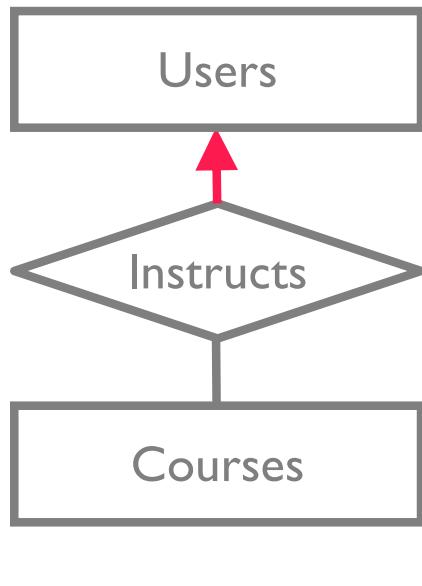
User and Courses are separate relations



```
CREATE TABLE Instructs(  
    uid int NOT NULL,  
    cid int,  
    PRIMARY KEY (cid),  
    FOREIGN KEY (uid) REFERENCES Users,  
    FOREIGN KEY (cid) REFERENCES Courses  
)
```

Key Constraint → Relation

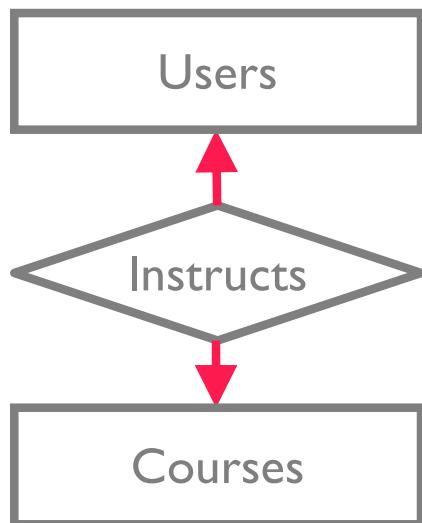
Alternatively combine Courses and Users
(this is the preferred way)



```
CREATE TABLE Course_Instructs(  
    cid int  
    uid int,  
    name text,  
    loc text,  
    PRIMARY KEY (cid),  
    FOREIGN KEY (uid) REFERENCES Users  
)
```

Key Constraint → Relation

How to translate this ER diagram?

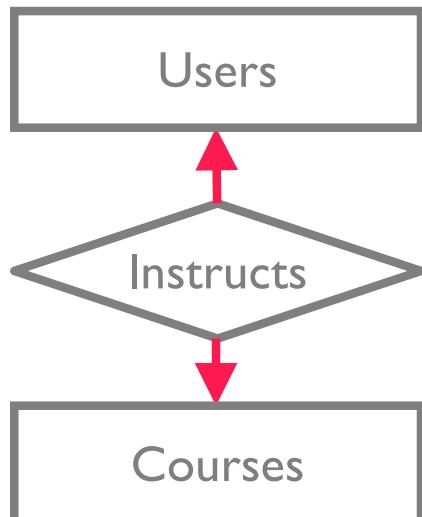


```
CREATE TABLE Course_Instructs(  
    ????  
)
```

Key Constraint → Relation

Translation I.

Only merge Courses and Instructs

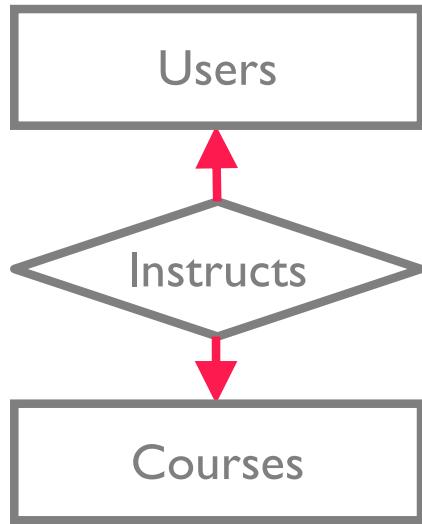


```
CREATE TABLE Course_Instructs(  
    cid int,  
    uid int UNIQUE NOT NULL,  
    name text,  
    loc text,  
    PRIMARY KEY (cid),  
    FOREIGN KEY (uid) REFERENCES Users  
)
```

Key Constraint → Relation

Translation 2.

Merge all three relations.



