

CS460: Intro to Database Systems

Class 4: *The Relational Model*

Instructor: Manos Athanassoulis

<https://bu-disc.github.io/CS460/>

The Entity-Relationship Model

Basic ER modeling concepts

Constraints

Complex relationships

Conceptual Design

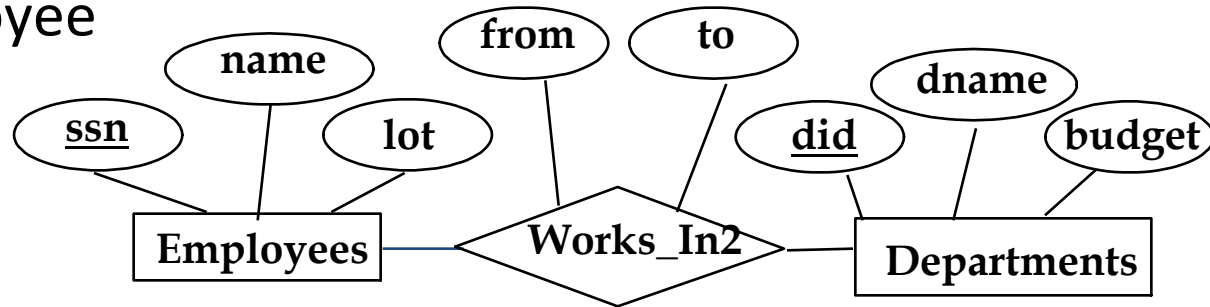
Readings: Chapter 2.5

From Previous Class

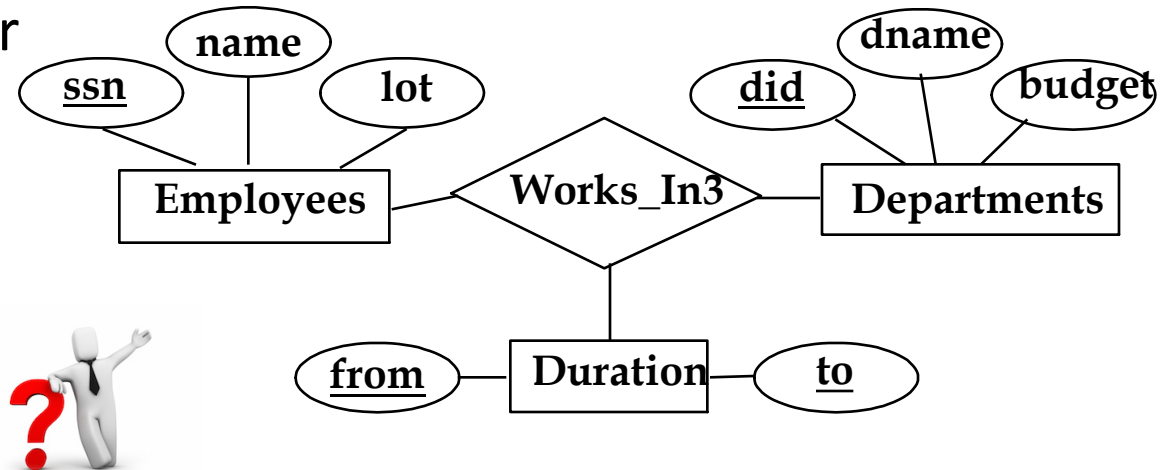
Review: Entity vs. Attribute

Works_In2 does not allow an employee to work in a department for two or more periods

why?

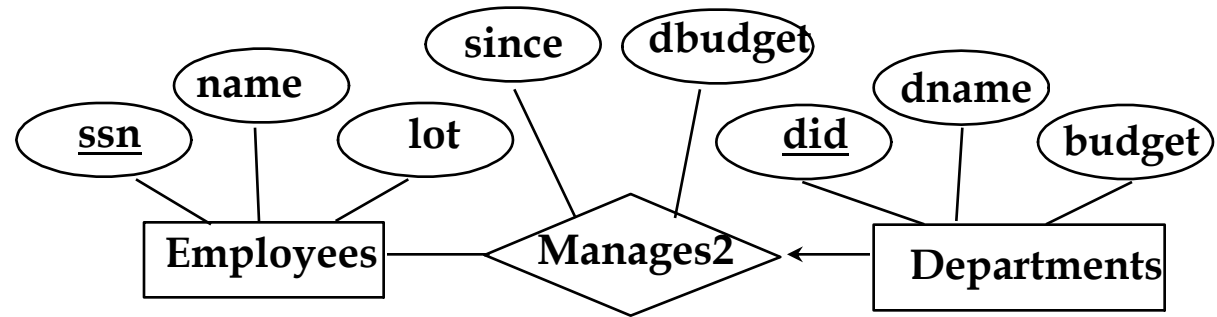


Approach: Similar to the problem of wanting to record several addresses for an employee: we want to record *several values of the descriptive attributes for each instance of this relationship*

