

The Relational Model

Juliana Freire

Some slides adapted from L. Delcambre, R. Ramakrishnan, G. Lindstrom and Silberschatz, Korth and Sudarshan

Database Model

- Provides the means for *specifying particular data structures*, for *constraining the data sets* associated with these structures, and for *manipulating* the data
- Data definition language (DDL): define structures and constraints
- Data manipulation language (DML): specify manipulations/operations over the data

Why Study the Relational Database Model?

- Extremely useful and simple
 - Single data-modeling concept: relations = 2-D tables
 - Allows clean yet powerful manipulation languages
- Most widely used model
 - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- Recent competitors: object-oriented model, semi-structured (XML) model
 - ObjectStore, Versant, Tamino
 - A synthesis emerging:
 - *object-relational model*: Informix Universal Server, UniSQL, O2, Oracle, DB2
 - XML-enabled databases: Oracle, DB2, SQLServer

The Evolution of the Relational Model

- **Codd's seminal paper:** use relations as data structures, algebra for specifying queries, no mechanisms for updates or constraints
- **Codd's follow-up papers:** introduced new language based on first-order logic and showed it is equivalent to the algebra, introduced integrity constraints
- **Other extensions:** update operators, arithmetic operators, aggregation and sorting
- **Relational Model** is the broad class of database models that have *relations as the data structure*, and that *incorporate query and update capabilities, and integrity constraints*

Example: A Relation

*Relation
name*

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

The *Account* relation keeps track of bank accounts.

Facts about real-world entities:

J. Smith owns a checking account whose number is 101 and balance is 1000.00

Example: A Relation

The name of the attributes (columns)

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Each row/tuple corresponds to an account entity

Attribute domains:

Number must be a 3-digit number

Owner must be a 30-character string

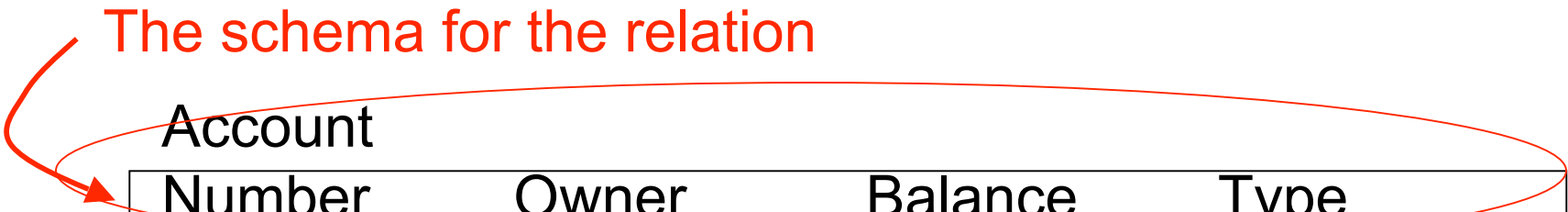
Type must be “checking” or “savings”

Basic Structure

- Given sets D_1, D_2, \dots, D_n a **relation** r is a subset of $D_1 \times D_2 \times \dots \times D_n$
Thus a relation is a **set** of n-tuples (a_1, a_2, \dots, a_n) where each $a_i \in D_i$
- Each attribute of a relation has a name
- Set of allowed values for attribute is called the **domain** of the attribute
- Attribute values are required to be **atomic**, that is, indivisible
 - E.g. multi-valued attribute values are not atomic
 - E.g. composite attribute values are not atomic
- The special value *null* is a member of every domain

Example: Relation Schema

The schema for the relation



Account			
Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Account(Number, Owner, Balance, Type)

The **schema** sets the structure of the relation---it is the **definition** of the relation.

(Note: the schema specifies more information than what is shown – remember keys, constraints...)

Relation Schema

- A_1, A_2, \dots, A_n are *attributes*
- $R = (A_1, A_2, \dots, A_n)$ is a *relation schema*
E.g. Account-schema =
 $(number, owner, balance, type)$
- $r(R)$ is a *relation* on the *relation schema* R
E.g. *account* (Account-schema)
 - Says “account is a relation conforming to Account-schema”
- Attributes of a relation form a *set*, not a list!
 - We often must specify a standard order

Relation Instance

An instance of the relation...

the current contents or data in the relation.

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Relation Instance (cont.)

Another instance of the relation
(two rows added, one (103) deleted)

Account				
Number	Owner	Balance	Type	
101	J. Smith	1,000.00	checking	
102	W. Wei	2,000.00	checking	
103	M. Jones	1,000.00	checking	
104	H. Martin	10,000.00	checking	
105	W. Yu	7,500.00	savings	
107	R. Jones	432.55	checking	
109				

103 deleted

new

Rows/Tuples

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Each entry in the relation is called a **row**, or a **tuple**, or a **record**.

Order of tuples is irrelevant. Why?

Rows/Tuples

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Each entry in the relation is called a **row**, or a **tuple**, or a **record**.

Order of tuples is irrelevant. Why?
relation is a **set, not a list!**

Degree and Cardinality

Degree or arity of a relation is the number of attributes

Degree of this relation (or table) is 4
because there are 4 attributes

Account	Number	Owner	Balance	Type
→	101	J. Smith	1000.00	checking
→	102	W. Wei	2000.00	checking
→	103	J. Smith	5000.00	savings
→	104	M. Jones	1000.00	checking
→	105	H. Martin	10,000.00	checking

Cardinality of this instance is 5 (because there are 5 rows)

Cardinality of a relation = the number of rows in the current instance

Relational Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information
 - E.g.: *account* : information about accounts
 - deposit*: information about deposits into accounts
 - check* : information about checks
- Storing all information as a single relation such as *bank(account-number, balance, customer-name, deposit-date, deposit-amount..)* results in
 - repetition of information (e.g., two customers own an account)
 - the need for null values (e.g., represent a customer without an account)

Relational Database Example

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

Check	Account	Check-number	Date	Amount
	101	924	10/23/00	125.00
	101	925	10/24/00	23.98

Relational Database Example (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

Check	Account	Check-number	Date	Amount
	101	924	10/23/98	125.00
	101	925	10/24/98	23.98

Each Relation has a key.... where the values must be unique.

Keys

- Let $K \subseteq R$
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
 - Example: $\{account\text{-}number, account\text{-}owner\}$ and $\{account\text{-}number\}$ are both superkeys of *Account*, if no two accounts can possibly have the same number.
- K is a **candidate key** if K is minimal
Example: $\{account\text{-}number\}$ is a candidate key for *Account* – it is a superkey and no subset of it is a superkey

Relational Database Example (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00
	106	5	12/5/00	555.00

Is this legal? If not, how do we prevent it from happening?

Relational Database Example (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00
	106	5	12/5/00	555.00

We say that **Deposit.Account** is a foreign key that references **Account.Number**. If the DBMS enforces this constraint we say we have **referential integrity**.