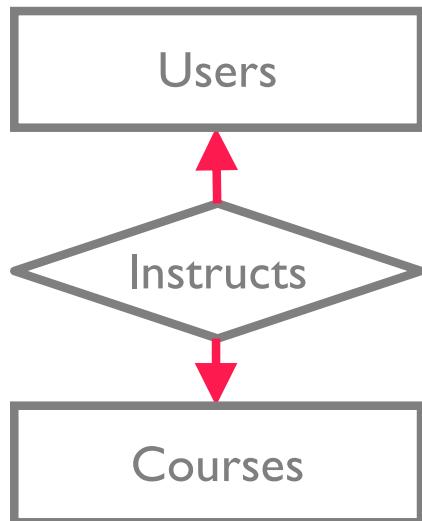
```
CREATE TABLE Course_Instructs_Users(  
    cid int  
    uid int,  
    name text,  
    loc text,  
    username text,  
    age text,  
    PRIMARY KEY (cid, uid)  
)
```

This can get ugly

Key Constraint → Relation

Translation 3.

Keep all three relations.



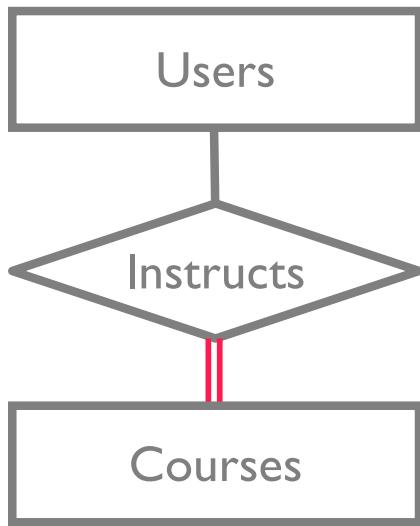
```
CREATE TABLE courses(cid, ...)  
CREATE TABLE users(uid, ...)
```

```
CREATE TABLE instructs(  
    uid int UNIQUE NOT NULL,  
    cid int UNIQUE NOT NULL,  
    PRIMARY KEY (uid, cid)  
)
```

IF there is a relationship, it must satisfy constraints in instructs

Participation Constraint → Relation

We only consider participation constraints with one entity set in binary relationship (others need CHECK constraint)

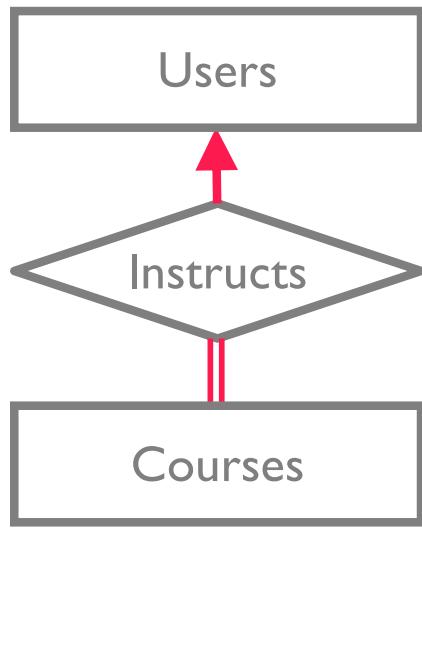


```
CREATE TABLE Course_Instructs(  
    cid int  
    uid int NOT NULL,  
    name text,  
    loc text,  
    PRIMARY KEY (cid),  
    FOREIGN KEY (uid) REFERENCES Users  
        ON DELETE NO ACTION  
)
```

Not a fully correct translation. Why?

Participation Constraint → Relation

Actually expresses exactly 1 relationship



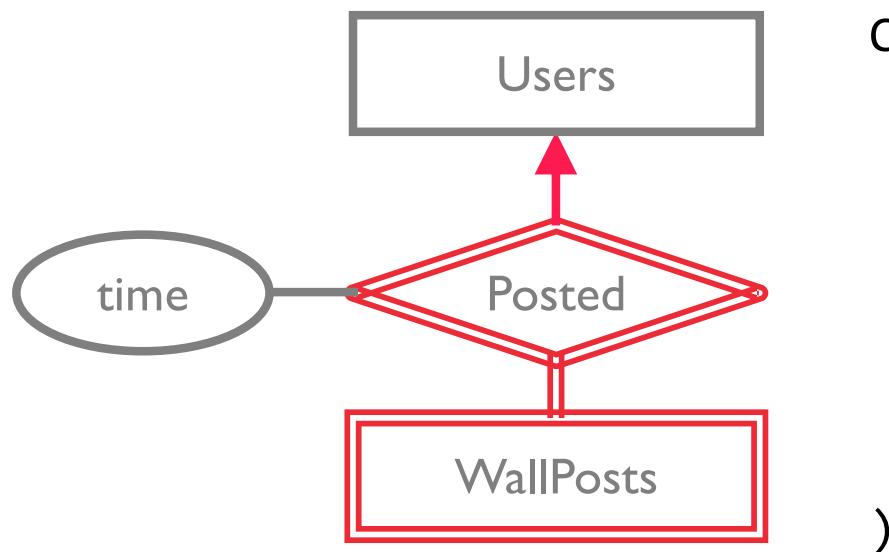
```
CREATE TABLE Course_Instructs(  
    cid int  
    uid int NOT NULL,  
    name text,  
    loc text,  
    PRIMARY KEY (cid),  
    FOREIGN KEY (uid) REFERENCES Users  
        ON DELETE NO ACTION  
)
```

Weak Entity → Relation

Weak entity set and identifying relationship set are translated into a single table.

