L3
The Relational Model

Eugene Wu

Fall 2015

Background

Most widely used data model in the world

Legacy Models

- IMS hierarchical
- CODASYL network

Recently popular: NoSQL

- attempt at a flexible model

Key Principles

Data redundancy (or how to avoid it)

Physical data independence

- programs don't worry about physical structure

Logical data independence

- logical structure can change and legacy programs can continue to run

High level languages

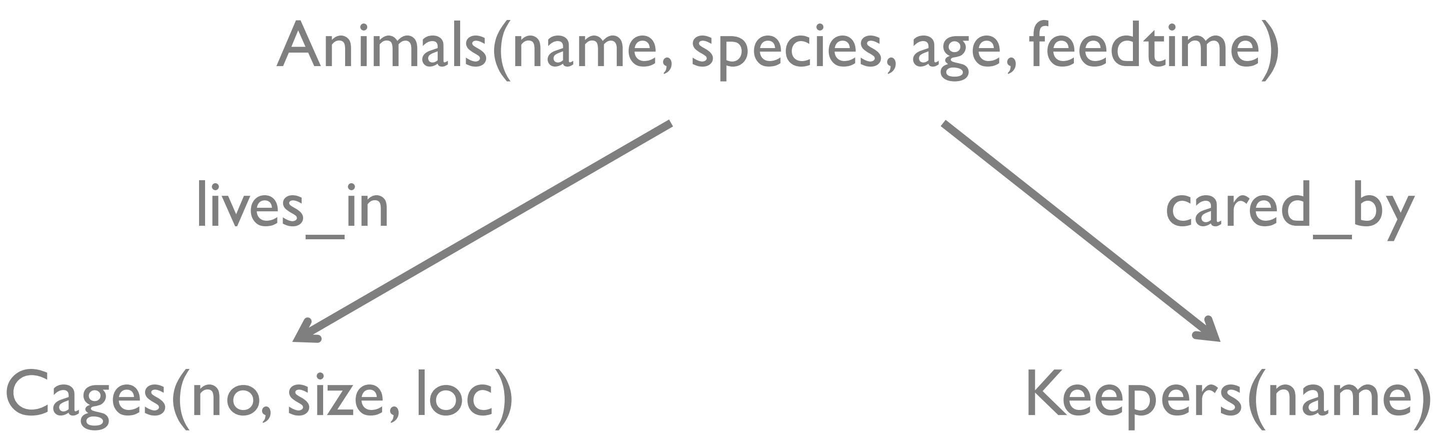
Historical Context (not on test)

Hierarchical model (IMS)
Network model (CODASYL)

70s

Relational model (SQL/QUEL)

80-90s

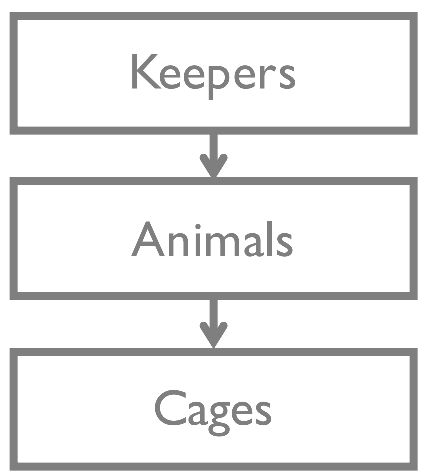


Hierarchical Model (IMS, circa 1968)

Segment types (objects)

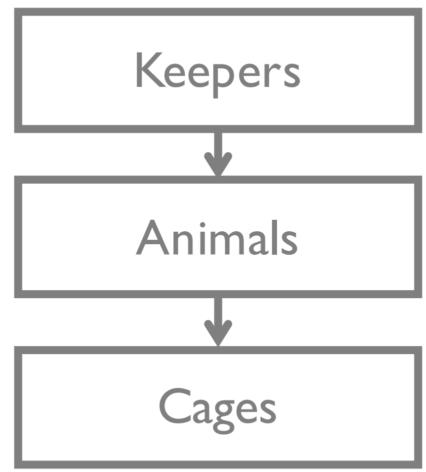
Segment instances (records)

Segment types form a tree



Hierarchical Model (IMS, circa 1968)

```
Jane (Keeper) (HSK 1)
   Bob, iguana, … (2)
      1, 100ft², … (3)
   Joe, student, … (4)
      1, 100ft², … (5)
…
```



What's repeated?

