# L12: ER modeling 5

CS3200 Database design (sp18 s2)

https://course.ccs.neu.edu/cs3200sp18s2/

2/22/2018

# Announcements!

- Keep bringing your name plates ☺
- Exam 1 discussion: questions on grading: Piazza, send to instructors only
- Project part 1 due tomorrow
- HW4 also extended to Friday next week
- Poll comments (next page)
  - We are slowing down even more
- Policies:
  - HW: soft graded, chance to learn, explore
  - Test: real grading, time-constrained to discourage cheating, similar to practice in class, HW or Piazza, multiple exams per year to reduce anxiety
- Outline
  - Continue with ER modeling and Normalization

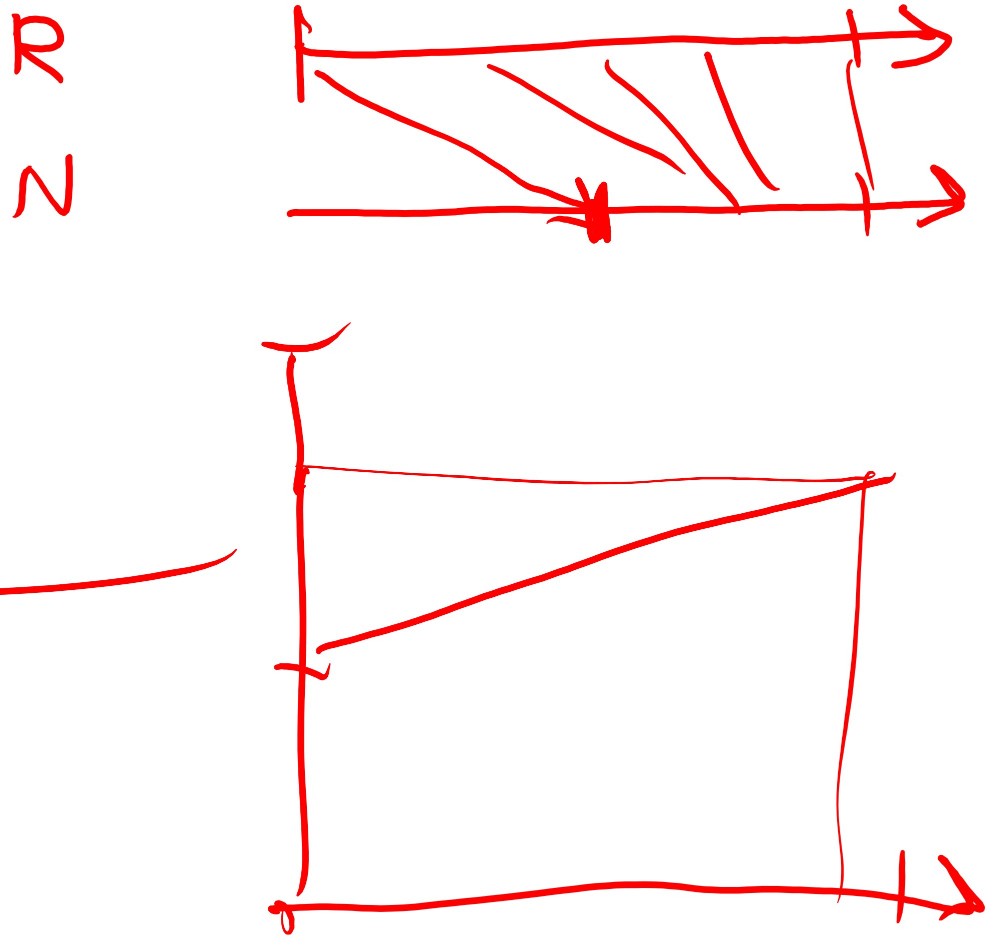
# From the poll: Speed and "What is going on"

| Row Labels | Count of Speed |
|---|---|
| 4 | 2 |
| 5 | 9 |
| 6 | 8 |
| 7 | 9 |
| 8 | 5 |
| 9 | 1 |
| **Grand Total** | **34** |

| Row Labels | Count of What |
|---|---|
| 4 | 1 |
| 5 | 3 |
| 6 | 5 |
| 7 | 7 |
| 8 | 12 |
| 9 | 4 |
| 10 | 2 |
| **Grand Total** | **34** |

# Exam 1 statistics

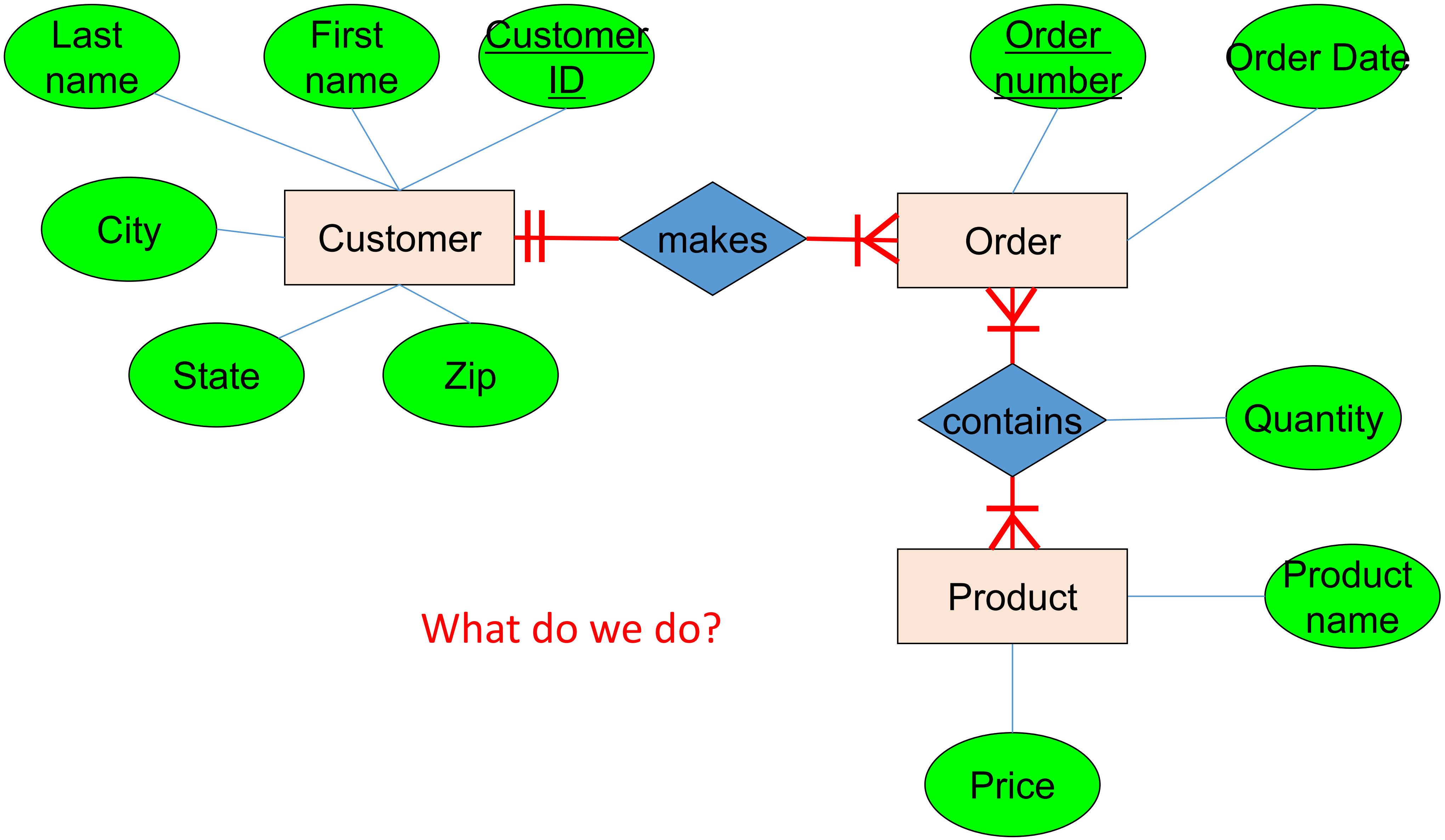
| | | |
|---|---|---|
| 5 | 0-2 | 4 |
| 6 | 2-4 | 5 |
| 7 | 4-6 | 6 |
| 8 | 6-8 | 10 |
| 9 | 8-10 | 7 |
| 10 | 10-12 | 11 |
| 11 | 12-14 | 10 |
| 12 | 14-16 | 7 |
| 13 | 16-18 | 5 |
| 14 | 18-20 | 4 |
| 15 | **Grand Total** | **69** |