When owner entity is deleted, all owned weak entities also deleted.

pid in this example is a partial key: it alone does not identify a given wall post! For example, if each user's first wall post is given pid 1, then pid does not uniquely identify a wall post since another user's first post may also be pid 1



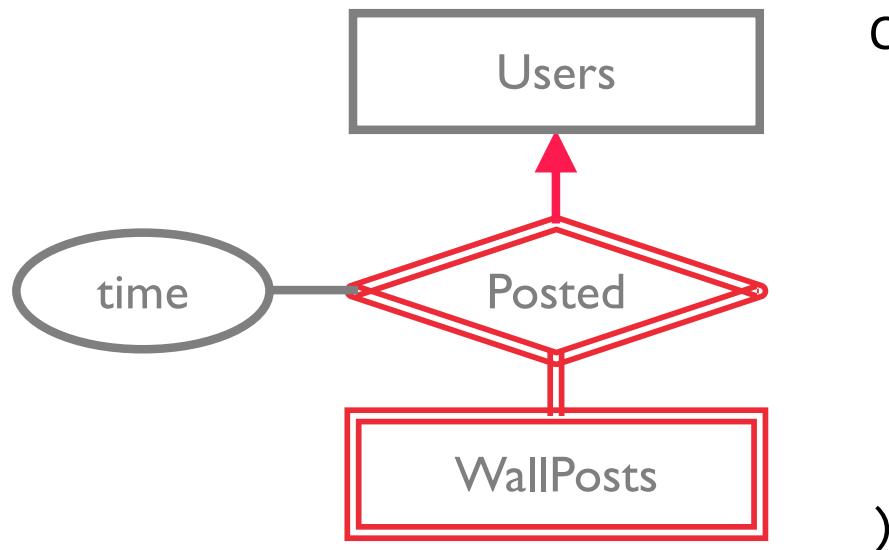
```
CREATE TABLE WallPosts_Posted(  
    pid int,  
    post_text text,  
    posted_at DATE,  
    uid int NOT NULL,  
    PRIMARY KEY (pid, uid),  
    FOREIGN KEY (uid) REFERENCES Users  
        ON DELETE CASCADE  
)
```

Weak Entity → Relation

Weak entity set and identifying relationship set are translated into a single table.

When owner entity is deleted, all owned weak entities also deleted.

In practice, pid would be serial (auto-generated and unique), so the primary key could simply be pid. It is still weak due to ON DELETE CASCADE



```
CREATE TABLE Wall_Posted(  
    pid int serial,  
    post_text text,  
    posted_at DATE,  
    uid int NOT NULL,  
    PRIMARY KEY (pid),  
    FOREIGN KEY (uid) REFERENCES Users  
    ON DELETE CASCADE  
)
```

Specialization Hierarchies

Option 1: Keep parent relation

B and C's attributes stored in A

Option 2: Keep parent and child relations

A stores only attributes of A

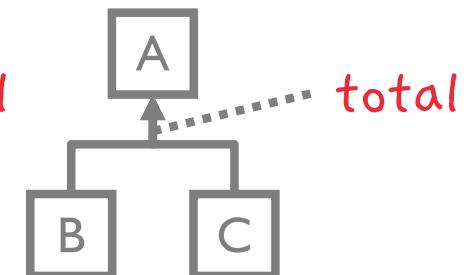
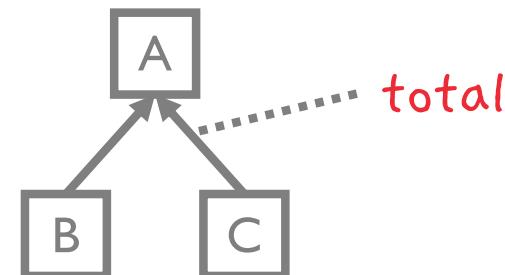
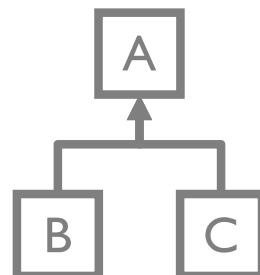
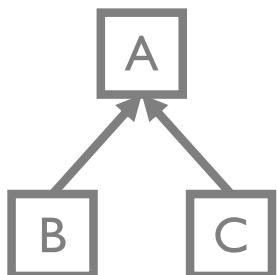
B and C store their own attributes

JOIN between child and base relations for all attributes

Option 3: Only keep child relations

B and C have copies of A's attributes

} Only under total specialization!