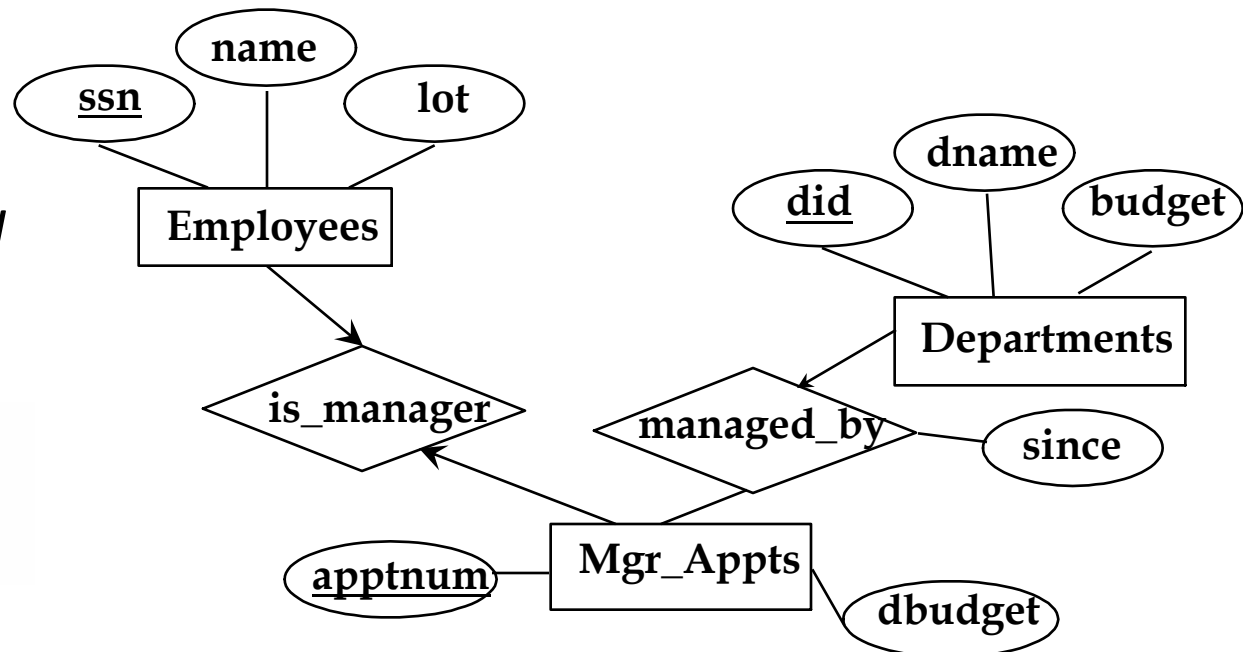


Review: Entity vs. Relationship

OK as long as a manager gets a separate discretionary budget (*dbudget*) for each department



What if manage's *dbudget* covers *all* managed departments? (can repeat value, but such redundancy is problematic)



Context: Overall Database Design Process

Requirements Analysis

Last time user needs; what must database do?



Conceptual Design

high level description (often done w/ER model)

Today: **Logical Design**

translate ER into DBMS data model

Schema Refinement

consistency, normalization

Physical Design

indexes, disk layout

Security Design

who accesses what

The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

ER to Relational

ISA to Relational

The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

ER to Relational

ISA to Relational

Why the Relational Model?

most widely used model

IBM, Microsoft, Oracle, etc.

”Legacy systems” in older models

e.g., IBM’s IMS

object-relational model incorporates oo concepts

IBM DB2, Oracle 11i

more recently: key-value store

Relational

tables with rows and columns

well-defined schema

data model fits data rather than
functionality

deduplication

Key/Value

collections of documents

schema-less (each document can
have different schema)

data stored in an application-
friendly way

possible duplication

Relational Database: Definitions

relational database: a collection (set) of *relations*

each relation: made up of 2 parts

schema: name of relation, name & type of each column

Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)

instance : a *table*, with rows and columns.

#rows = *cardinality*

#fields = *degree / arity*

can think of a relation as a *set* of rows or *tuples*

- (1) all rows are distinct
- (2) no order among rows

Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

cardinality = 3, arity = 5, all rows distinct

do all values in each column of a relation
instance have to be distinct?



SQL - A language for Relational DBs

SQL* (a.k.a. “Sequel”), standard language

Data Definition Language (DDL)

create, modify, delete relations

specify constraints

administer users, security, etc.