# Quiz today

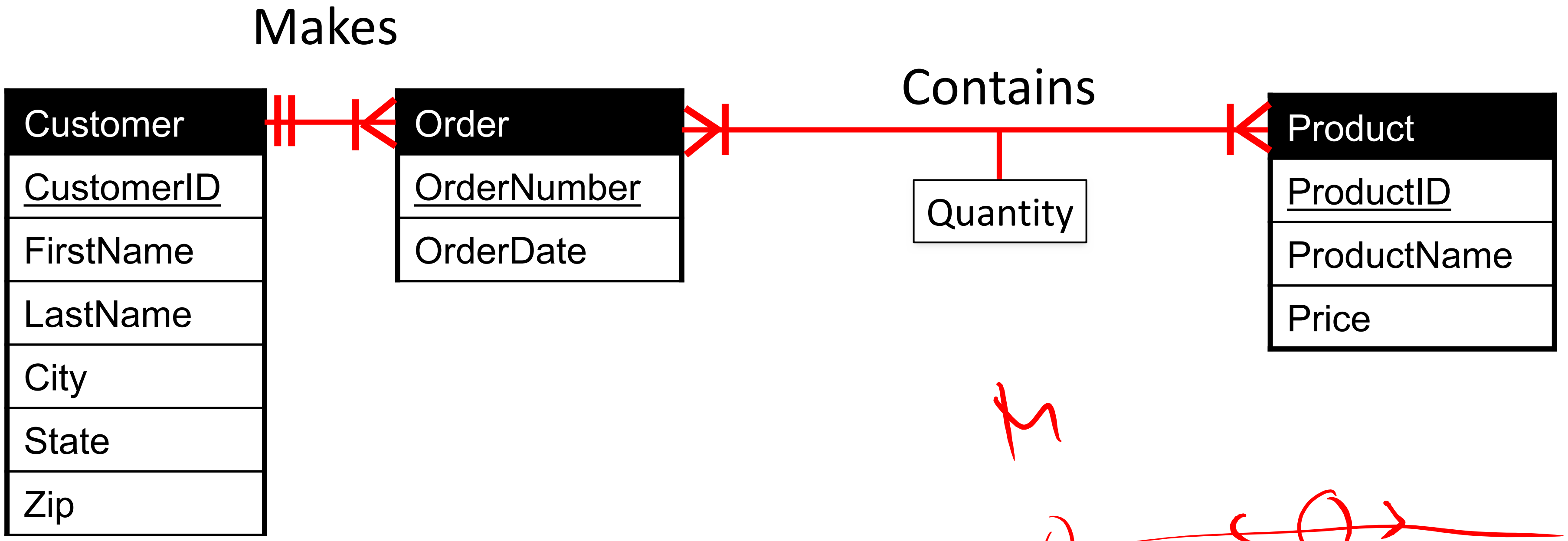
- Difference ERD vs. Relational Schema

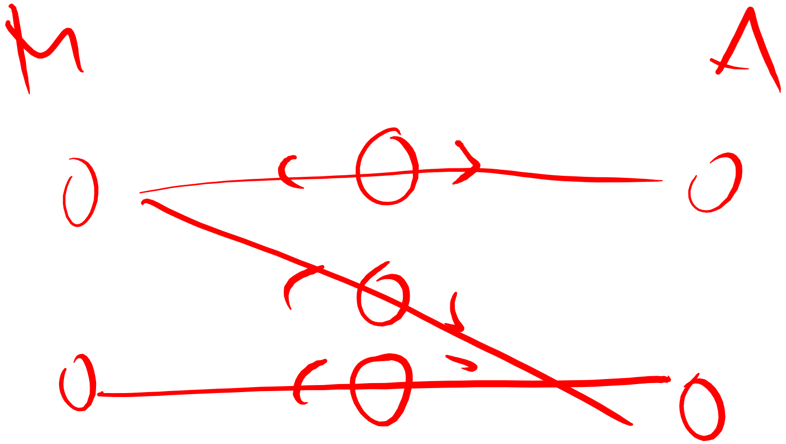
# ERD (Chen notation)




What do we do?

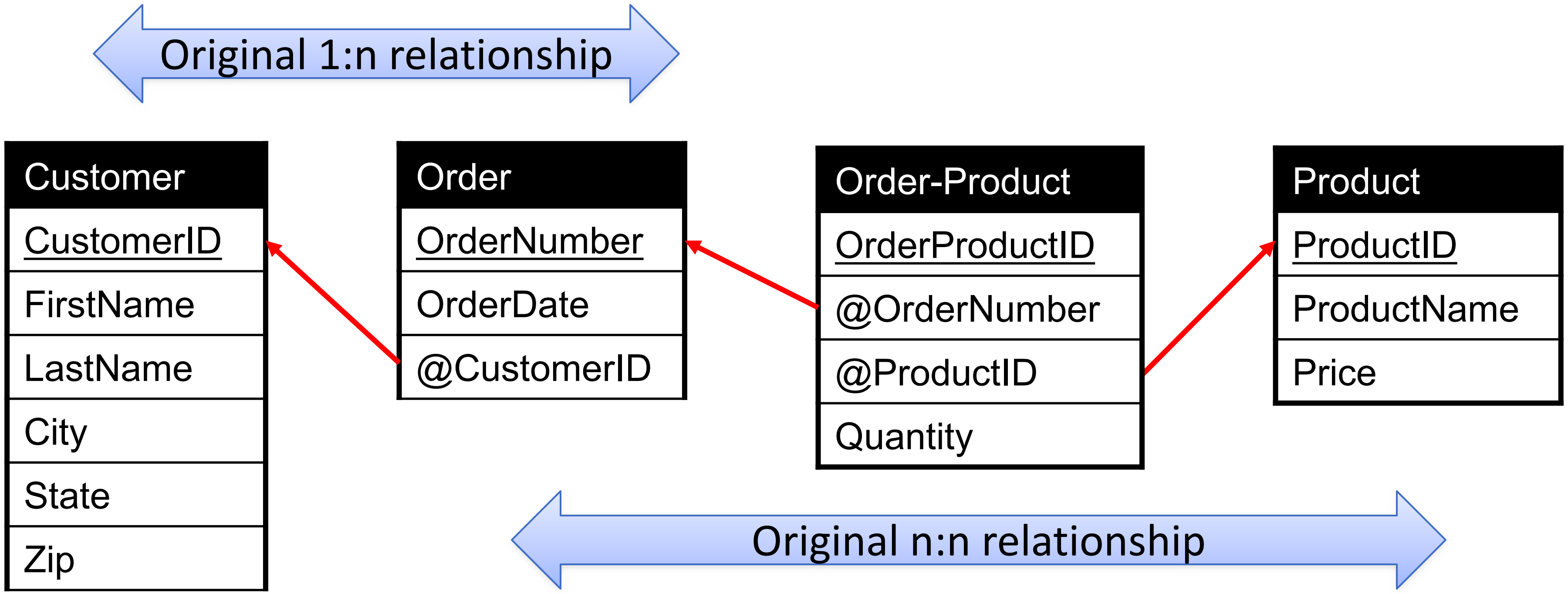
# ERD (UML / crow-feet notation)



What do we do?