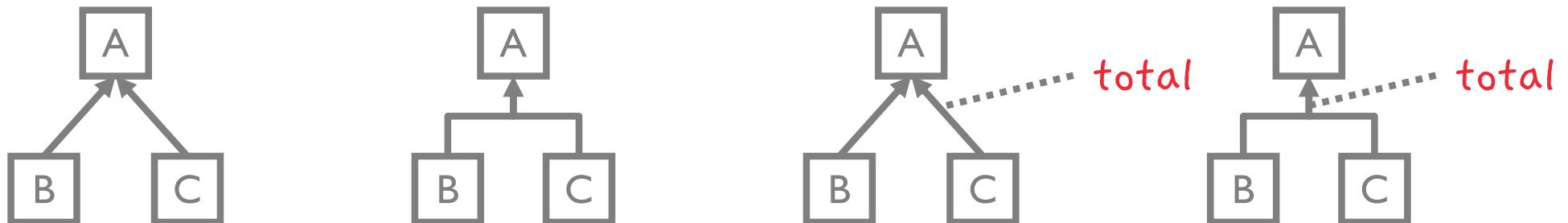
Specialization Hierarchies: Option I

A(a int PRIMARY KEY, name text)

B(b int PRIMARY KEY, a int REFERENCES A)

C(c int PRIMARY KEY, a int REFERENCES A)

Supports overlap = true, total = no



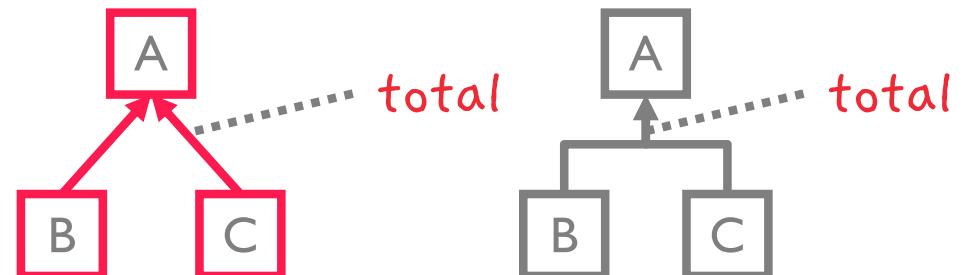
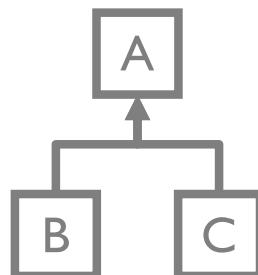
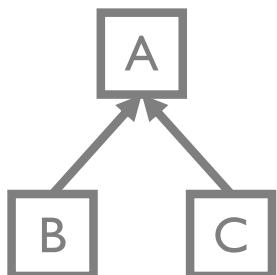
Specialization Hierarchies: Option 2

B(id int PRIMARY KEY, name text)

C(id int PRIMARY KEY, name text)

Enforces total specialization.

Can't have relationship with As in general



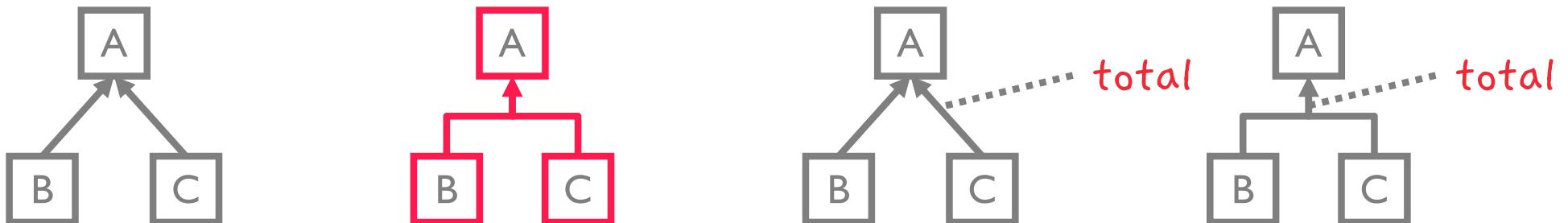
Specialization Hierarchies: Option I

A(

```
a int PRIMARY KEY,  
name text,  
b int,  
c int,  
CHECK (not (b is not null and c is not null))
```

)