Data Manipulation Language (DML)

specify *queries* to find tuples that satisfy criteria

add, modify, remove tuples

* Structured Query Language

SQL Overview

```
CREATE TABLE <name> ( <field> <domain>, ... )
```

```
INSERT INTO <name> (<field names>)  
VALUES (<field values>)
```

```
DELETE FROM <name>  
WHERE <condition>
```

```
UPDATE <name>  
SET <field name> = <value>  
WHERE <condition>
```

```
SELECT <fields>  
FROM <name>  
WHERE <condition>
```

Creating Relations in SQL

type (**domain**) of each field is specified

also enforced whenever tuples are added or modified

```
CREATE TABLE Students  
  (sid CHAR(20),  
   name CHAR(20),  
   login CHAR(10),  
   age INTEGER,  
   gpa FLOAT)
```

Table Creation (continued)

Enrolled: holds information about courses students take

```
CREATE TABLE Enrolled  
  (sid CHAR(20),  
   cid CHAR(20),  
   grade CHAR(2))
```

Adding and Deleting Tuples

Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES ('53688', 'Smith', 'smith@cs', 18, 3.2)
```

Can delete all tuples satisfying some condition (e.g., name = Smith):

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

Powerful variants of these commands are available;
more later!

The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

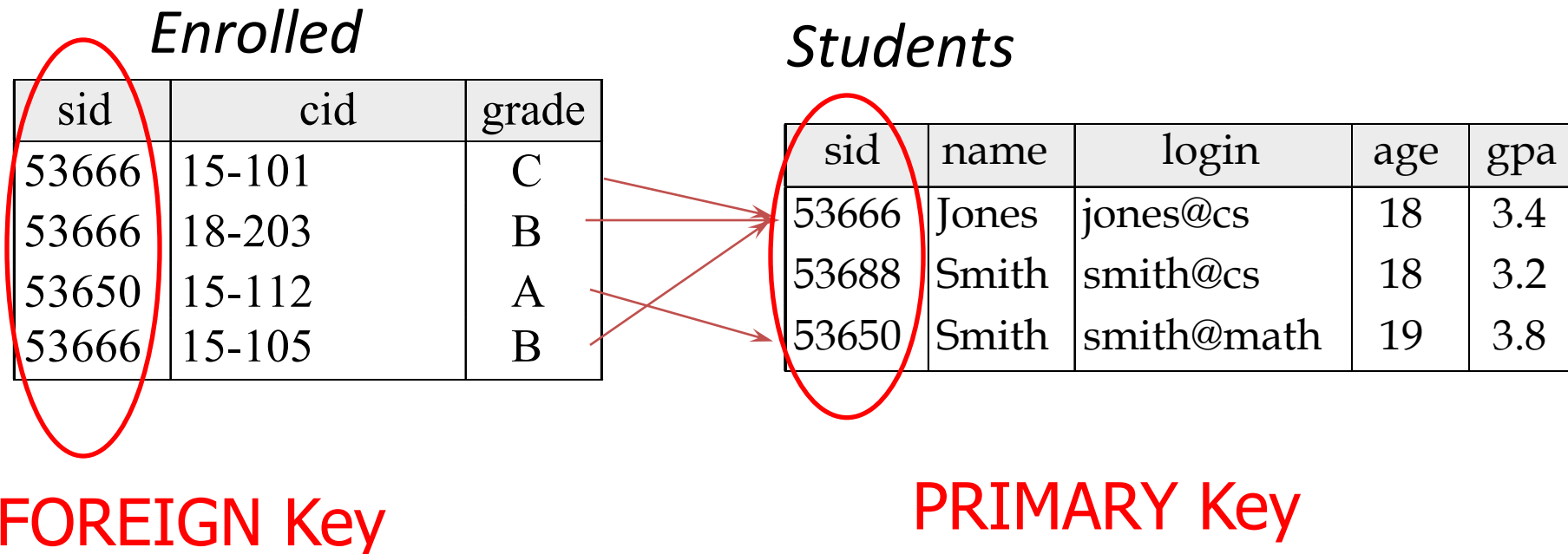
ER to Relational

ISA to Relational

Keys

keys: associate tuples in different relations

keys are one form of integrity constraint (IC)



Primary Keys

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

A set of fields is a superkey if:

No two distinct tuples can have same values in all key fields

Is <sid> a superkey?

What about <sid,name>?

What about <sid,name,age>?

What about <age,name>?



Primary Keys

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

A set of fields is a superkey if:

No two distinct tuples can have same values in all key fields

A set of fields is a key for a relation if :

It is a superkey

No subset of the fields is a superkey



Is <sid> a key? <sid,name>? <sid,name,age>? <age,name>?

Primary Keys

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

A set of fields is a superkey if:

No two distinct tuples can have same values in all key fields

A set of fields is a key for a relation if :

It is a superkey

No subset of the fields is a superkey



what if >1 key for a relation?

chose one as the **primary key** / rest called **candidate** keys

Primary and Candidate Keys in SQL

possibly many *candidate keys* (specified using **UNIQUE**),
one of which is chosen as the *primary key*

keys must be defined carefully!

“for a given student and course, there is a single grade”

```
CREATE TABLE Enrolled  
(sid CHAR(20)  
  cid CHAR(20),  
  grade CHAR(2),  
  PRIMARY KEY (sid,cid))
```

VS.