Relational Database Example (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Check	Account	Check-number	Date	Amount
	101	924	10/23/98	125.00
	101	925	10/24/98	23.98

Similarly, Check.Account is a foreign key that references Account.Number.

Specification of a Relational Schema

- Select the relations, with a **name for each table**
- Select **attributes for each relation** and give the **domain for each attribute**
- Specify the **key(s)** for each relation
- Specify all appropriate **foreign keys** and **integrity constraints**
- *Database schema* is the set of schemas for the relations in a design

Another Example Database (keys are underlined)

Teacher (Number, Name, Office, E-mail)

Course (Number, Name, Description)

Taught-By (Quarter, Course, Section, Teacher, TimeDays)

Student (Number, Name, Major, Advisor)

Completed (Student, Course, Quarter, Section, Grade)

Example Database (cont.)

(with foreign keys shown informally, with arrows)

Teacher (Number, Name, Office, E-mail)

Course (Number, Name, Description)

Taught-By (Quarter, Course, Section, Teacher, TimeDays)

Student (Number, Name, Major, Advisor)

Completed (Student, Course, Quarter, Section, Grade)

What foreign keys are present in the Completed table?

Example Database (cont.)

(with foreign keys shown informally, with arrows)

Teacher (Number, Name, Office, E-mail)

Course (Number, Name, Description)

Taught-By (Quarter, Course, Section, Teacher, TimeDays)

Student (Number, Name, Major, Advisor)

Completed (Student, Course, Quarter, Section, Grade)

Foreign keys in the Completed table are shown above.

Any feeling of deja vous?

Logical Design

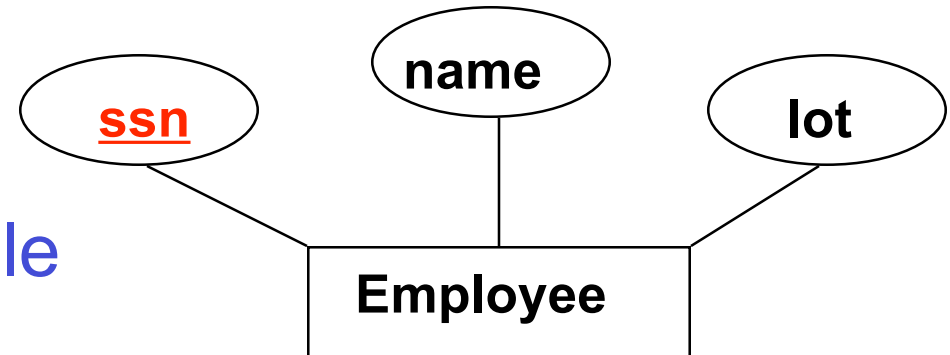
- ER used in **conceptual design**
- ER → Relational Schema: **logical design**
- Primary keys allow entity sets and relationship sets to be expressed uniformly as *tables* which represent the contents of the database
- For each entity set and relationship set there is a unique table which is assigned the name of the corresponding entity set or relationship set
- Each table has a number of columns (generally corresponding to attributes), which have unique names

Translating an ER Diagram into a Relational Schema

First Approximation

- Turn each entity into a relation with the same set of attributes
- Replace relationship by a relation whose attributes are the keys for the connected entity sets
- But we still need to deal with *special situations*
 - Multi-valued and composite attributes
 - Weak entity sets
 - ISA relationships
 - Relations that result from an entity set and 1-to-many relationship

1. Translate each (strong) entity set into a table with keys



- **Entity:**
 - can be represented as a table in the relational model – entity set attributes become attributes of the relation
 - has a **key** ... which becomes a key for the table

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

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

A DBMS may or may not allow multi-valued attributes.
If it doesn't,
2. Create a table for the multi-valued attribute.

How many offices can one employee have?

*Just
one*

Employee (SSN, E-Name, Office)

vs.

*More
than
one*

Employee (SSN, E-Name)
Office-Assignment (SSN, Office)

Sample Data

Employee (SSN, E-Name, **Office**)

*Just
one*

12	Smith	O-105
15	Wei	O-110
20	Jones	O-112

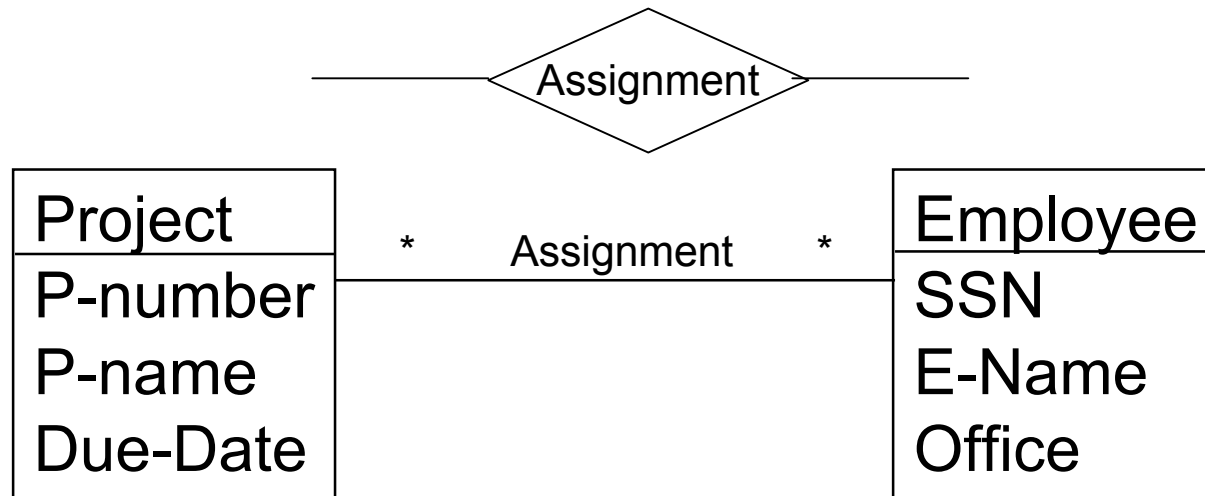
Employee (SSN, E-Name)

*More
than
one*

12	Smith
15	Wei

Office-Assignment (SSN, Office)

12	O-105
12	O-106
15	O-110



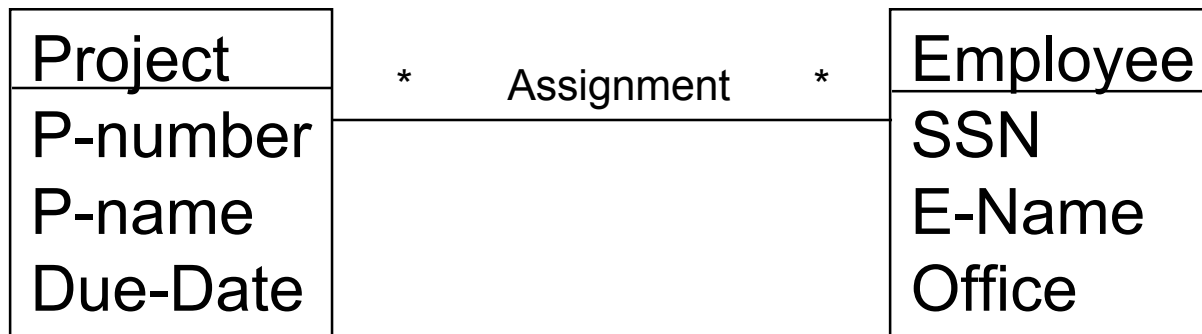
3. Translate each **many-to-many** relationship into a table: key and attributes of relationship become the key and attributes of relation

What are the attributes and what is the key for Assignment?