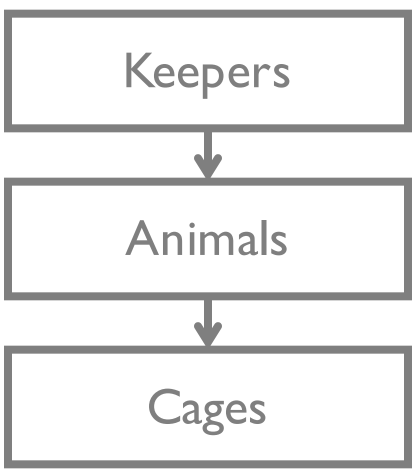
Inconsistencies possible, lack of protections

Hierarchical Model (IMS, circa 1968)

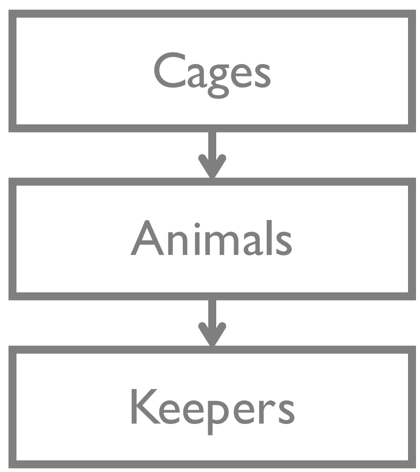
Segment types (objects)

Segment instances (records)

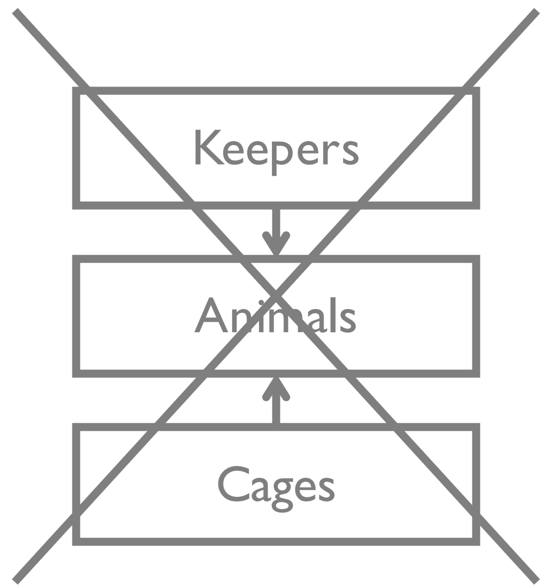
Segment types form a tree



Repeats cage data
(>1 animals in a cage)



Repeats keeper data
(>1 animals/keeper)