```
CREATE TABLE Enrolled  
(sid CHAR(20)  
  cid CHAR(20),  
  grade CHAR(2),  
  PRIMARY KEY (sid),  
  UNIQUE (cid, grade))
```



Foreign Keys, Referential Integrity

foreign key: set of fields in one relation that is used to “refer” to a tuple in another

correspond to the primary key of the other relation a “logical pointer”

If all foreign key constraints are enforced, referential integrity is achieved (i.e., no dangling references)

Foreign Keys in SQL

Example: Only students listed in the Students relation should be allowed to enroll for courses.

sid is a foreign key referring to **Students**

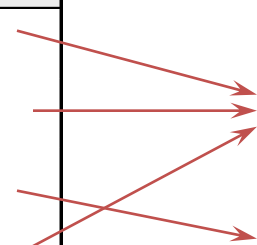
```
CREATE TABLE Enrolled  
(sid CHAR(20),cid CHAR(20),grade CHAR(2),  
PRIMARY KEY (sid,cid),  
FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8



Enforcing Referential Integrity

Students and Enrolled; *sid* in Enrolled is a FK references Students

What to do if a tuple with a non-existent *sid* is inserted in Enrolled?

What should be done if a Students tuple is deleted?



Enforcing Referential Integrity

Students and Enrolled; *sid* in Enrolled is a FK references Students

What to do if a tuple with a non-existent *sid* is inserted in Enrolled?

What should be done if a Students tuple is deleted?

Also delete all Enrolled tuples that refer to it?

Disallow deletion of a Students tuple that is referred to?

Set *sid* in Enrolled tuples that refer to it to a *default sid*?

(In SQL we can set *sid* to be equal to *null*, denoting “unknown” or “inapplicable”)

Similar issues arise if primary key of Students tuple is updated



Integrity Constraints (ICs)

IC: must be true for *any* instance of the database
(e.g., *domain constraints*)

ICs are specified when schema is defined

ICs are checked when relations are modified

a *legal* instance of a relation satisfies *all specified ICs*

DBMS should not allow illegal instances

if the DBMS checks ICs, stored data is more faithful to real-world meaning
avoids data entry errors, too!

Where do ICs Come From?

ICs are based upon the *real-world semantics*

we can check a database instance to see if an IC is violated, but we cannot infer that an IC hold

An IC is a statement about *all possible* instances!

From example, we know *name* is not a key, but the assertion that *sid* is a key is given

key and foreign key ICs are the most common
(more general ICs supported too)