Assignment (?)

Project (P-number, P-name, Due-Date)

Employee (SSN, E-Name, Office)



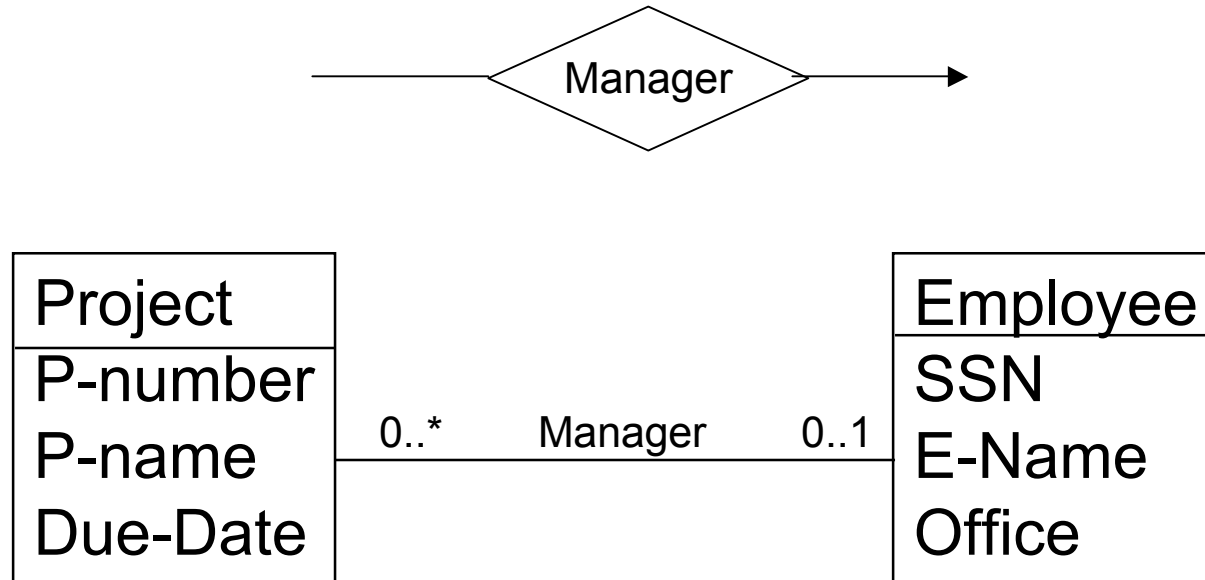
Answer: Assignment (P-Number, SSN)

Project (P-Number, P-Due-Date)

Employee (SSN, E-Name, Office)

P-Number is a foreign key for Project

SSN is a foreign key for Employee

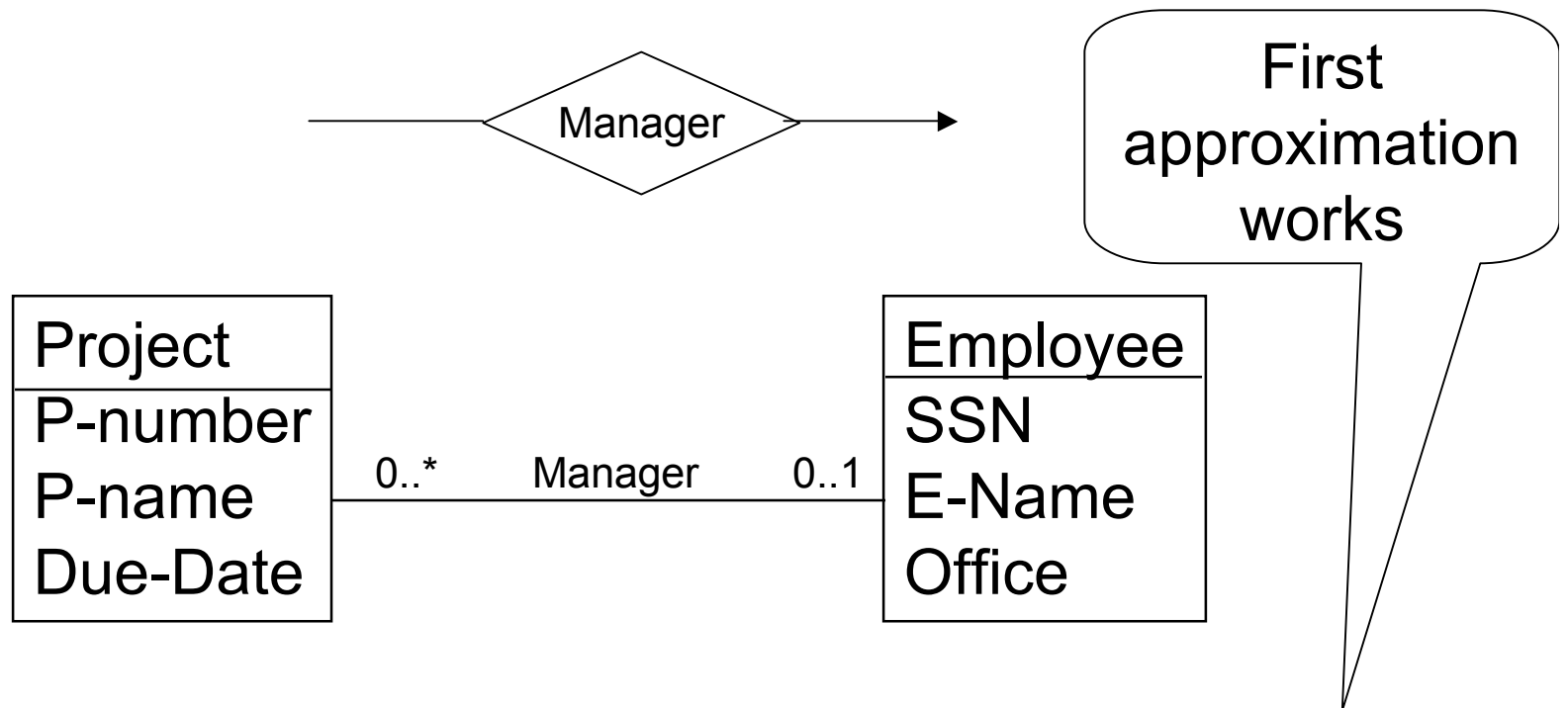


4. Translating **one-to-many** relationships

Manager (?)