Supports overlap = no, total = no



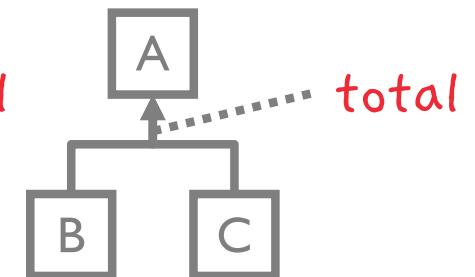
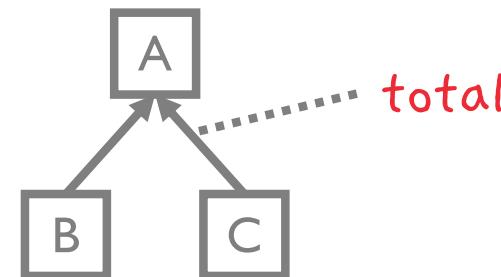
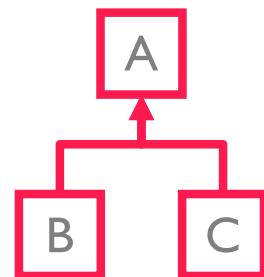
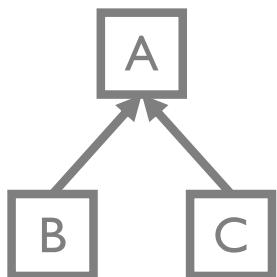
Specialization Hierarchies: Option I

A(

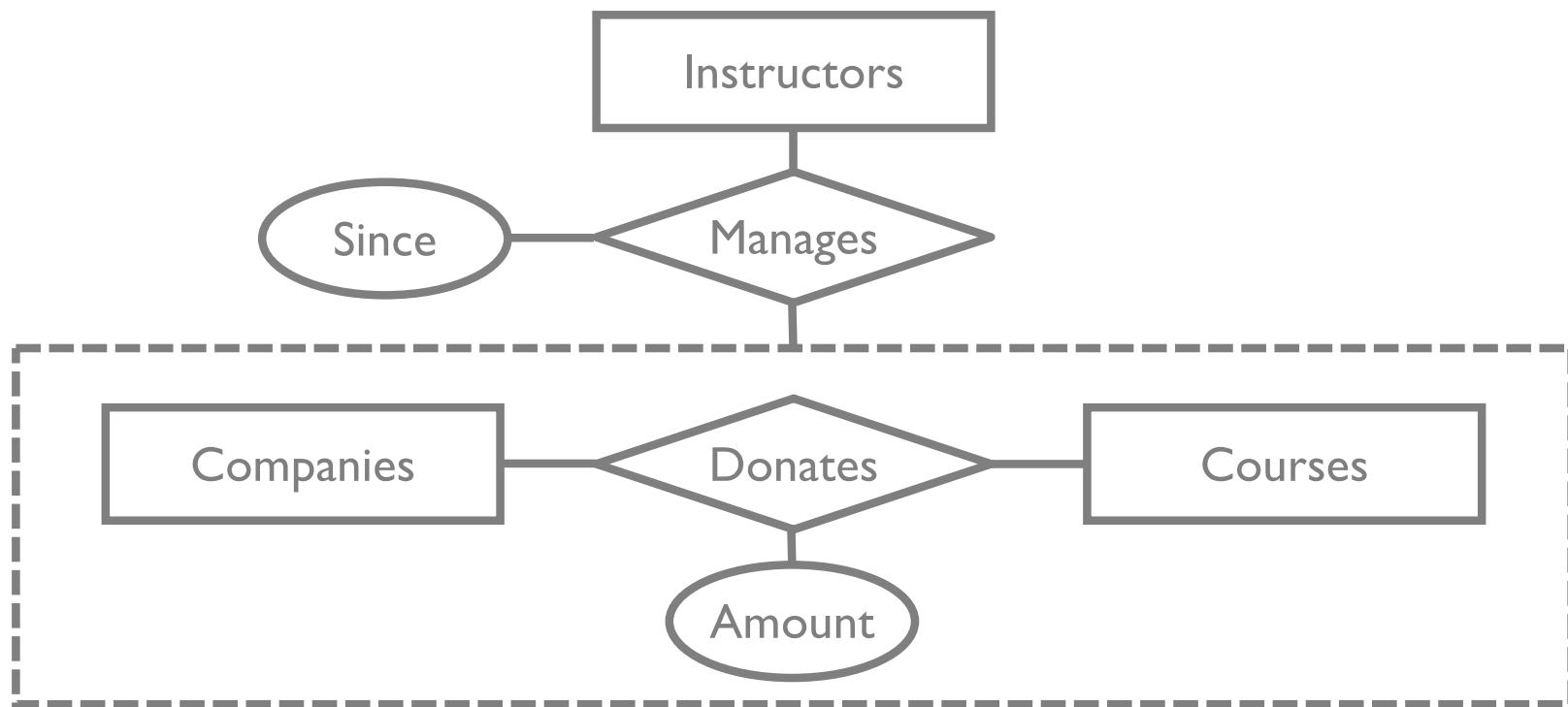
```
a int PRIMARY KEY,  
name text,  
b int,  
c int,  
CHECK (b is null or c is null) and  
(b is not null or c is not null)
```