The Relational Model

Intro & SQL overview

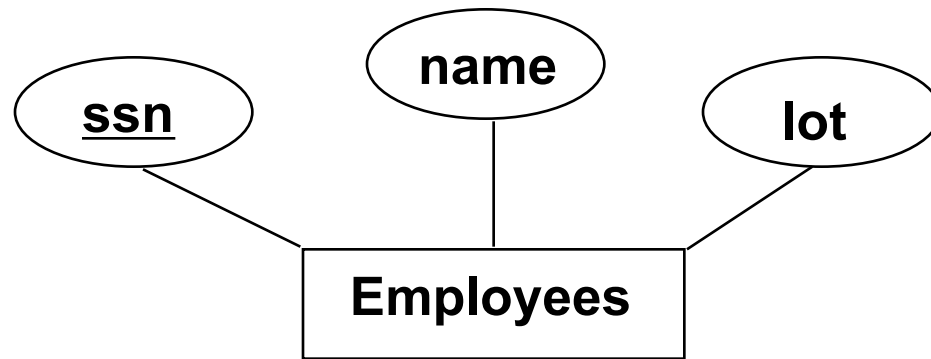
Keys & Integrity Constraints

ER to Relational

ISA to Relational

Logical DB Design: ER to Relational

Entity sets to tables

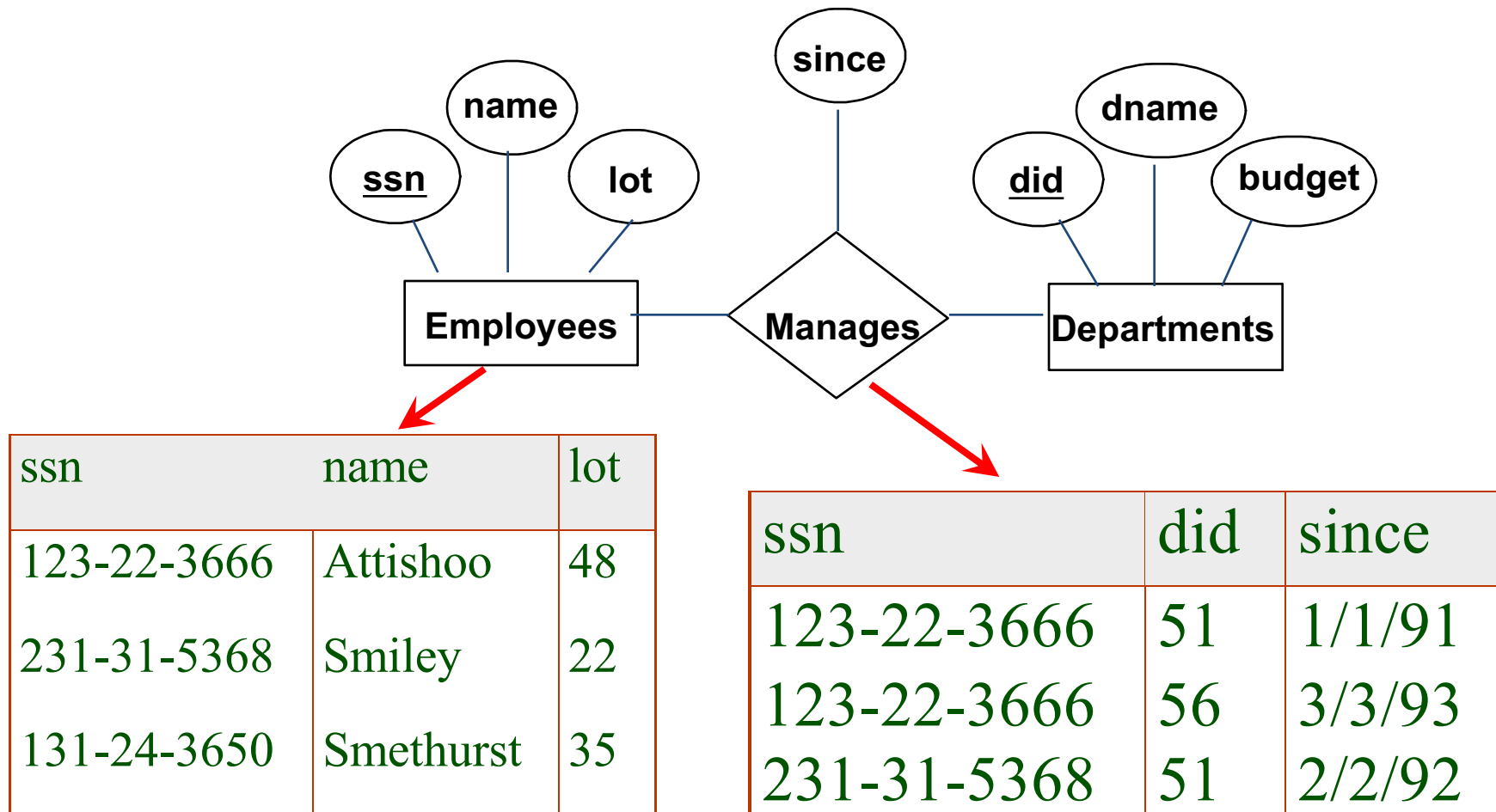


ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

```
CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn))
```

Relationship Sets to Tables

Our favorite example:



Relationship Sets to Tables

In translating a **many-to-many** relationship set to a relation, attributes of the relation must include:

Keys for each participating entity set (as foreign keys). This set of attributes forms a **superkey** for the relation.

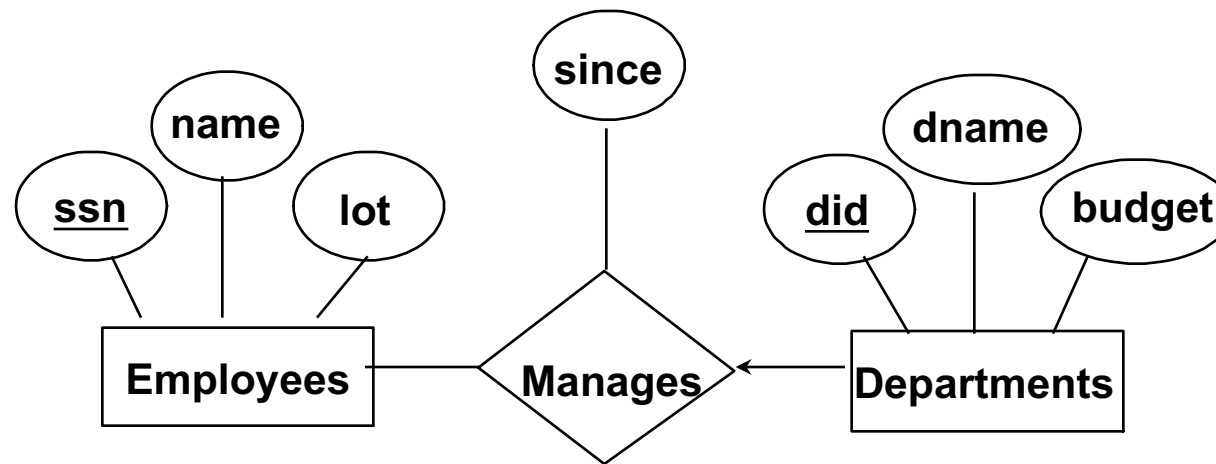
All descriptive attributes.

```
CREATE TABLE Manages(  
    ssn CHAR(11),  
    did INTEGER,  
    since DATE,  
    PRIMARY KEY (ssn, did),  
    FOREIGN KEY (ssn)  
        REFERENCES Employees,  
    FOREIGN KEY (did)  
        REFERENCES Departments)
```