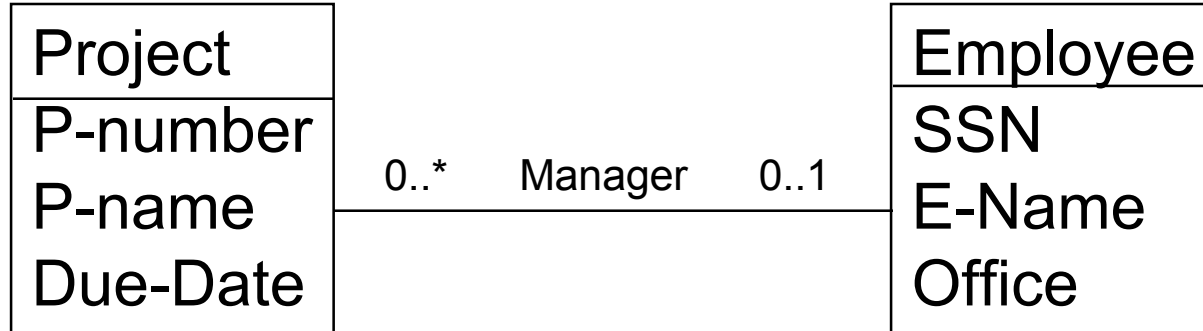
Project (P-number, P-name, Due-Date)

Employee (SSN, E-Name, Office)



4.1 Translating **one-to-many** relationships: create relation for relationship

Manager (SSN, P-number)
 Employee (SSN, E-Name, Office)
 Project (P-number, P-name, Due-Date)



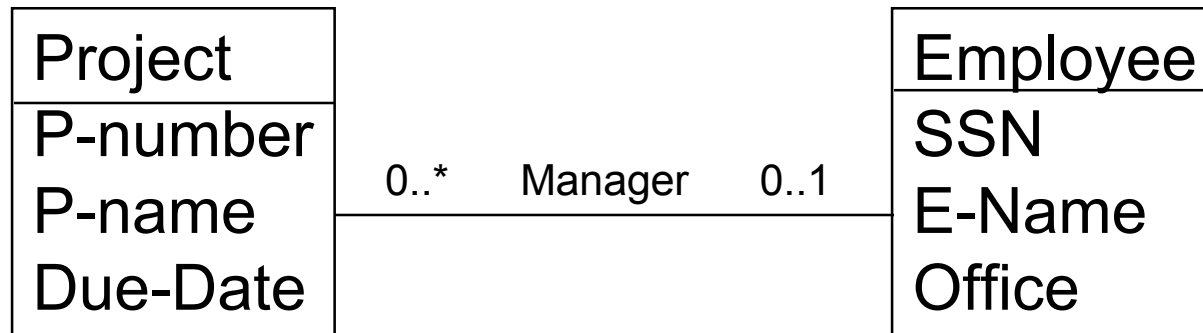
Project (P-number, P-name, Due-Date, **Manager**)
 Employee (SSN, E-Name, Office)



4.2 Translating **one-to-many** relationships: add a foreign key to the many relation, i.e., add key from the “one” side to the relation corresponding to the “many” side

Manager is a foreign key (referencing the Employee relation)
 value of Manager must match an SSN

no need to represent the relationship as a relation!

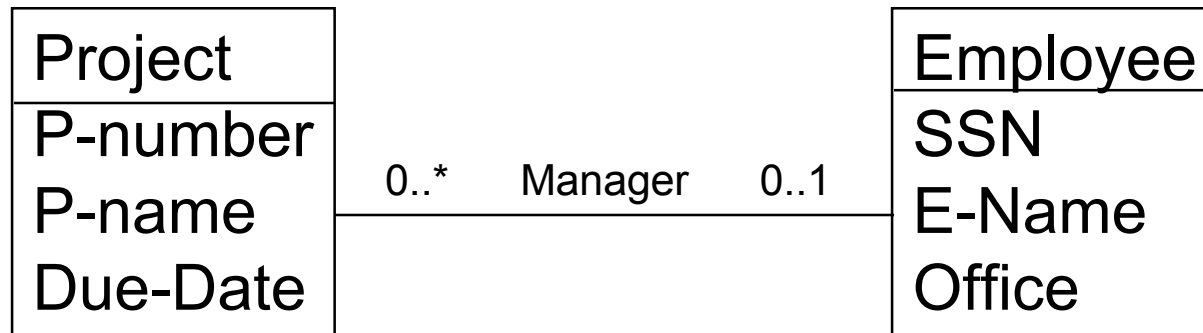


Project (P-number, P-name, Due-Date, **Manager**)
 Employee (SSN, E-Name, Office)

vs.

Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office)
Manager (P-number, SSN)

What are the tradeoffs between these two?

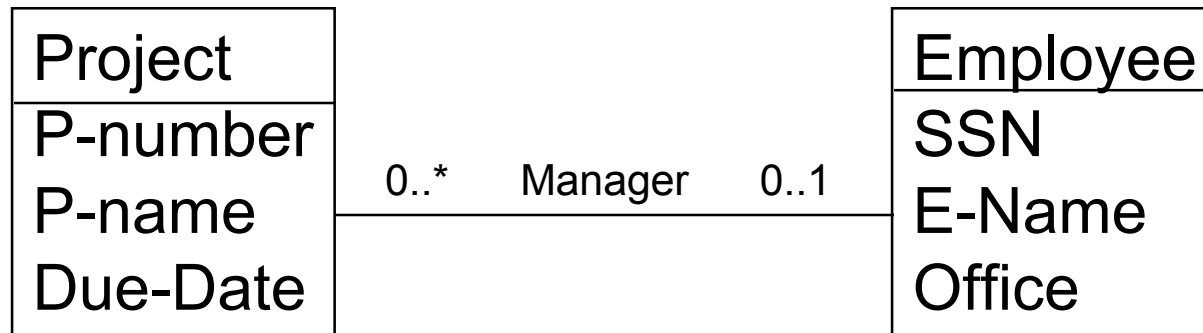


What about:

Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office, P-number)

vs.

Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office)
 Manager (P-number, SSN)



What about:

Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office, P-number)

vs.

Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office)
 Manager (P-number, SSN)

Repeat
 employee info
 for multiple
 projects



Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office, P-number)

VS.

Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office)
 Manager (P-number, SSN)

What if SSN is the key for Manager?



Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office, Managed-project)

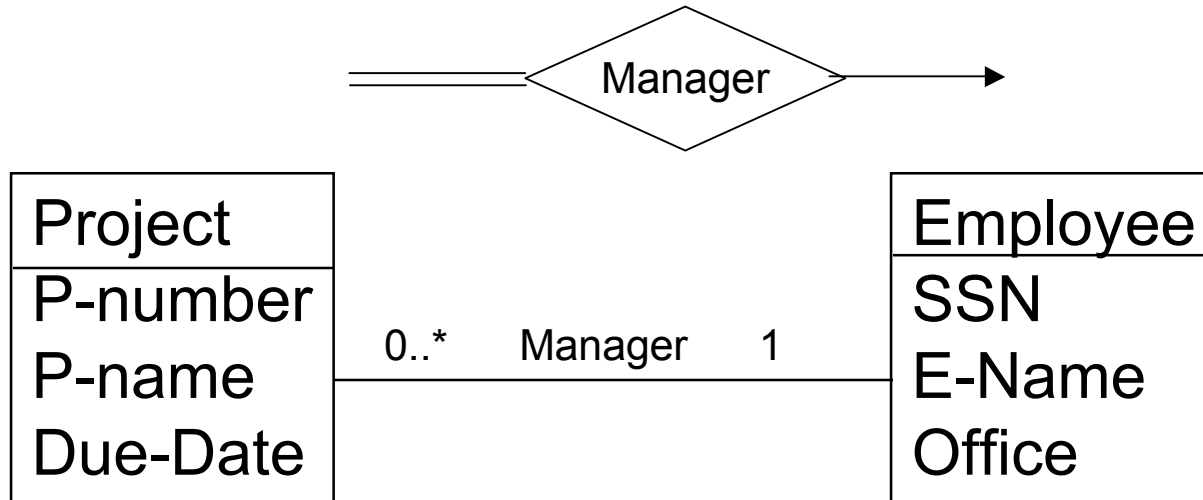
vs.

Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office)
 Manager (P-number, SSN)

Manager can manage a single project

2 managers could manage the same project

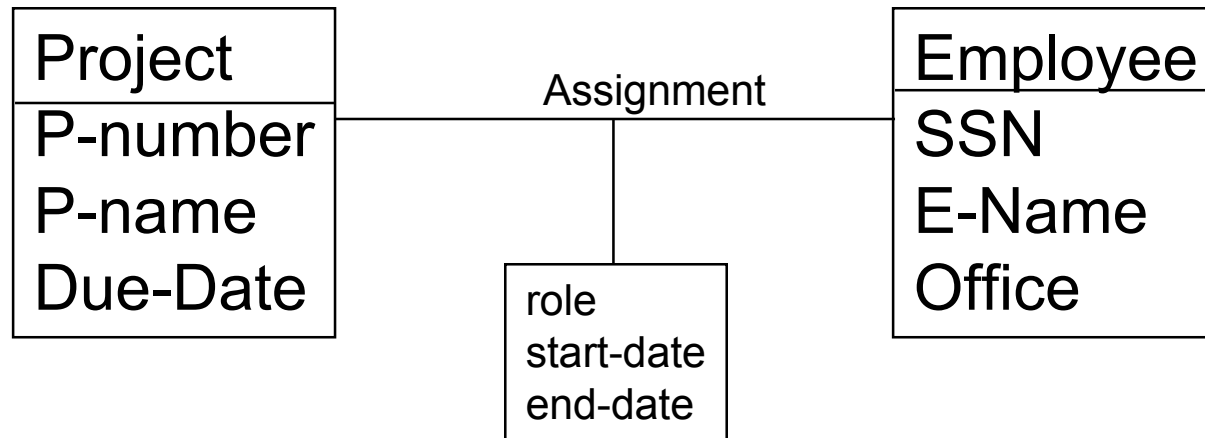
A project cannot exist without a manager



Project (P-number, P-name, Due-Date, **Manager**)
Employee (SSN, E-Name, Office)

4.3 Translating **one-to-many** relationships with *total participation*:

Create a foreign key for a 1-to-many relationship,
and forbid it from taking a NULL value

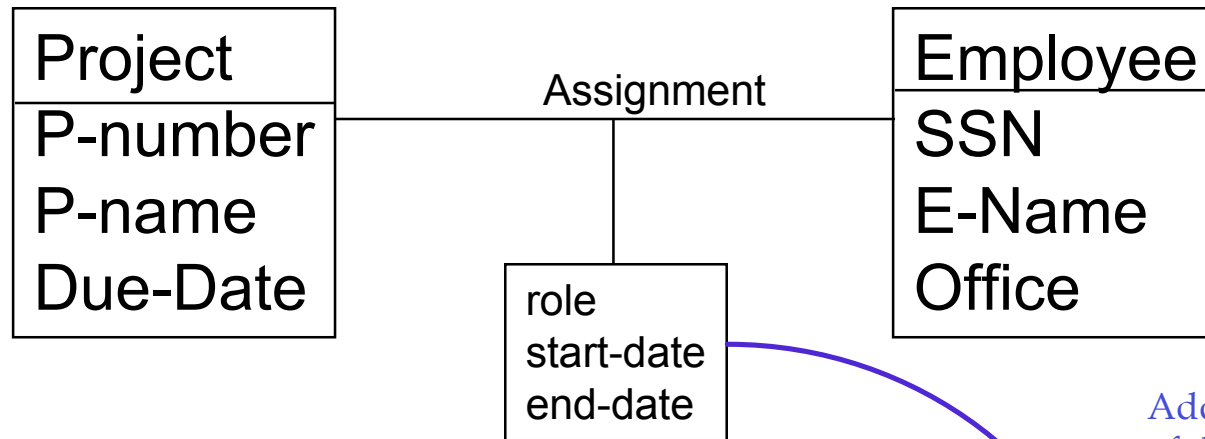


What do we do when a many-to-many relationship has an attribute?

Assignment (A-project, A-SSN)

Project (P-number, P-name, Due-Date)

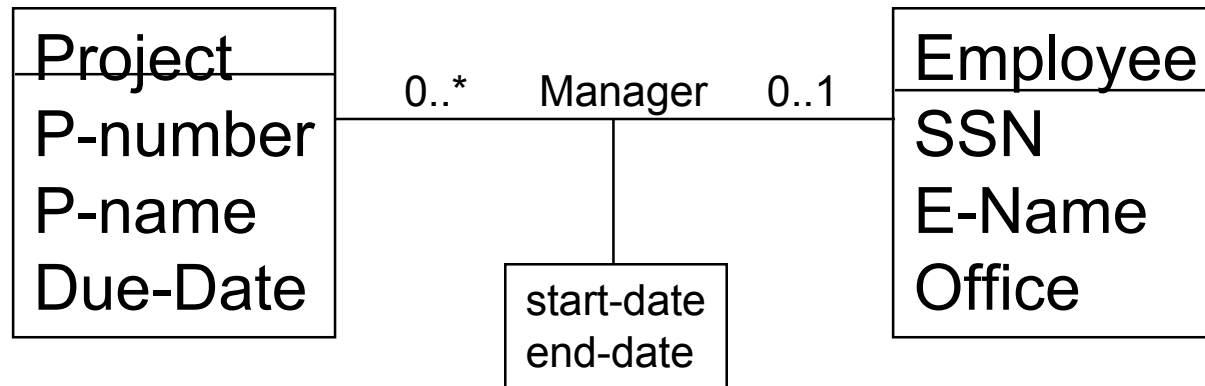
Employee (SSN, E-Name, Office)



What do we do when a many-to-many relationship has an attribute?