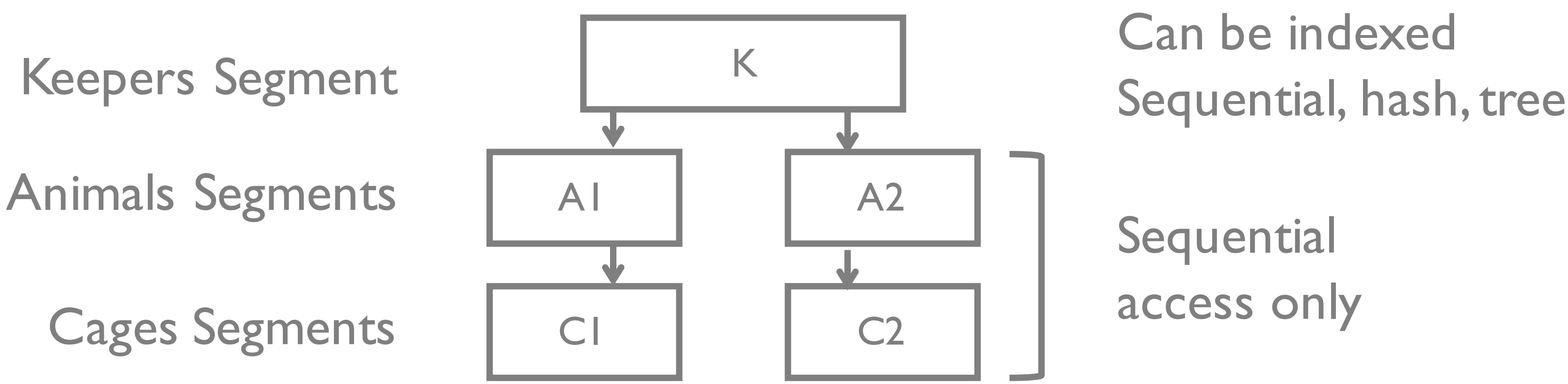
Disallowed

Physical Storage

Stored hierarchically

Only root segment can be indexed

Other segments only accessed sequentially



Hierarchical Querying: DL-1

Navigational Querying through a tree structure

Core operations

GX(seg, pred) general form, takes seg type and a predicate

Get Unique (GU) start at parent (root) segment

Get Next (GN) next record in HSK order in database

Get Next in Parent (GNP) next in HSK order until end of subtree

Fetch cages that Eugene entered

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

Find Keepers of Cage 6

```
keeper = GU(Keeper)
GNP(Cages, no=6)
print keeper
Until no more records
    keeper = GN(Keeper)
    GNP(Cages, no=6)
    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 GN calls now different

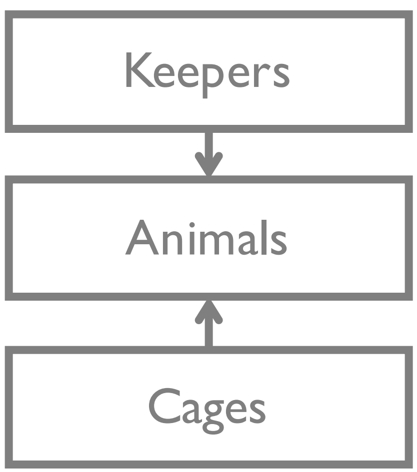
In reality, schemas change all the time

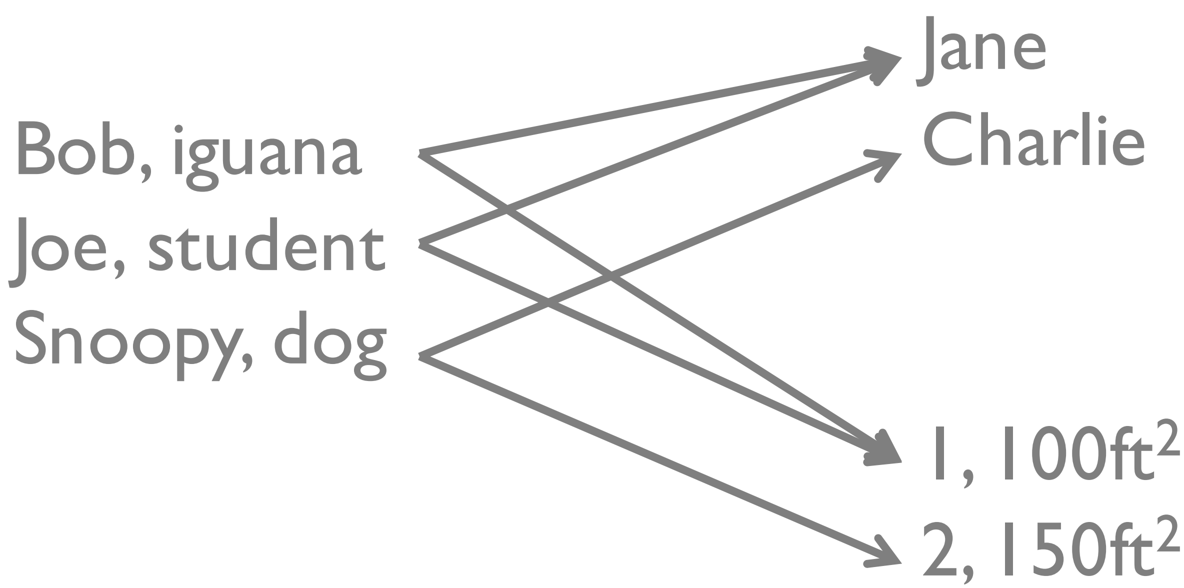
- Keepers now responsible for a whole cage
- Hummingbirds require multiple feedings
- Merge with another zoo

Network Models (CODASYL, 1969)

Abstraction

- Types of Records
- Connected by named sets (one to many relationships)
- Modeled as a graph





Network Models: Queries

Queries are programs that follow pointers (IMS style)