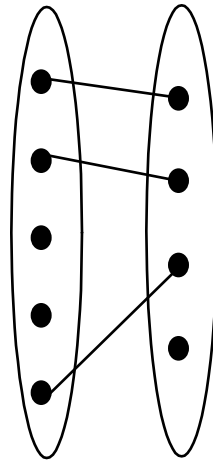
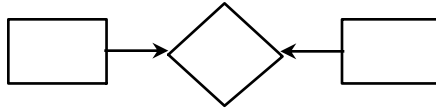
ssn	did	since
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

Review: Key Constraints in ER

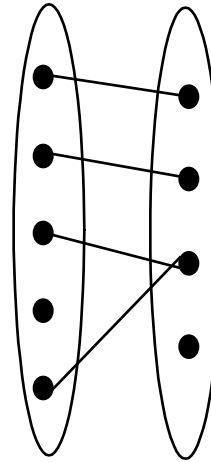
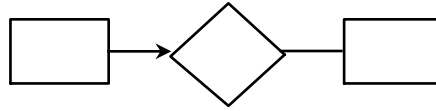
Each dept has at most one manager, according to the key constraint on Manages



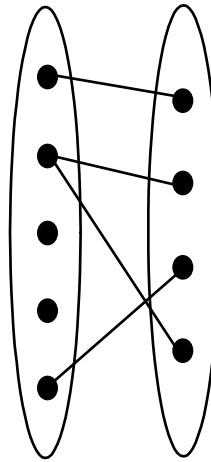
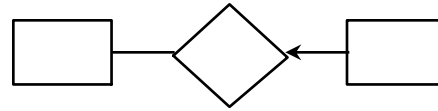
Review: Key Constraints in ER



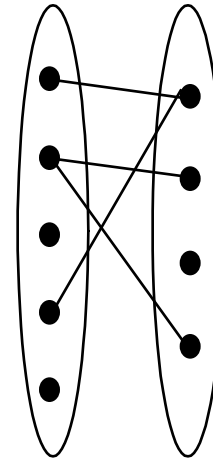
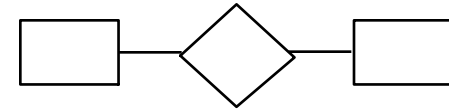
1-to-1



Many-to-1

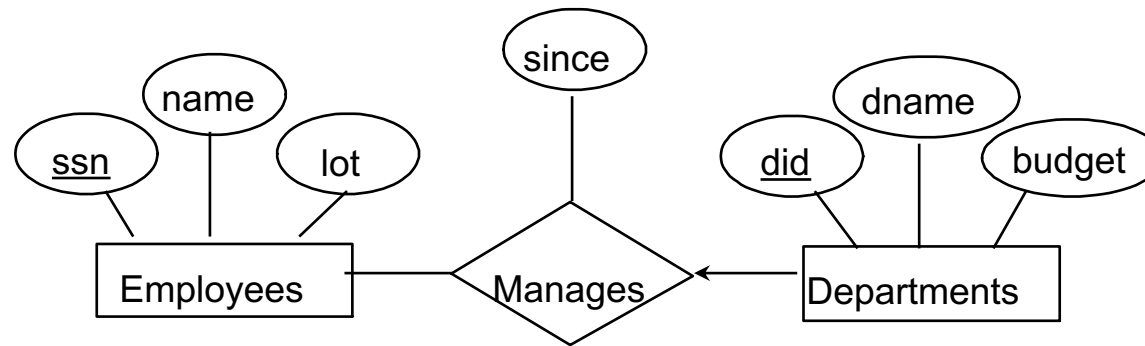


1-to Many



Many-to-Many

Translating ER with Key Constraints



since each department has a unique manager, we could instead combine Manages and Departments

```

CREATE TABLE Manages(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES
  Employees,
  FOREIGN KEY (did) REFERENCES
  Departments)
  
```

Vs.

```

CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn)
  REFERENCES Employees)
  
```

What if the toy department has no manager (yet) ?



Can be NULL!