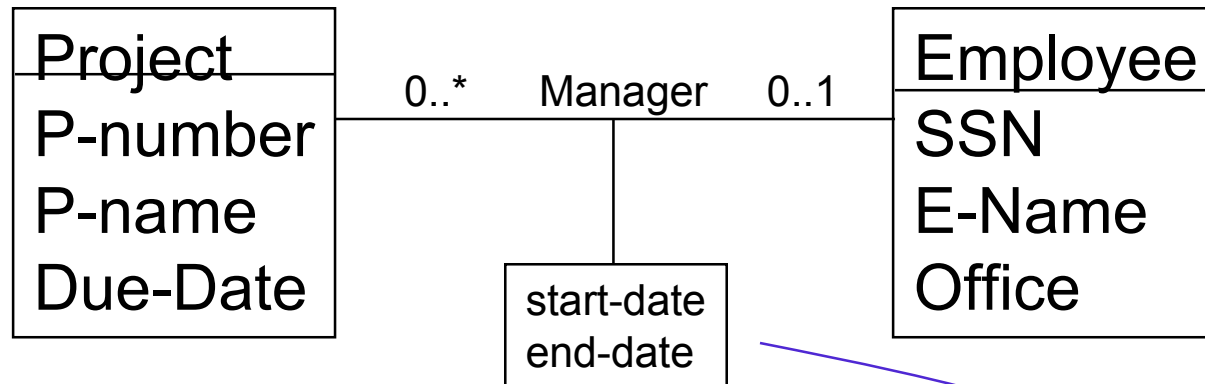
Add attributes of the relationship to the table representing the relationship

Assignment (A-project, A-SSN, role, start-date, end-date)
 Project (P-number, P-name, Due-Date)
 Employee (SSN, E-Name, Office)



What do we do when a 1-to-many relationship has an attribute?

Project (P-number, P-name, Due-Date, Manager)
 Employee (SSN, E-Name, Office)



Add attributes of the relationship to the table on 1-side of the relationship

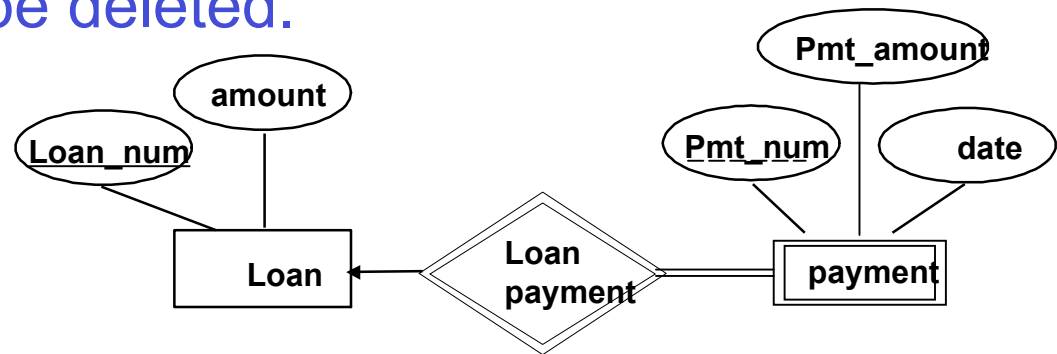
What do we do when a 1-to-many relationship has an attribute?

Project (P-number, P-name, Due-Date, Manager, start-date, end-date)

Employee (SSN, E-Name, Office)

Translating Weak Entity Sets

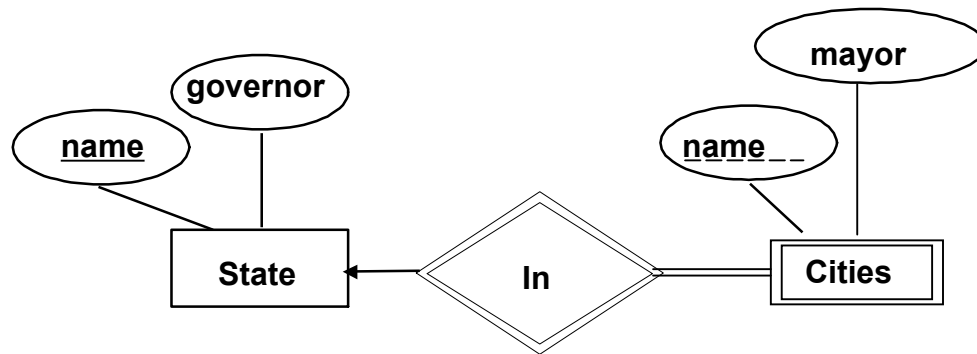
- Weak entity and identifying relationship are translated into a single table. Must include **key of strong entity**, as a foreign key.
- When the owner entity is deleted, all owned weak entities must also be deleted.



```
CREATE TABLE LoanPayment(  
  pmt_num      CHAR(5),  
  pmt_amt     REAL,  
  pmt_date    DATE,  
  loan_num    CHAR(11) NOT NULL,  
  PRIMARY KEY (loan_num,pmt_num),
```

```
FOREIGN KEY (ssn) REFERENCES Loan,  
ON DELETE CASCADE)
```

Reminder about Weak Relationships



- Can a city belong to 2 distinct states?
- Can two cities in the same state have the same name?
- Can two states have governors with the same name?
- Can 2 cities have the same name?