```
Find Keeper (name = 'Eugene')
until no more
   Find next Animal in cares_for
   Find Cage in lives_in
   Get current record
```

Very Smart people (Charles Bachman, '73 Turing Award)
strongly defended this model but…

Network Models: Problems

Very complex due to navigational programming

    (not for mere mortals!)

Still no physical nor logical data independence

Implementations were limiting

    must load all data at once

Trades off increased programmer pain for modeling 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

Roadmap

History lesson

DDLs: Data definition language

- Integrity Constraints

DMLs: Data Manipulation Language Selection

- Queries

ER → Relational Model

Basic Definitions

Database a set of relations

- Instance set of instances for relations in database

Relation 2 parts

- Instance a table with rows and columns
- Schema name of relation + name & type of each column
- e.g., Students(sid: int, name: string, login: string, age: int)

Think of relation as a *set* (no duplicate rows)

Relation colored glasses

- Everything (data, relationships, query results) is a relation

Terminology

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

Example *Instance* of Students Relation

<u>sid</u>	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

Cardinality 3
Degree 5

Do rows have to be distinct?
Do columns have to be distinct?

DDL: CREATE TABLE

Create the Students Relation

Note: attribute domains are defined & enforced by DBMS

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

Create the Enrolled relation

```
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
    www.postgresql.org/docs/8.1/static/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

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

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

What does this say?

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

Foreign Keys

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

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

Enrolled

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

Students

sid	name
1	eugene
2	luis

Referential Integrity

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

Examples where referential integrity is not enforced

- HTML links
- Yellow page listing
- Restaurant menus

CHECK Constraints

Boolean constraint expression added to schema

Very powerful mechanism.

More specific constraints in next slides

```
CREATE TABLE Enrolled(
    sid: int,
    cid: int,
    grade: char(2),
    CHECK (
        grade = 'A' or grade = 'B' or
        grade = 'C' or grade = 'D' or
        grade = '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?

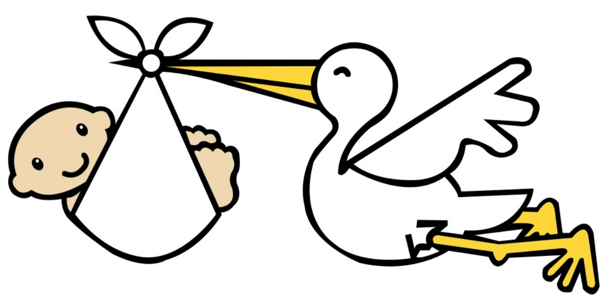
Based on application semantics and use cases

Can check if database instance satisfies ICs

IC is statement about the world (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.



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.

Normalization refines ER design by considering FDs.

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.

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

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’
```