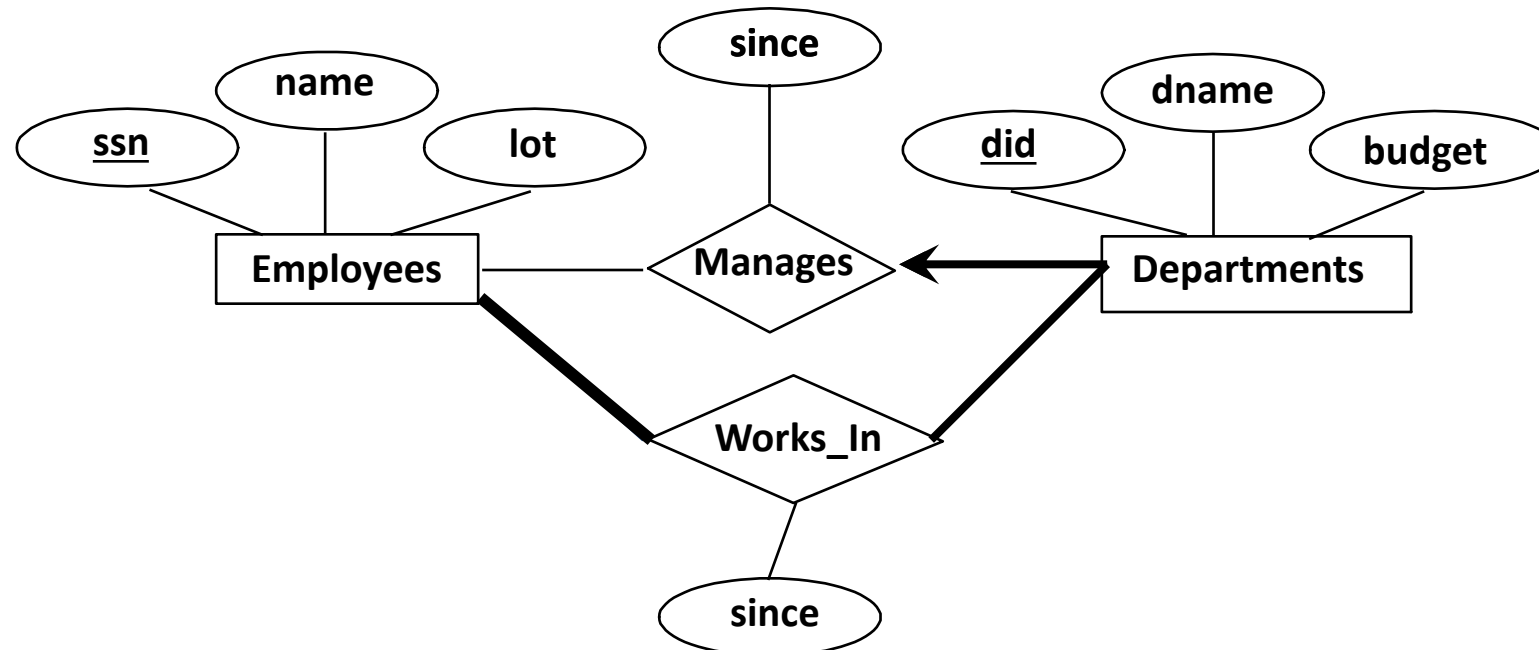
```
CREATE TABLE Dept_Mgr(  
  did INTEGER,  
  dname CHAR(20),  
  budget REAL,  
  ssn CHAR(11),  
  since DATE,  
  PRIMARY KEY (did),  
  FOREIGN KEY (ssn)  
    REFERENCES Employees)
```

Review: Participation Constraints

does every employee work in a department?

If so, this is a participation constraint: the participation of Departments in Manages is said to be *total (vs. partial)*

Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



Participation Constraints (PC) in SQL

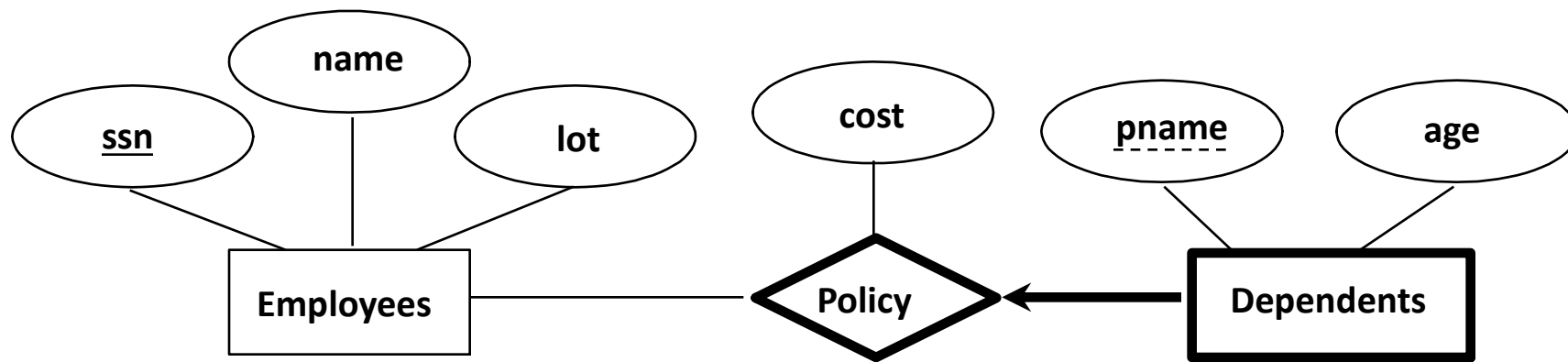
PCs of one entity set in a binary relationship, yes!
but little else (without resorting to `CHECK` constraints)

```
CREATE TABLE Dept_Mgr(  
    did INTEGER,  
    dname CHAR(20),  
    budget REAL,  
    ssn CHAR(11) NOT NULL,  
    since DATE,  
    PRIMARY KEY (did),  
    FOREIGN KEY (ssn) REFERENCES Employees,  
    ON DELETE NO ACTION)
```