Translating ISA Relationship Sets

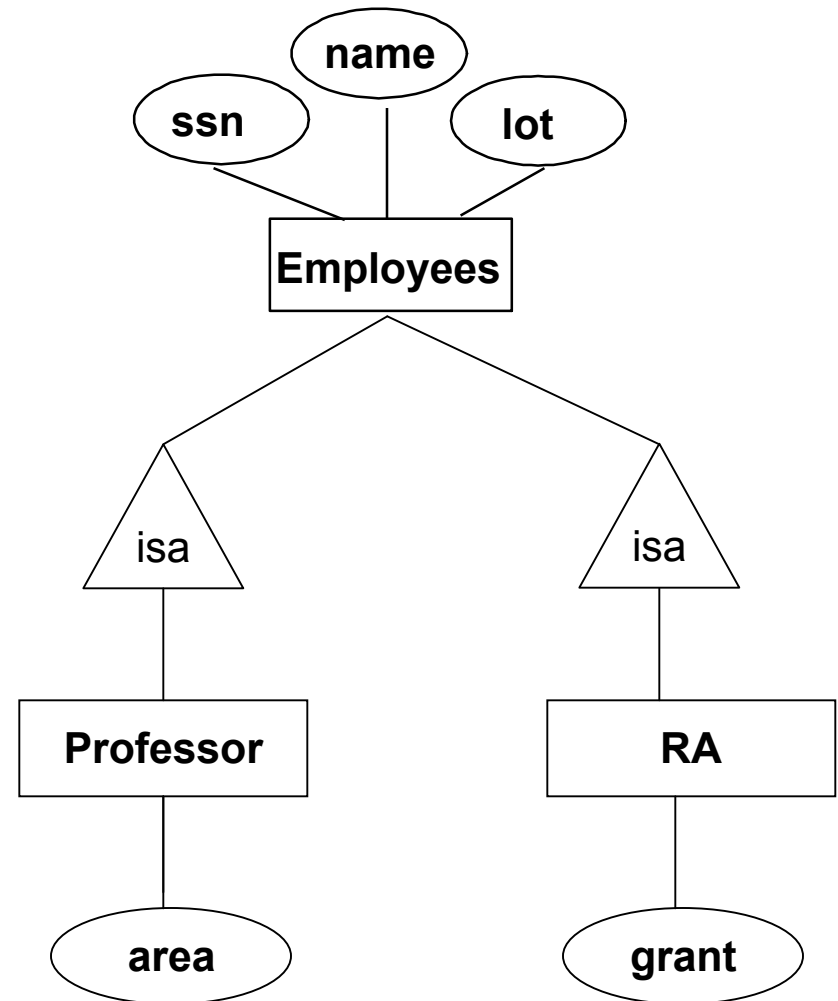
Some alternatives:

- 1) ER-style: For each entity E, create a relation that includes the key attributes of the *root* and any attributes of E

Employees(ssn, name, lot)

Professor(ssn, area)

RA(ssn, grant)



Translating ISA Relationship Sets

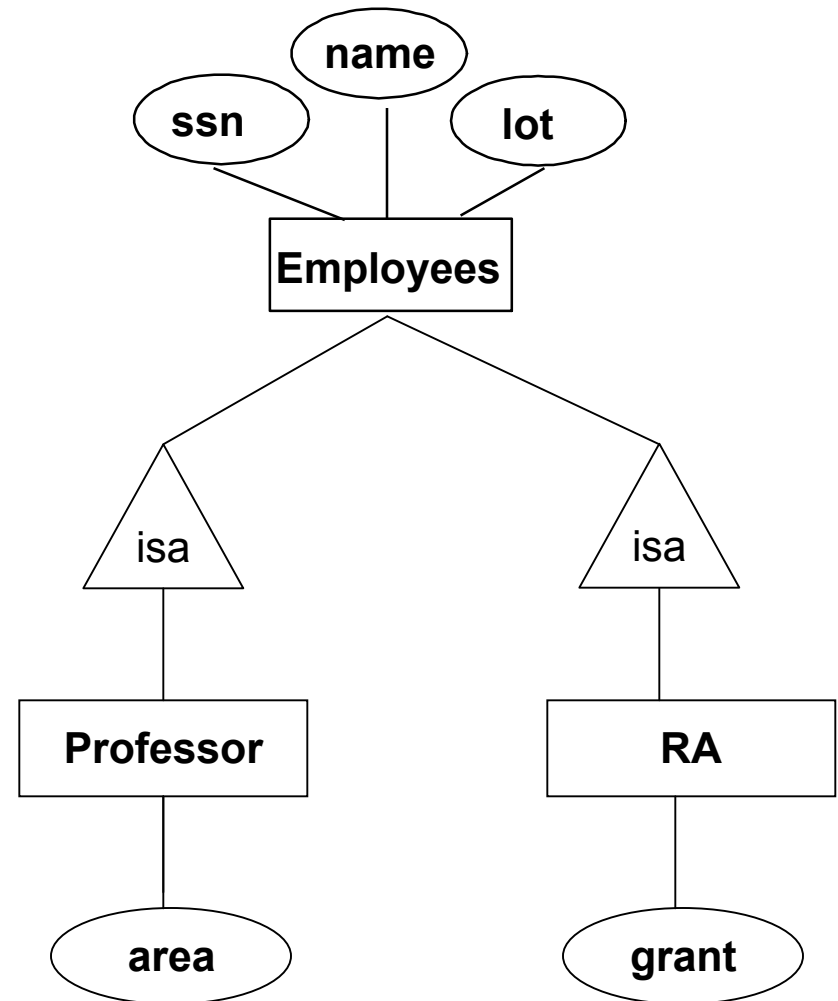
2) OO: For each entity E, create a relation that includes the key attributes of the *root* and any attributes of E

Employees(ssn, name, lot)

Professor(ssn, name, lot, area)

RA(ssn, name, lot, grant)

One relation per *class*

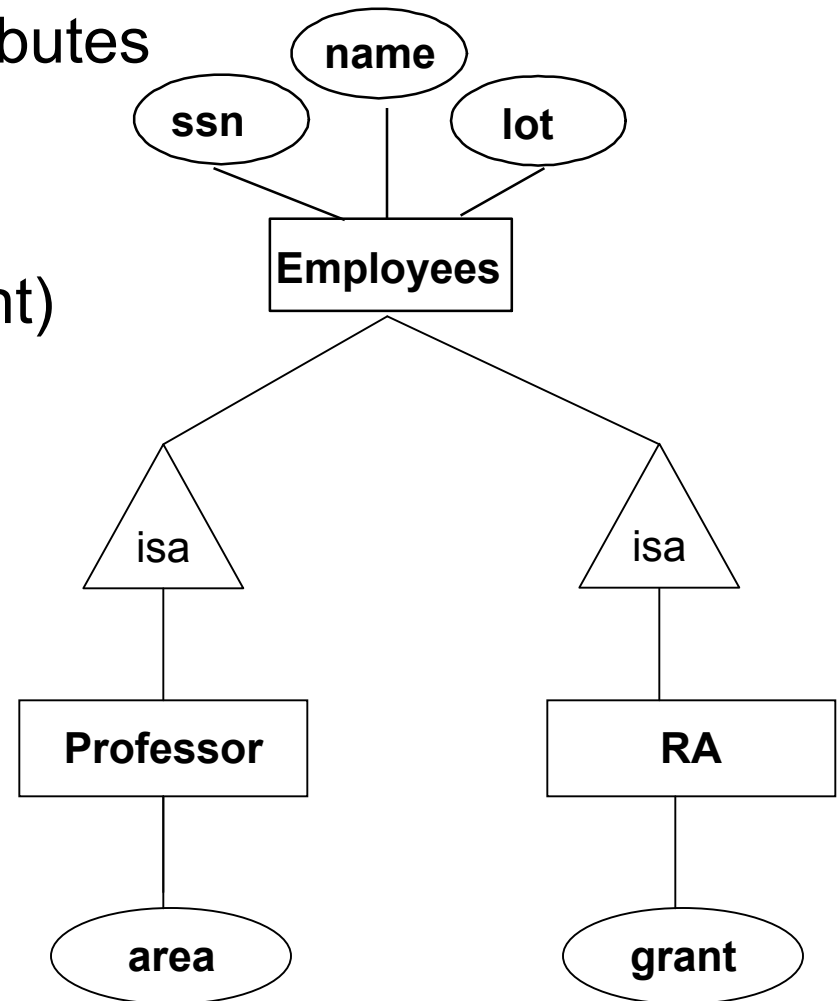


Translating ISA Relationship Sets

3) Using null values: Create a single relation with the union of all attributes in the hierarchy.