Basic SELECT

Get all attributes of <21 year old students

```
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

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

Single Table Semantics

A *conceptual evaluation method* for previous query:

1. FROM clause: retrieve Students relation
2. WHERE clause: Check conditions, discard tuples that fail
3. SELECT clause: Delete unwanted fields

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

Multi-Table Semantics

Modify the FROM clause evaluation

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

Enrolled

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

Students

sid	name
1	eugene
2	luis

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

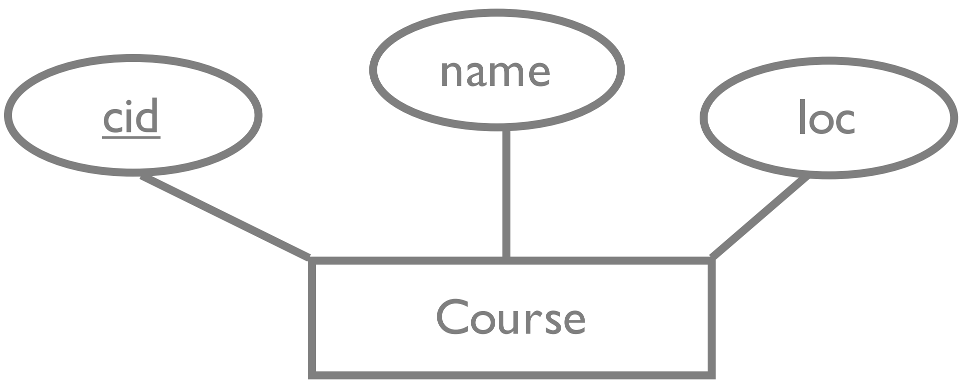
Multi-Table Semantics

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: Delete unwanted fields

Translating ER → Relational Models

Entity Set → Relation

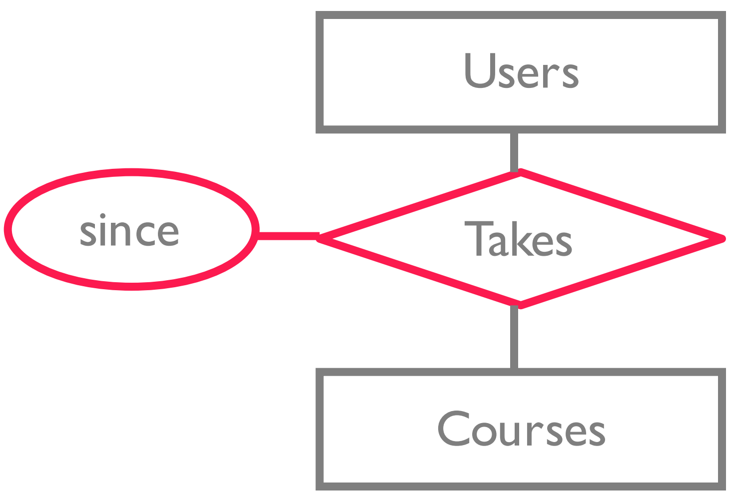



```
CREATE TABLE Course(
   cid int,
   name text,
   loc text,
   PRIMARY KEY (cid)
)
```

Relationship Set w/out constraint → Relation

Relation must include

- Keys for each entity set as foreign keys
    - these form superkey for relation
- All descriptive attrs.

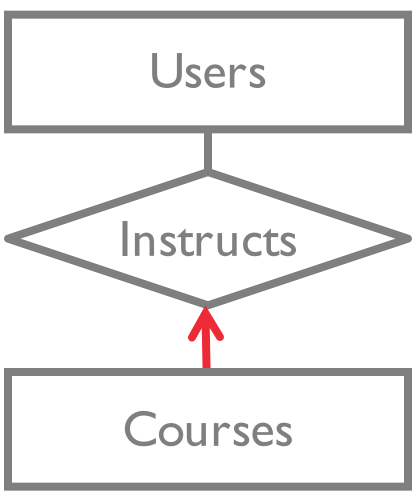



```
CREATE TABLE Takes(
    uid int,
    cid int,
    since date,
    PRIMARY KEY (uid, cid),
    FOREIGN KEY (uid) REFERENCES Users,
    FOREIGN KEY (cid) REFERENCES Courses
)
```

Key Constraint → Relation

Note only cid is a Key

User and Courses are separate relations

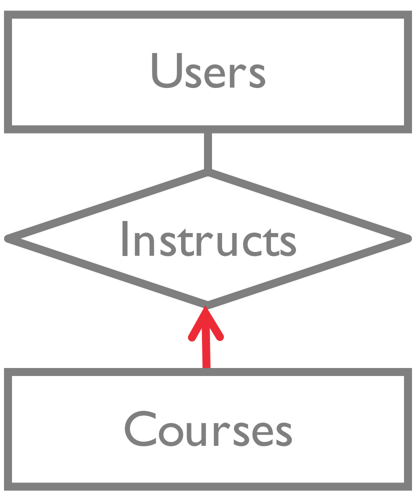