)

Supports overlap = no, total = yes

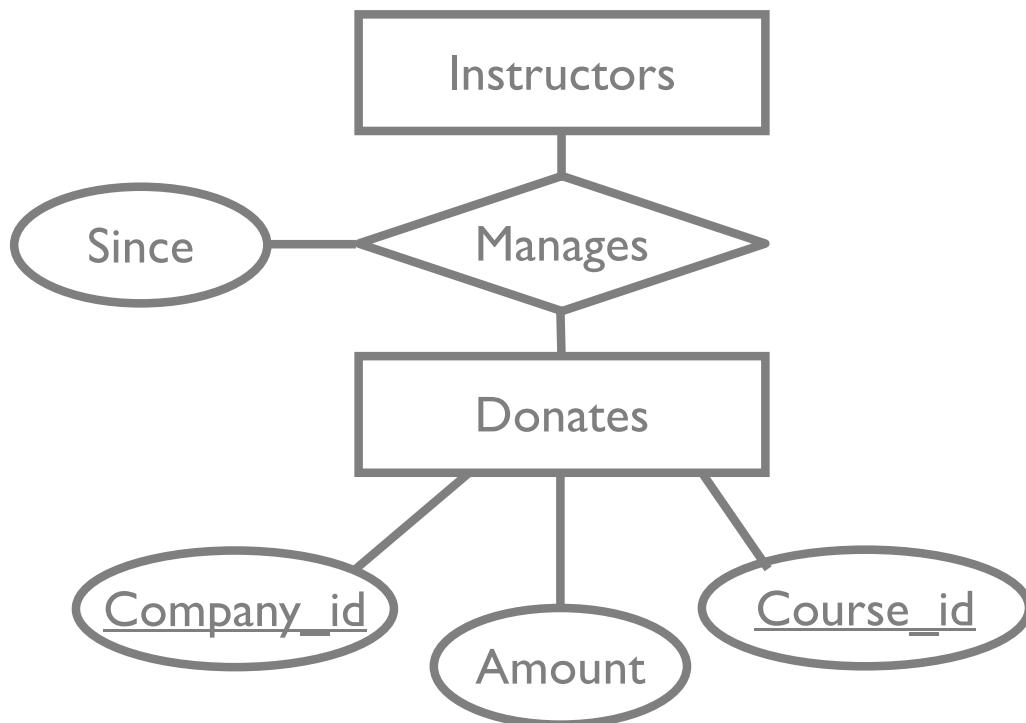


Aggregation



Aggregation

However you translate the donates relationship set is the table you use to model the relationship with instructors.



Demo!

Other Popular Data Models

Matrix

TYPE is a field (+, * obey “usual algebra”)

Rows and Cols are ordered and indexed

Values can be missing.

Lots of missing values is “sparse”. Else dense

TYPE	1	...	n
1			
...			
m			

Other Popular Data Models

Matrix: expressible as relation?

```
Matrix(  
    i int, j int,  
    v TYPE,  
    primary key (i, j)  
)
```

TYPE	1	...	n
1			
...			
m			

Other Popular Data Models

Relation expressible as matrix?

- ML expects matrices.
- Turn attribute types into field (e.g., a number)
- Hot-one encoding, featurization