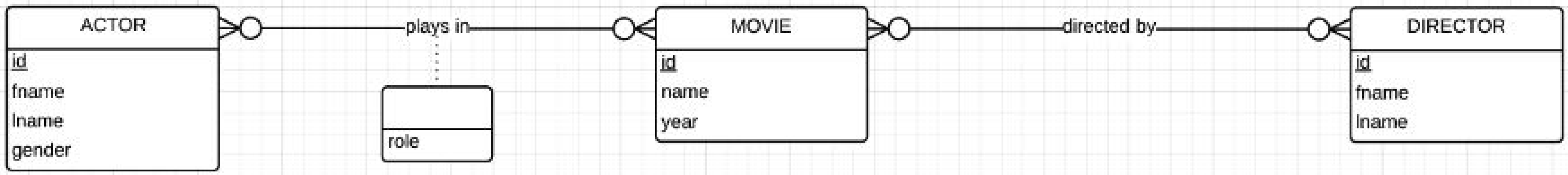
# Relational schema



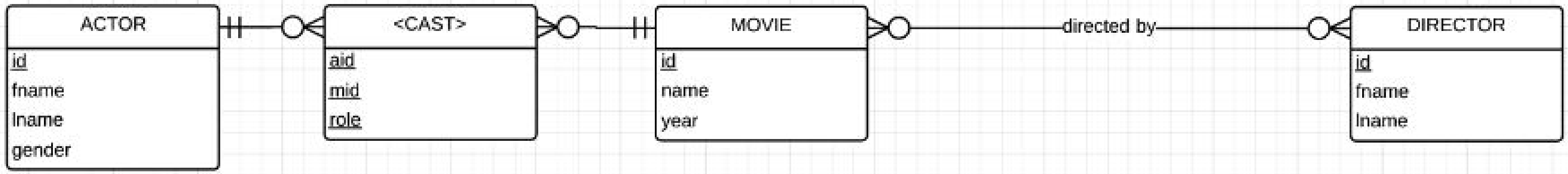

- Order-Product is a decomposed many-to-many relationship
  - Order-Product has a 1:n relationship with Order and Product
  - Now an order can have multiple products, and a product can be associated with multiple orders

# CAST in our IMDB movie database

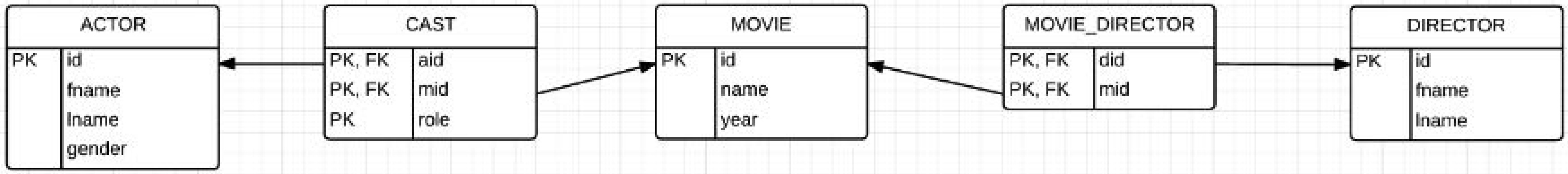
ER diagram: don't forget identifiers, but no FKs




ER diagram: CAST as associative entity can be justified

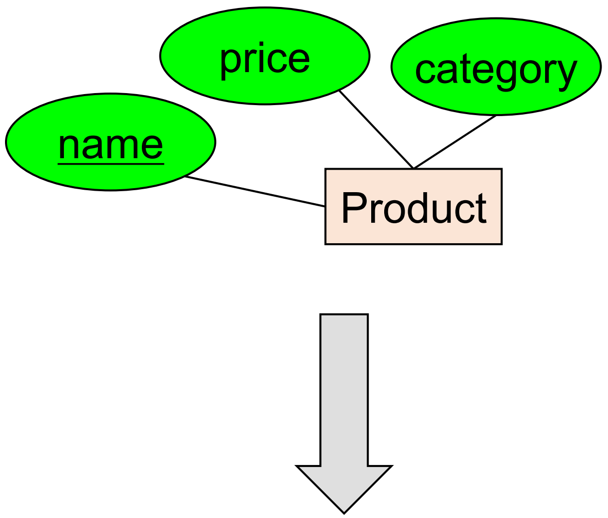



Relational schema: don't forget PKs and FKs



# From E/R Diagrams to Relational Schema

- An entity set becomes a relation (multiset of tuples / table)
  - Each tuple is one entity
  - Each tuple is composed of the entity's attributes, and has the same primary key

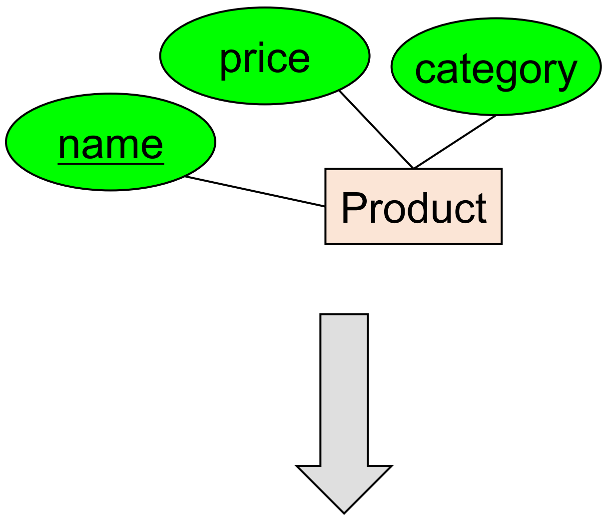



**Product**

| name | price | category |
|---|---|---|
| Gizmo1 | 99.99 | Camera |
| Gizmo2 | 19.99 | Edible |

# From E/R Diagrams to Relational Schema

```
CREATE TABLE Product(
  name     CHAR(50) PRIMARY KEY,
  price    DECIMAL(8,2),
  category VARCHAR(30)
)
```