```
CREATE TABLE Instructs(
    uid int,
    cid int
    PRIMARY KEY (cid),
    FOREIGN KEY (uid) REFERENCES Users,
    FOREIGN KEY (cid) REFERENCES Courses
)
```

Key Constraint → Relation

Alternatively combine Courses and Users

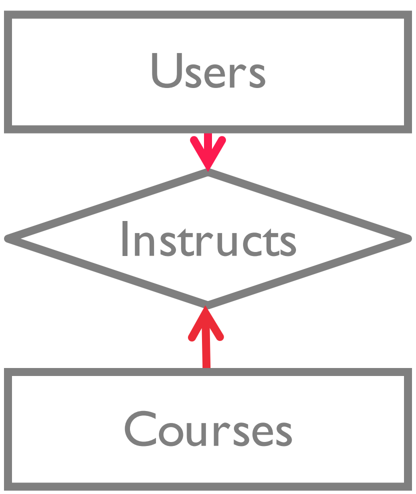
(this is the preferred way)




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

Key Constraint → Relation

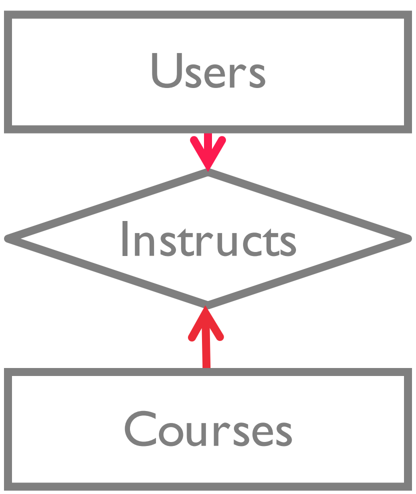
How to translate this ER diagram?




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

Key Constraint → Relation

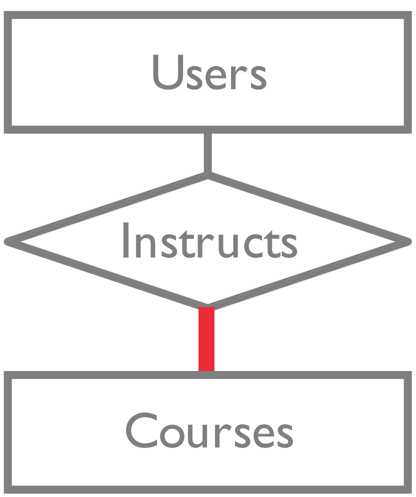
How to translate this ER diagram?




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

Participation Constraint → Relation

Only participation constraints with one entity set in binary relationship (others need CHECK constraint)

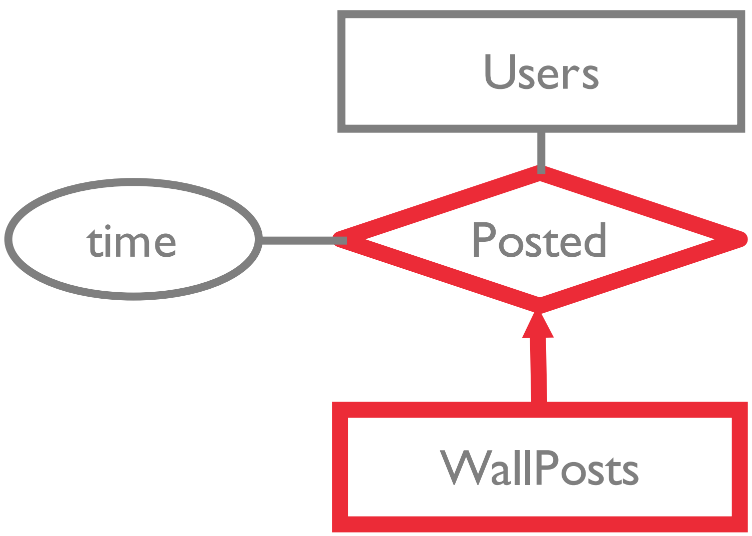



```
CREATE TABLE Course_Instructs(
    cid int
    uid int NOT NULL,
    name text,
    loc text,
    PRIMARY KEY (cid),
    FOREIGN KEY (uid) REFERENCES Users
        ON DELETE NO ACTION
)
```

Weak Entity → Relation

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

- When the owner entity is deleted, all owned weak entities must also be deleted.