AllEmployee(ssn,name,lot,area,grant)

A tuple has NULL value in each attribute that is not defined for the corresponding entity



Translating ISA Relationship Sets

AllEmployee(ssn,name,lot,area,grant)

vs.

Employees(ssn,name,lot)

Professor(ssn,area)

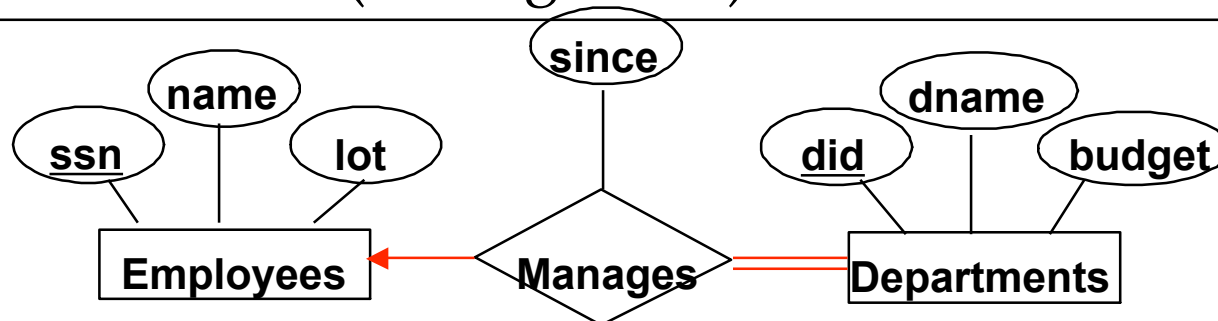
RA(ssn,grant)

What are the tradeoffs between these two alternatives?

Participation Constraints in SQL

- We can require an entity to be in a binary relationship using a foreign key which is required to be NOT NULL

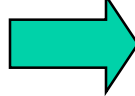
```
CREATE TABLE Department (  
  did          INTEGER,  
  dname        CHAR(20),  
  budget       REAL,  
  manager-ssn  CHAR(9) NOT NULL,  
  since        DATE,  
  PRIMARY KEY (did),  
  FOREIGN KEY (manager-ssn) REFERENCES Employee)
```



One Table/Entity can be split into two or more Tables/Entities with the same key

Employee
<u>id</u>
name
address
spouse
salary
deductions

could be replaced with:



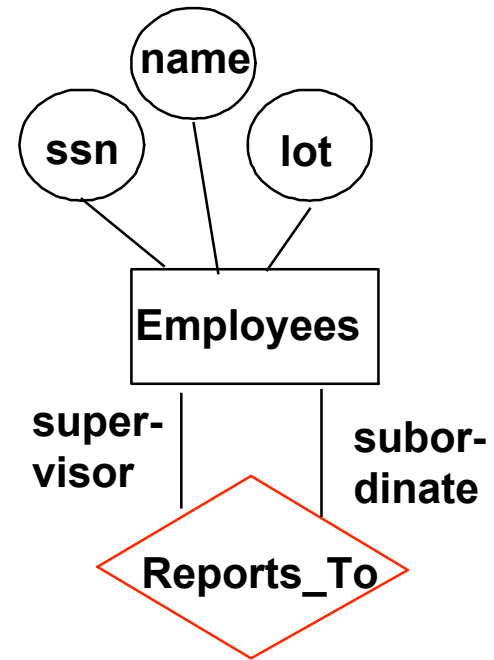
Employee-P
<u>id</u>
name
salary
deductions

Employee-S
<u>id</u>
address
spouse

note that the repeated key is also a foreign key

What are the implications of splitting the table?

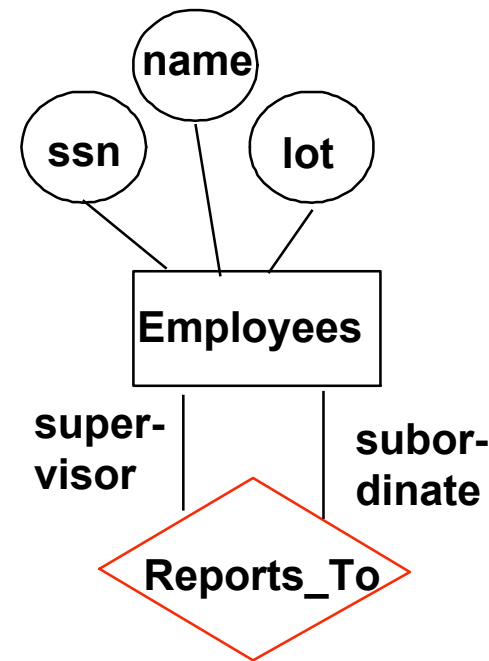
Now, translate this!



Now, translate this!

employees(ssn, name, lot)
supervises(ssn_super, ssn_sub)

- If one entity set is involved several times in a relationship, its key attributes each appear as many times as there are roles



Translation Steps: ER to Tables

1. Create table for each entity; include single-valued attributes. Choose key.
2. Create table for each weak entity type; include single-valued attributes. Include key of owner as a foreign key in the weak entity. Set key as foreign key of owner plus local, partial key.
3. For each 1:1 relationship, add a foreign key to one of the entities involved in the relationship (a foreign key to the other entity in the relationship).*
4. For each 1:N relationship, add a foreign key to the entity on the N-side of the relationship (to reference the entity on the 1-side of the relationship).*
5. For each M:N relationship, create a new table. Include a foreign key for each participant entity, in the relationship. The key for the new table is the set of all such foreign keys.
6. For each multi-valued attribute, construct a separate table. Repeat the key for the entity in this new table. It will serve as both the key for this table as well as a foreign key to the original table for the entity.

* Unless relationship has attributes. If it does, create new table for relationship.

This algorithm from Elmasri/Navathe, p. 174.

Summary

- Relations are simple data structures – easy to understand, accessible to a broad audience of end users
- Translate more complex ER design into a relational database schema
- Clean yet powerful declarative manipulation languages (more on this next class!)