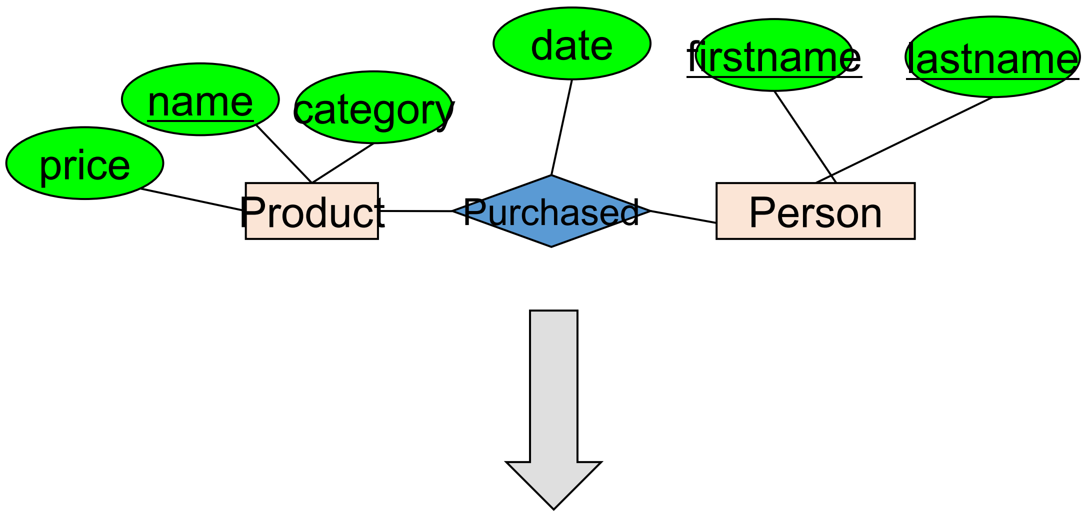



**Product**

| name | price | category |
|---|---|---|
| Gizmo1 | 99.99 | Camera |
| Gizmo2 | 19.99 | Edible |

# From E/R Diagrams to Relational Schema

- A many-to-many relation <u>between entity sets</u> $A_1$, …, $A_N$ also becomes a multiset of tuples / a table
  - Each row/tuple is one relation, i.e. one unique combination of entities $(a_1,…,a_N)$
  - Each row/tuple is
    - composed of the **union of the entity sets' keys**
    - has the entities' primary keys as foreign keys
    - has the union of the entity sets' keys as primary key

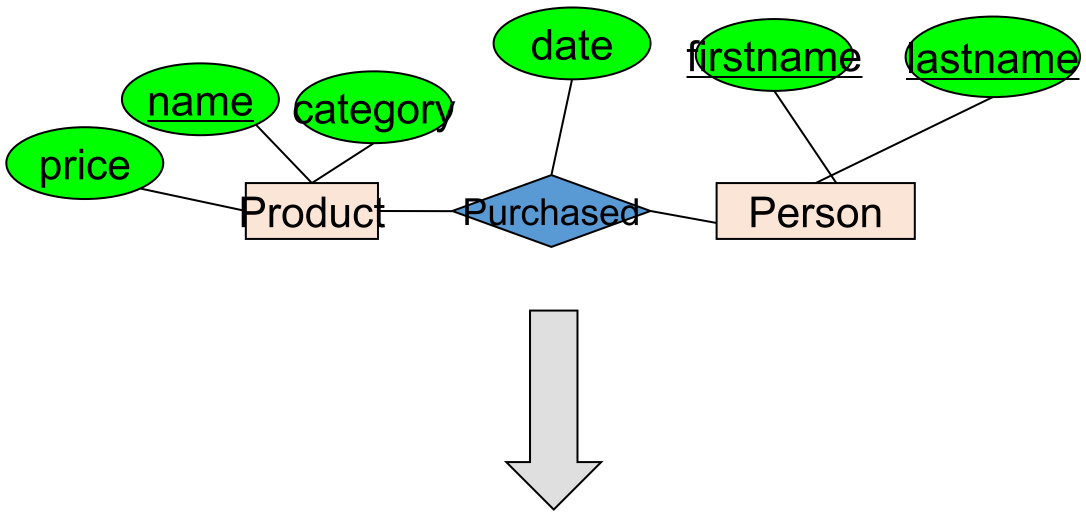



**Purchased**

| <u>name</u> | <u>firstname</u> | <u>lastname</u> | date |
|---|---|---|---|
| Gizmo1 | Bob | Joe | 01/01/15 |
| Gizmo2 | Joe | Bob | 01/03/15 |
| Gizmo1 | JoeBob | Smith | 01/05/15 |

# From E/R Diagrams to Relational Schema

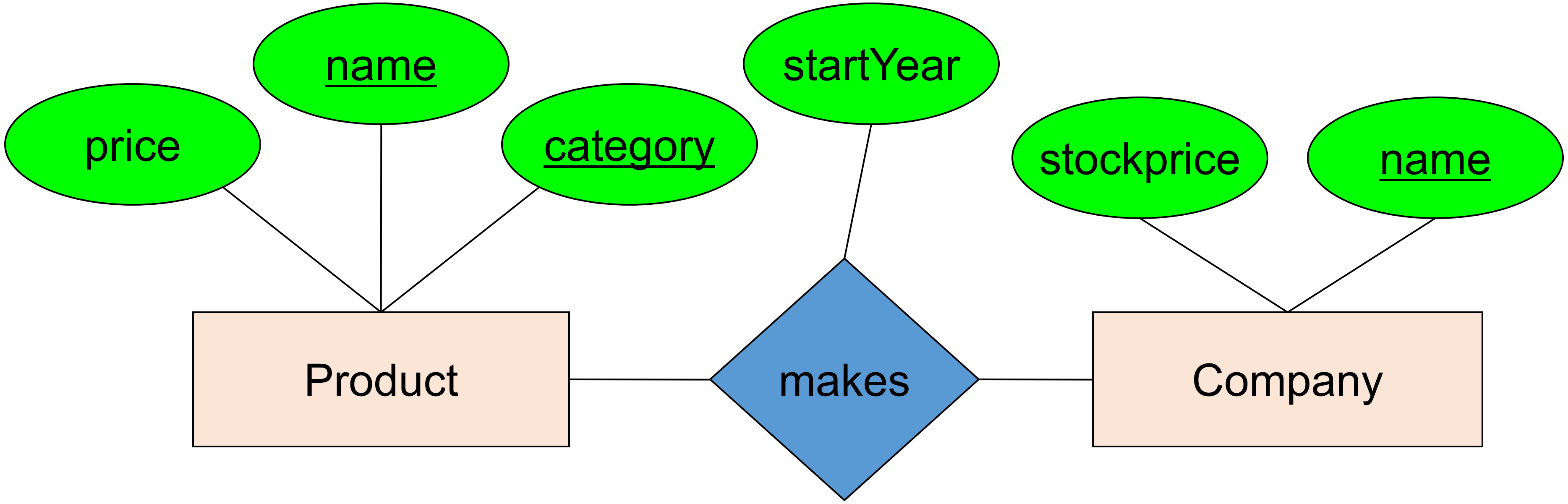
```
CREATE TABLE Purchased(
  name      CHAR(50),
  firstname CHAR(50),
  lastname  CHAR(50),
  date      DATE,
  PRIMARY KEY (name, firstname, lastname),
  FOREIGN KEY (name)
      REFERENCES Product,
  FOREIGN KEY (firstname, lastname)
      REFERENCES Person
)
```




**Purchased**

| name | firstname | lastname | date |
|---|---|---|---|
| Gizmo1 | Bob | Joe | 01/01/15 |
| Gizmo2 | Joe | Bob | 01/03/15 |
| Gizmo1 | JoeBob | Smith | 01/05/15 |