```
CREATE TABLE Wall_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
)
```

ISA Hierarchies

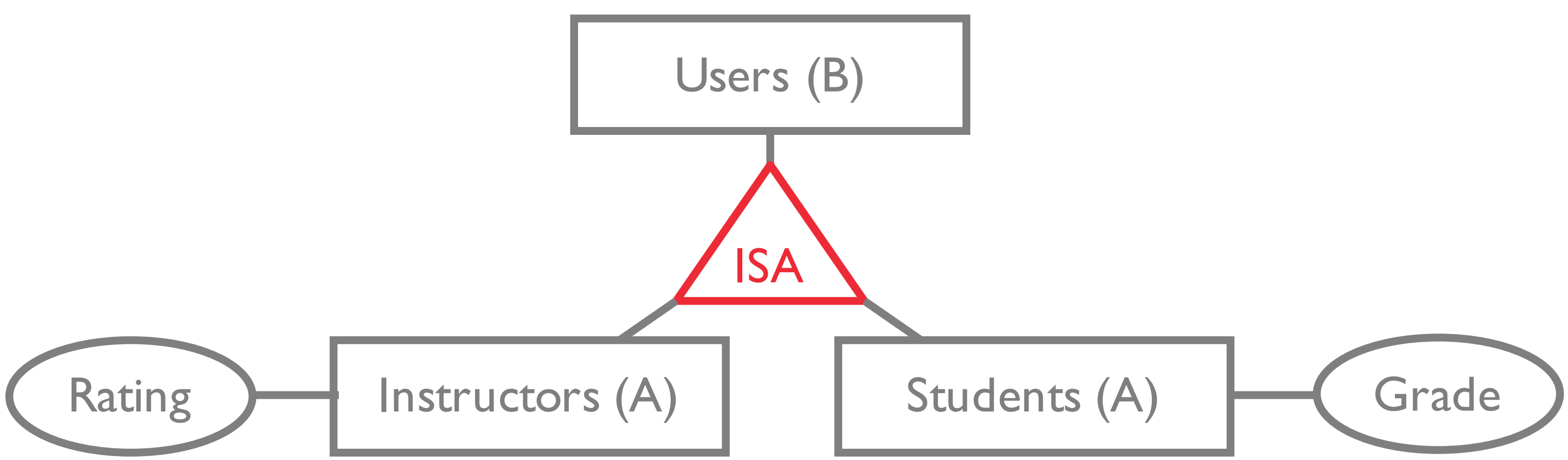
Option 1: Keep base relation

- Instructors & Students recorded in Users
- Extra info in Instructors or Students relation
- JOIN between child and base relations for all attributes