Other Popular Data Models

Value

TYPE	1
1	

Vector

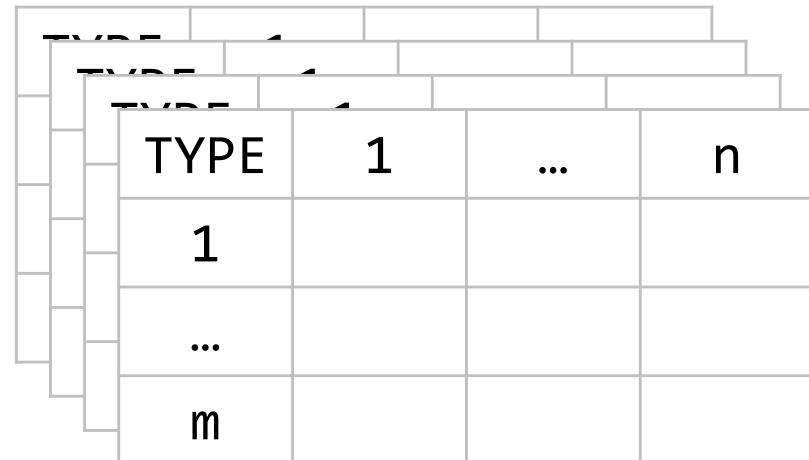
TYPE	1	2	...	n
1				

Matrix

TYPE	1	...	n
1			
...			
m			

Tensor

TYPE	1	...	n
1			
...			
m			



Other Popular Data Models

Pandas/R data frame

only columns have types

rows and cols are ordered

less mathematically grounded

	name ₁	name ₂	...	name _n
	type ₁	type ₂	...	type _n
name ₁				
...				
name _m				

REVIEW OF ER AND RELATIONAL

ER Overview

Ternary relationships

Relationships constraints

At most one

At least one

Exactly one

Weak entities

Aggregation

Is-A

Relational Review

Relations ~= Sets

Schema = structure // like a class definition

Instance = the data // like an object

"every relation is guaranteed to have a key"?