# Relationships to Relations



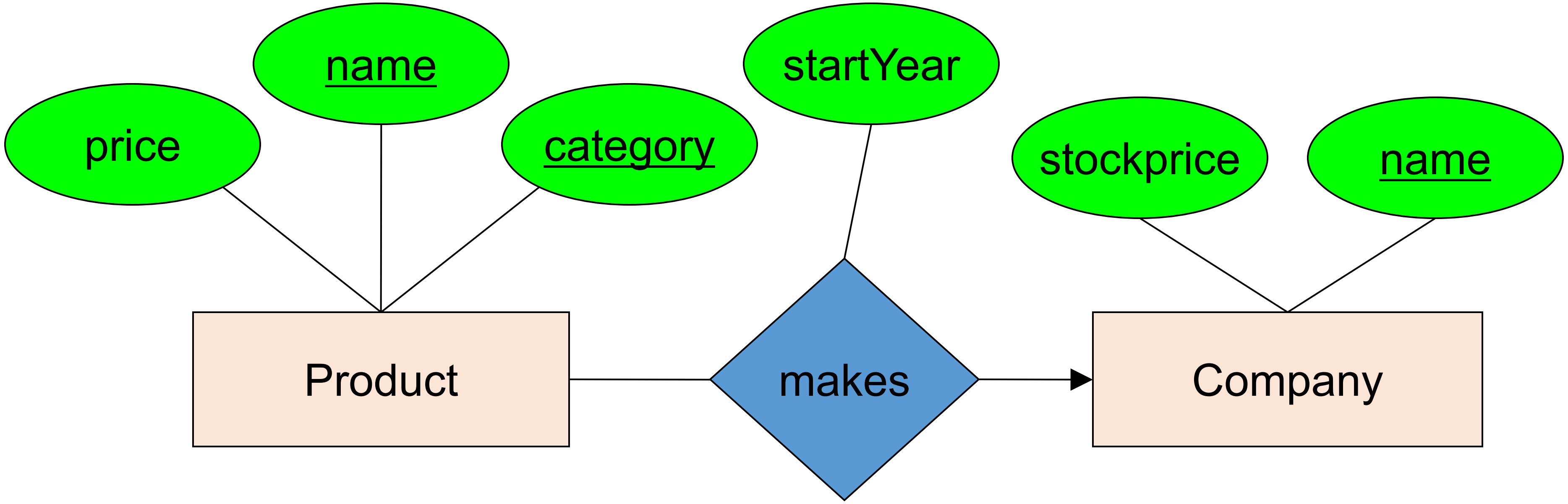

Watch out for attribute name conflicts

**Makes**(@productName, @category, @companyName, year)

Foreign keys

| ProductName | ProductCategory | CompanyName | startYear |
|---|---|---|---|
| Gizmo | Gadgets | GizmoWorks | 1963 |

# Relationships to Relations (with constraints)



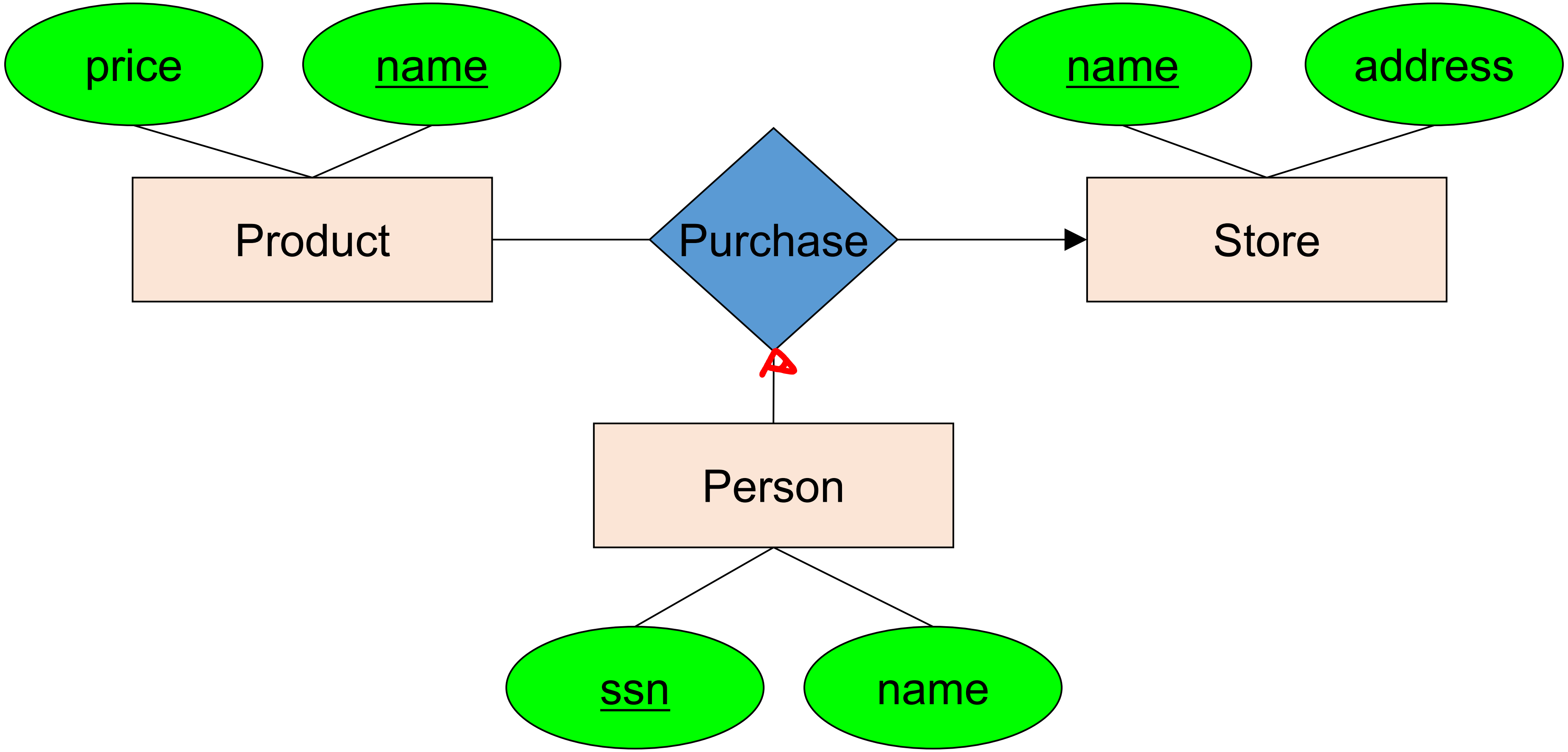

Only keep **Product** keys as primary key

**Makes**(<u>@productName, @category</u>, @companyName, year)

Better solution: get rid of **Makes**, modify **Product**:

| <u>prodName</u> | Category | Price | startYear | CompanyName |
|---|---|---|---|---|
| Gizmo | Gadgets | $19.99 | 1963 | GizmoWorks |