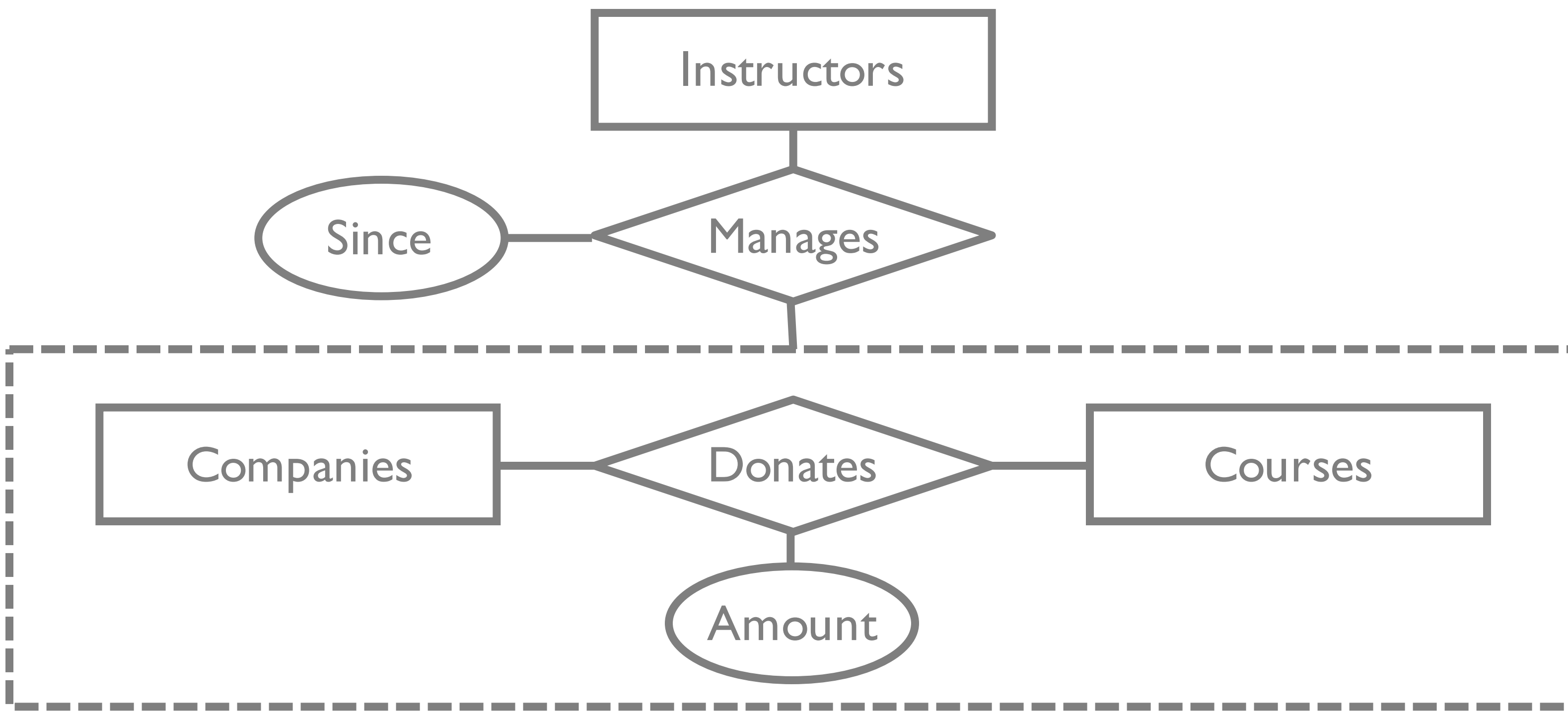
Option 2: Only keep child relations

- Instructors copies attributes from Users
- Instructors(uid, name, age, …, rating)

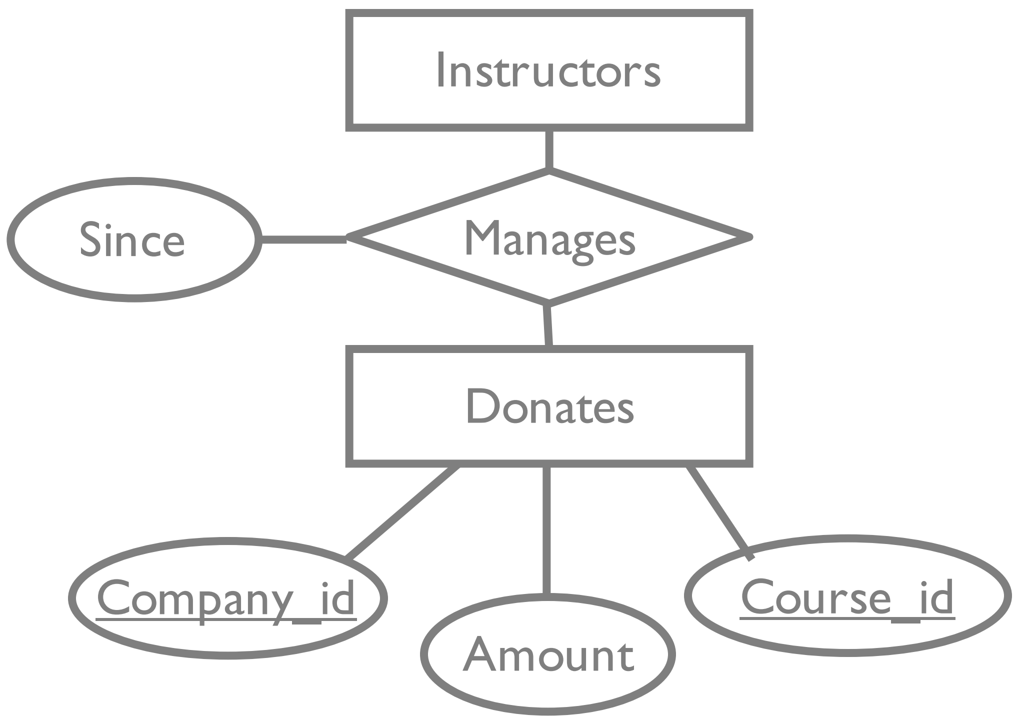
Only if covering constraint = yes



Aggregation



Aggregation



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