Integrity Constraints

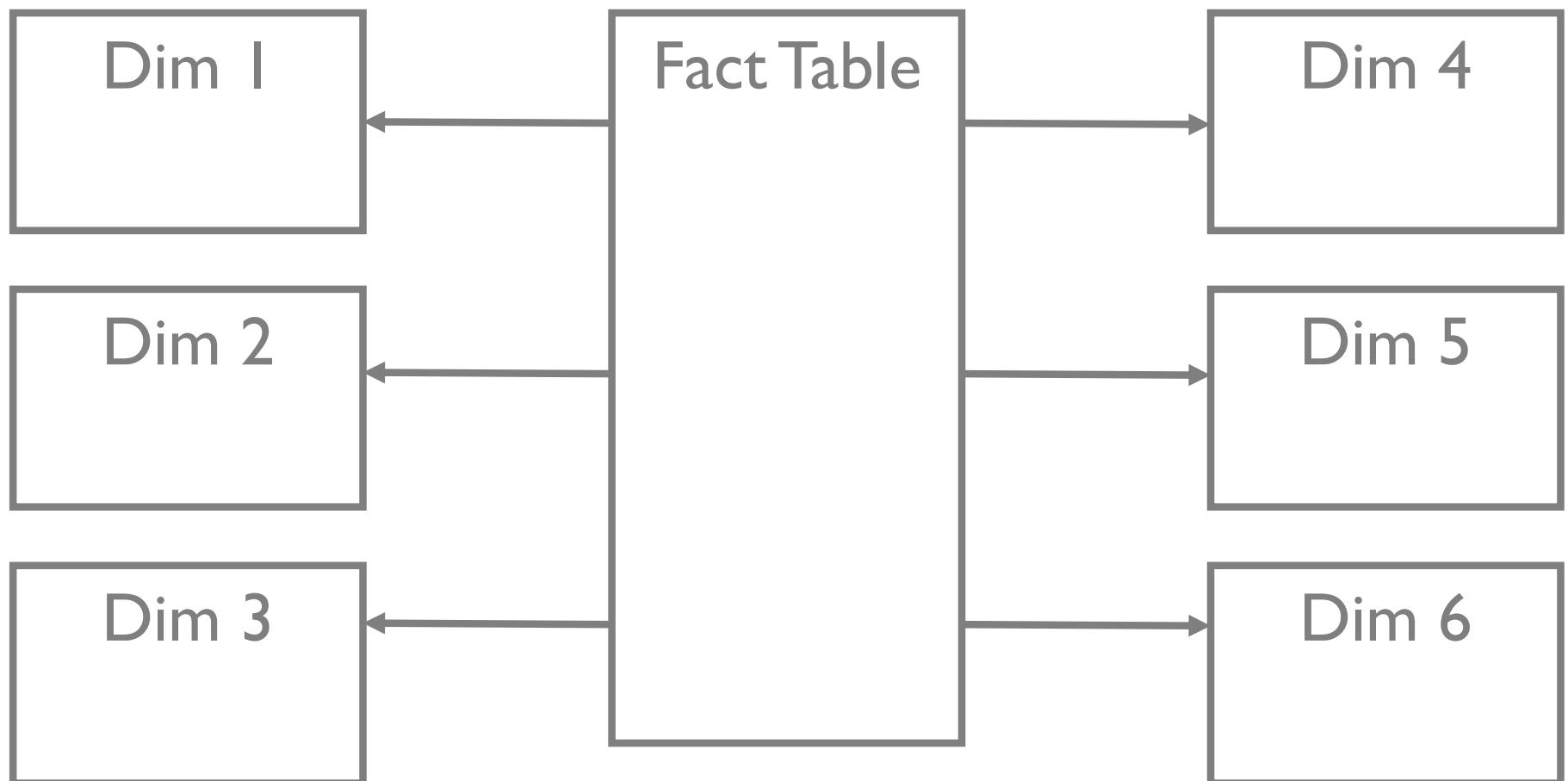
Candidate and primary keys

NOT NULL

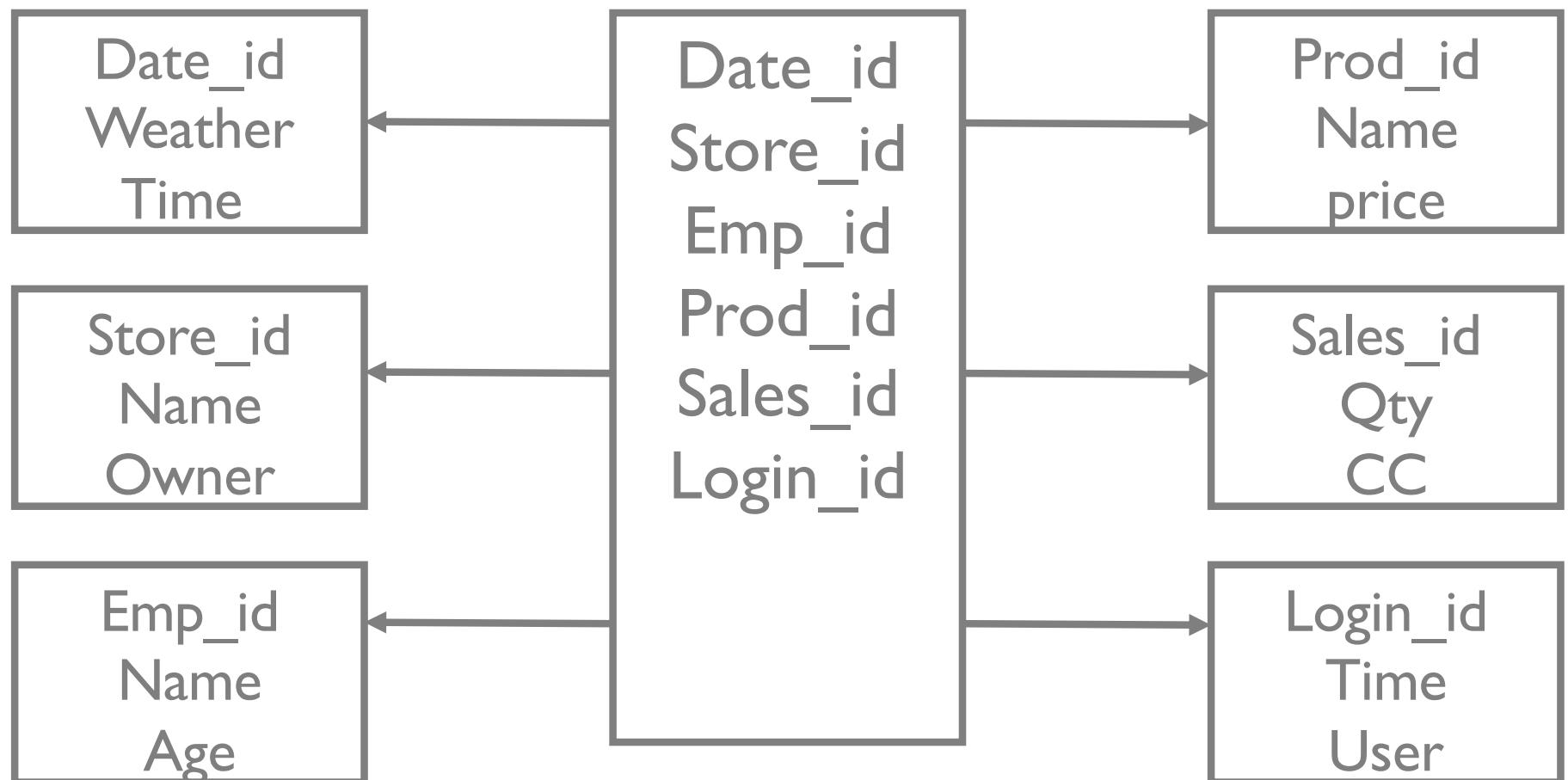
Referential integrity

How are foreign keys managed by the DBMS?

Star Schema



Star Schema



So What Happened?

1970 heated debates about CODASYL vs Relational Network arguments

- low level languages more efficient (performance)

- relational queries would never be fast (performance)

Relational arguments

- data independence

- high level *simpler* languages

Market spoke.

Other models beyond relational!

Summary

Better than IMS/CODASYL

- allows us to talk about constraints!

- allows us to talk at a logical level

- declarative queries better than navigational programs

Everything is a relation (table)

DBA specifies ICs based on app, DBMS enforces

- Primary and Foreign Keys most used

- Types == Domain constraints

SQL

Next Time

Relational Algebra

A set-oriented theory for relational data

Finish history lesson