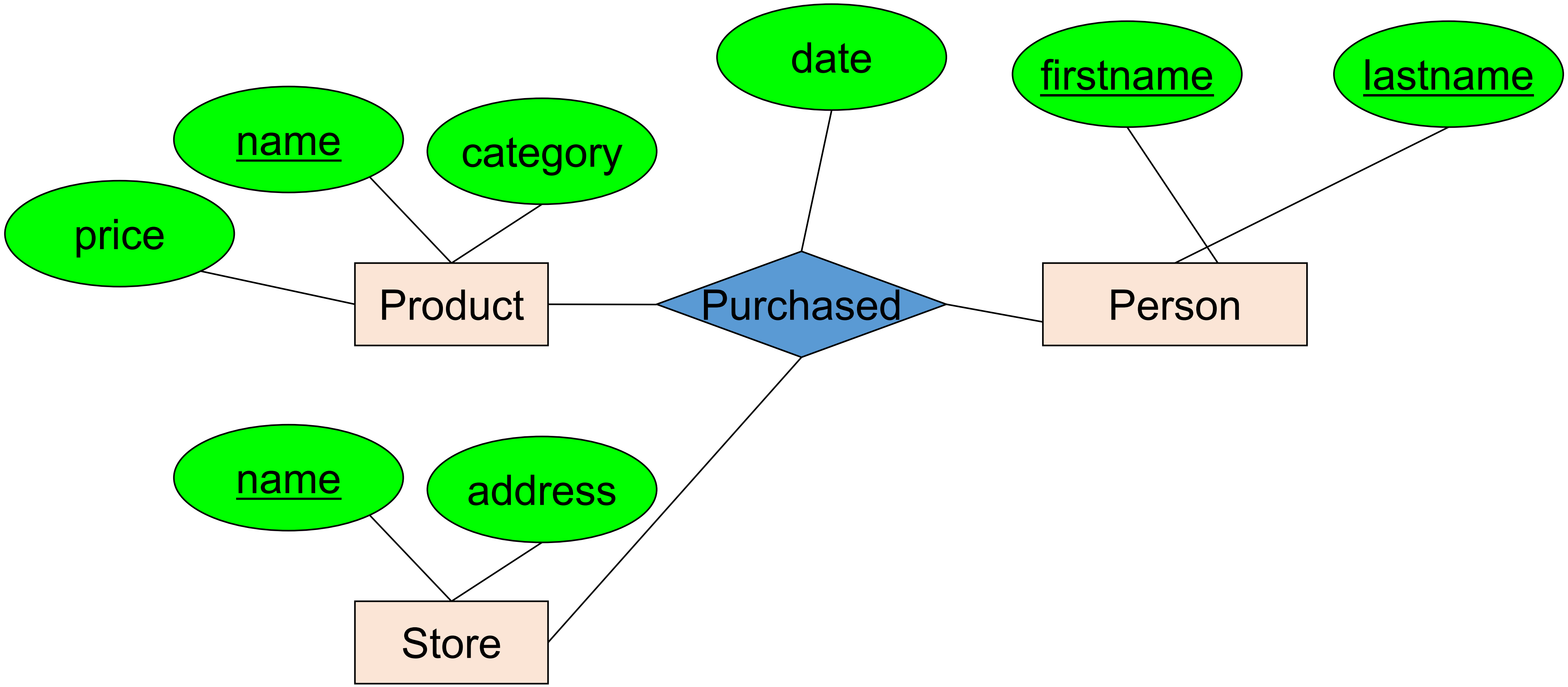
# Multi-way Relationships to Relations




**Purchase**(prodName, storeName, ssn)

# From E/R Diagram to Relational Schema

How do we represent this
as a relational schema?



# Relational Modeling: Entities & Attributes

# Relations

- A table consists of rows (records), and columns (attributes/fields)
- A relation is a named, two-dimensional table of data
- Six requirements for a table to qualify as a relation:
  1. The table must have a unique name.
  2. Columns (attributes) in tables must have unique names
  3. Every attribute value must be atomic (not multivalued, not composite)
  4. Every row must be unique (can't have two rows with exactly the same values for all their columns)
  5. The order of the columns must be irrelevant
     1. A(id, name) vs. A(name, id)
  6. The order of the rows must be irrelevant

# Mapping ER Models To Relations

- Relations (tables) correspond to entity types and to many-to-many relationship types
- Rows correspond to entity instances and to many-to-many relationship instances
- Columns correspond to attributes

- relation (in relational database) ≠ relationship (in E-R model)

EMPLOYEE1

| Emp_ID | Name | Dept_Name | Salary |
|---|---|---|---|
| 100 | Margaret Simpson | Marketing | 48,000 |
| 140 | Allen Beeton | Accounting | 52,000 |
| 110 | Chris Lucero | Info Systems | 43,000 |
| 190 | Lorenzo Davis | Finance | 55,000 |
| 150 | Susan Martin | Marketing | 42,000 |

# Relation Notation

- Here are two common notations for describing relations:

- Text statements – RELATION_NAME(attributes)
  - CUSTOMER(Customer_ID, Name, Address, City, State)
  - ORDER(Order_ID, Order_Date, Product_ID)

- Graphical notation:

# Key Fields

- Keys are special fields used to uniquely identify relations
- Primary keys are unique identifiers of the relation in question
  - Primary keys guarantee that all rows are unique
  - Examples
    - Employee ID numbers
    - Social security numbers
    - E-mail addresses
- Foreign keys are identifiers that refer to other primary keys
  - Useful as references
- Keys can be simple (single field) or composite (multiple fields)
- Keys are often used as indexes to speed up user queries

# Mapping Regular Entities to Relations

- Simple attributes:
  - E-R attributes map directly onto the relation

- Composite attributes:
  - Use only their simple, component attributes

- Multivalued Attribute
  - Becomes a separate relation with a foreign key taken from the superior entity

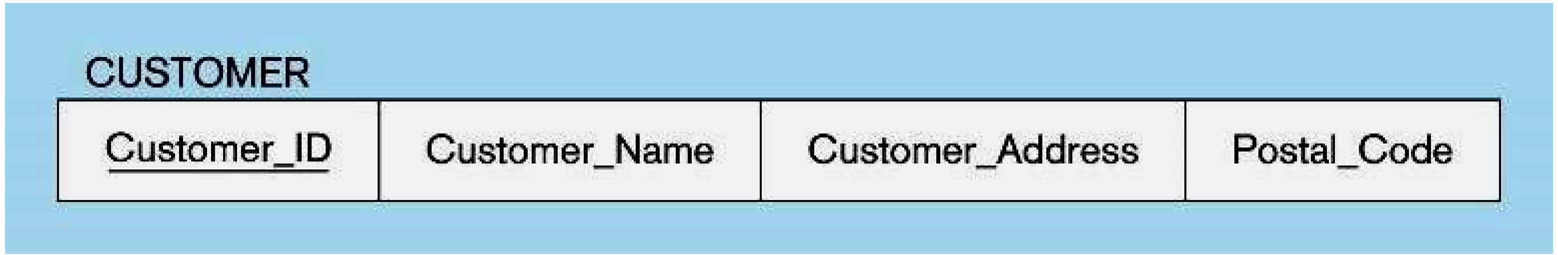
# Map Simple ER Attributes Directly Onto Relation