Review: Weak Entities

A *weak entity* can be identified uniquely by the primary key of another (*owner*) entity (+ some of its attributes)

- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities)
- Weak entity set must have total participation in this *identifying* relationship set



Translating Weak Entity Sets

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 Dep_Policy (  
    pname CHAR(20),  
    age INTEGER,  
    cost REAL,  
    ssn CHAR(11) NOT NULL,  
    PRIMARY KEY (pname, ssn),  
    FOREIGN KEY (ssn) REFERENCES Employees,  
    ON DELETE CASCADE)
```


The Relational Model

Intro & SQL overview

Keys & Integrity Constraints

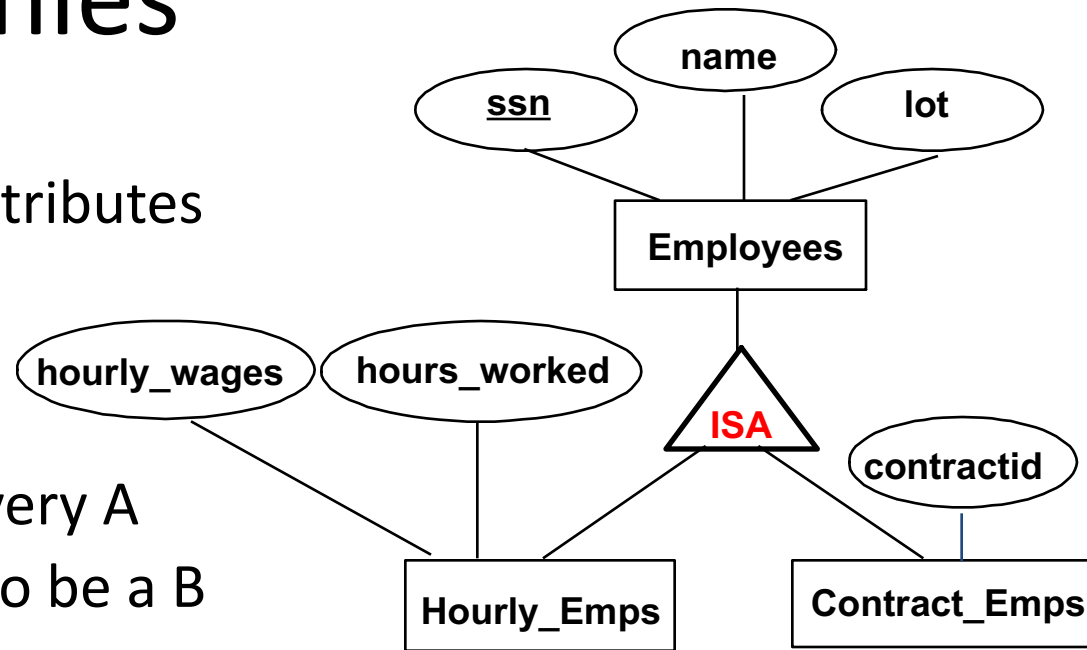
ER to Relational

ISA to Relational

Review: ISA Hierarchies

As in C++, or other PLs, attributes are inherited.

If we declare A **ISA** B, every A entity is also considered to be a B entity.



Overlap constraints: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)

Covering constraints: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? (*Yes/no*)

Translating ISA Hierarchies to Relations

```
CREATE TABLE Employees (  
  ssn CHAR(11) NOT NULL,  
  name CHAR(20),  
  lot INTEGER,  
  PRIMARY KEY (ssn))
```

```
CREATE TABLE Contract_Emps (  
  ssn CHAR(11) NOT NULL,  
  contractid INTEGER,  
  PRIMARY KEY (ssn),  
  FOREIGN KEY (ssn)  
    REFERENCES Employees)
```

```
CREATE TABLE Hourly_Emps (  
  ssn CHAR(11) NOT NULL,  
  hourly_wages REAL,  
  hours_worked REAL,  
  PRIMARY KEY (ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees)
```

what should happen if I delete an entry from Employees?

can we use ON DELETE CASCADE?

how to access name and hours worked ?

Join!



Alternative approach for ISA Hierarchies

```
CREATE TABLE Hourly_Emps (  
  ssn CHAR(11) NOT NULL,  
  name CHAR(20),  
  lot INTEGER,  
  hourly_wages REAL,  
  hours_worked REAL,  
  PRIMARY KEY (ssn))
```

```
CREATE TABLE Contract_Emps  
(ssn CHAR(11) NOT NULL,  
  name CHAR(20),  
  lot INTEGER,  
  contractid INTEGER,  
  PRIMARY KEY (ssn))
```

how to ensure that every employee is only in one of the two?



what about Employees that are neither?



what about querying for all employees?



Query 2 tables!

Relational Model: Summary

tabular representation of data

simple & intuitive, currently the most widely used

Integrity Constraints can be specified based on app semantics &
DBMS checks for violations

two important ICs: primary and foreign keys

in addition, we *always* have domain constraints

ER to Relational is (fairly) straightforward