CUSTOMER entity type with simple attributes

# Map Simple ER Attributes Directly Onto Relation

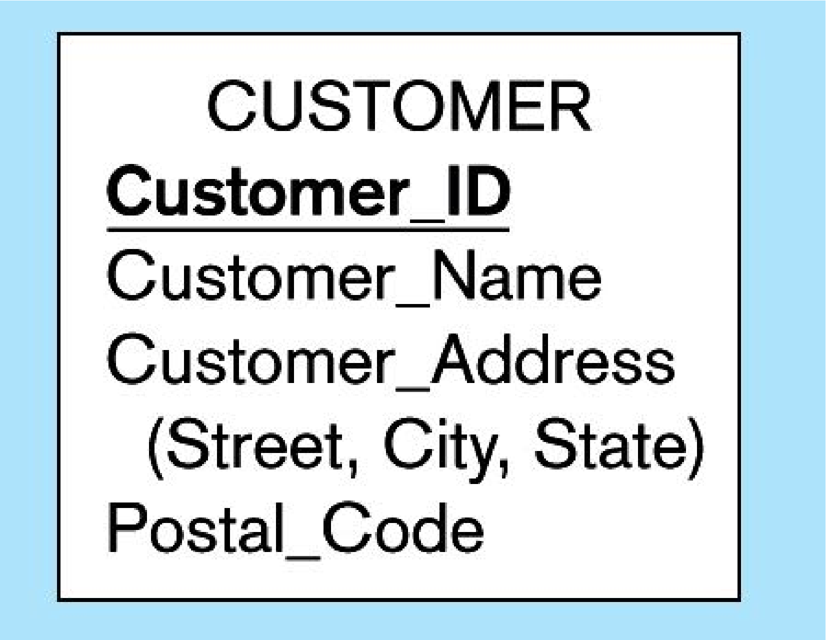
CUSTOMER entity type with simple attributes


CUSTOMER relation



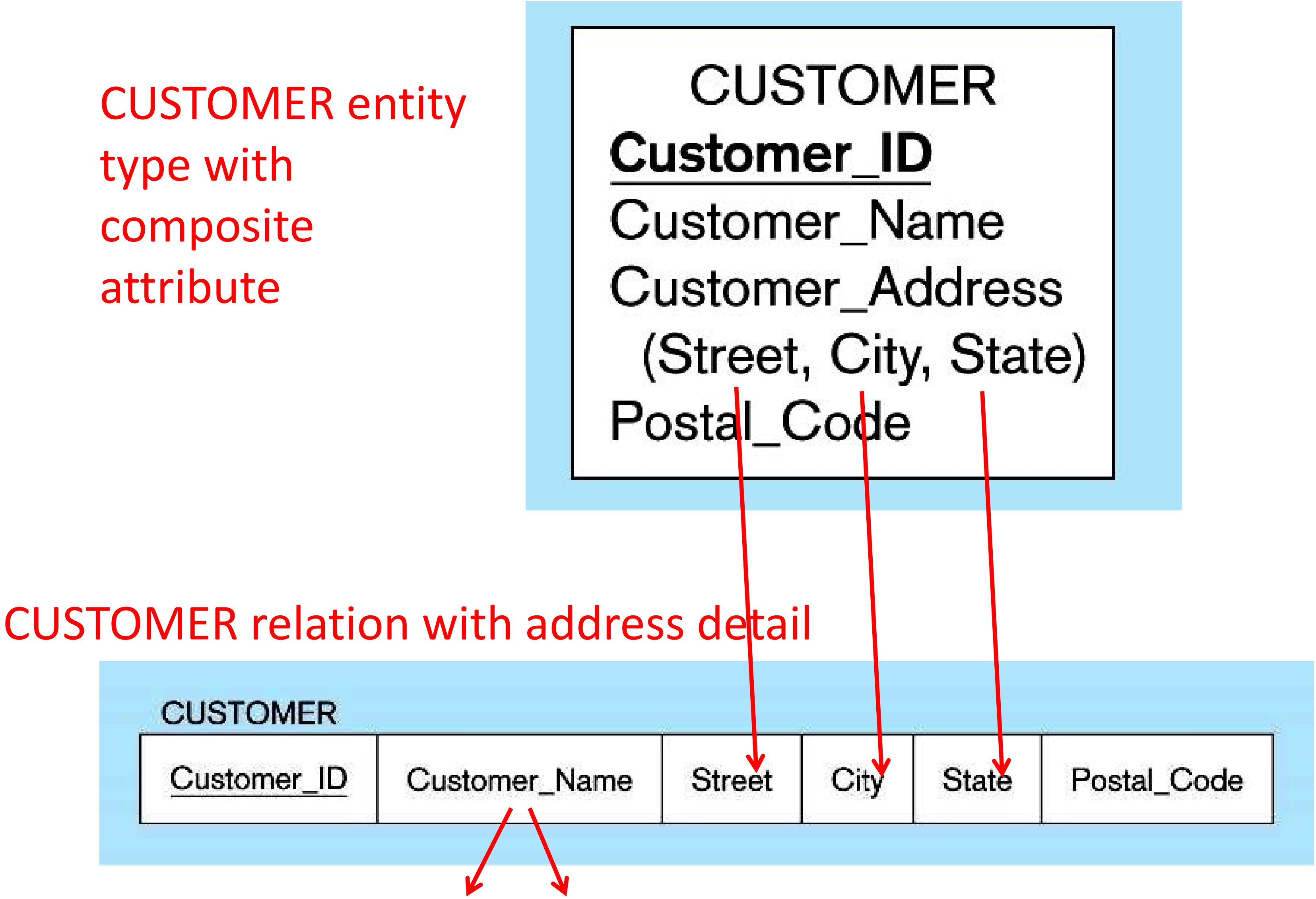
# Example: Mapping A Composite Attribute

CUSTOMER entity type with composite attribute

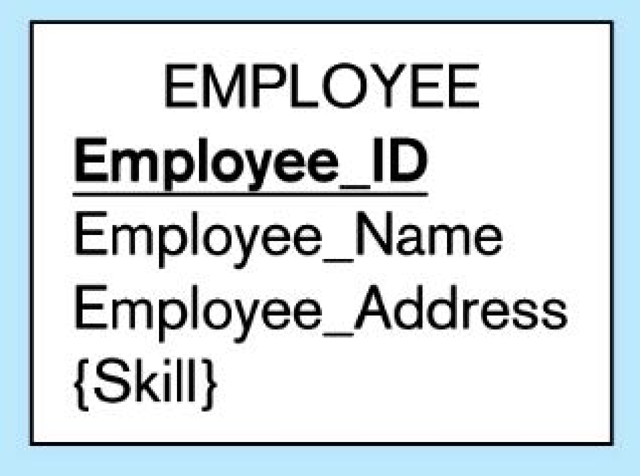


# Example: Mapping A Composite Attribute

CUSTOMER entity type with composite attribute

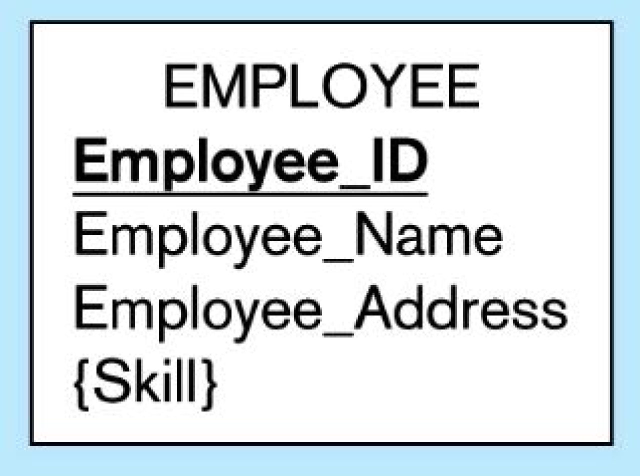


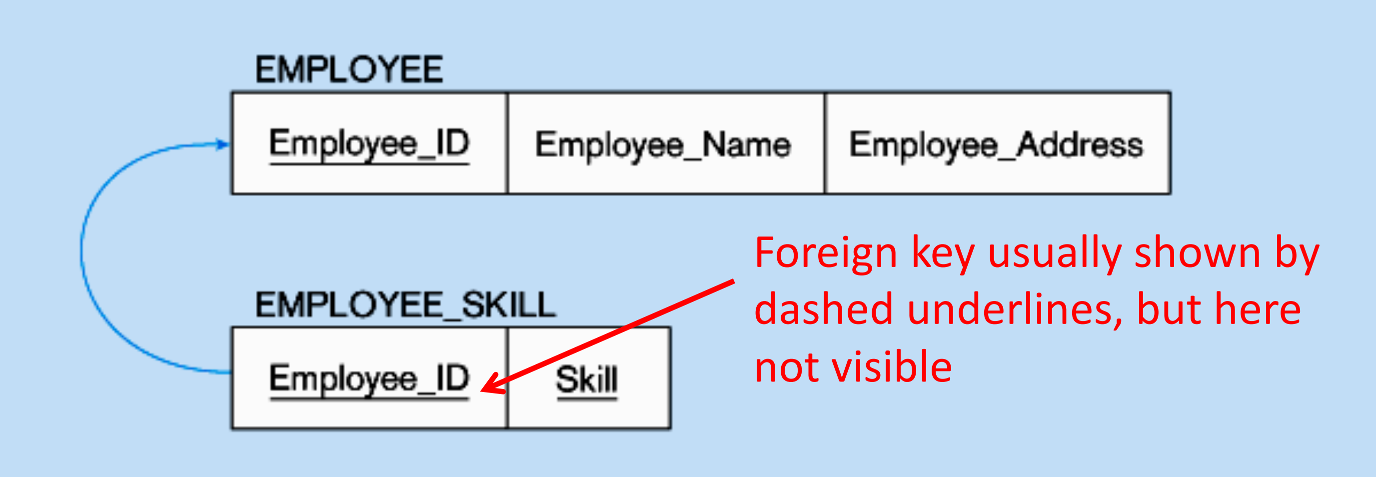
# Example: Mapping A Multivalued Attribute



# Example: Mapping A Multivalued Attribute

Multivalued Attribute becomes a separate relation with foreign key




1–to–many relationship between original entity and new relation

# Relational Modeling: Relationships

# Mapping Binary Relationships

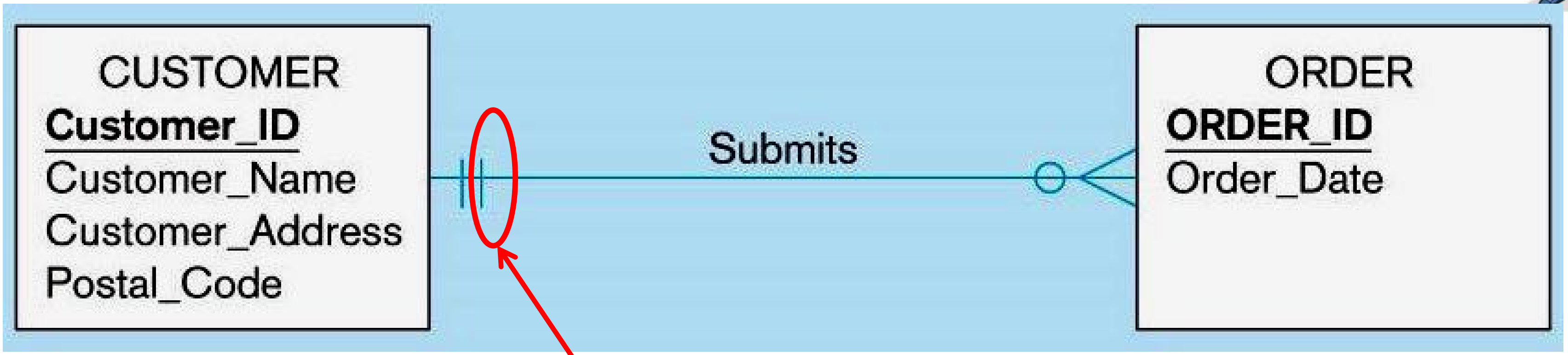
- 1. **One-to-Many**: Primary key on the one side becomes a foreign key on the many side

- 2. **Many-to-Many**: Create a new relation with the primary keys of the two entities as its primary key

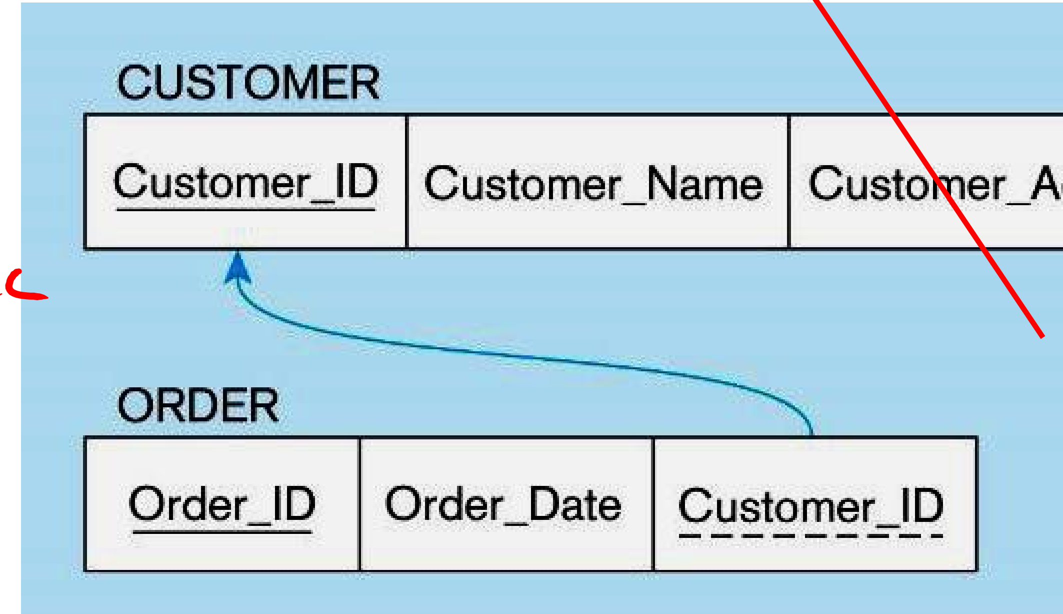
- 3. **One-to-One**: Primary key on the mandatory side becomes a foreign key on the optional side

# 1) Mapping a 1:M Relationship

# 1) Mapping a 1:M Relationship




Rule: Primary key on the one side becomes a foreign key on the many side




Again, no null value in the foreign key…this is because of the mandatory minimum cardinality

# Referential integrity constraints and NULL



SQLQuery12.sql - j..._Assignment (1220))* | SQLQuery11.sql - (...8OR\Wolfgang (59))

```
select *
from product;
```

```
---------------------------
-- Create the tables
---------------------------
create table Company (
    CName char(20) PRIMARY KEY,
    StockPrice int,
    Country char(20) );

create table Product (
    PName char(20),
    Price decimal(9, 2),
    Category char(20),
    Manufacturer char(20),
PRIMARY KEY (PName),
FOREIGN KEY (Manufacturer) REFERENCES Company(CName) );
```

Results | Messages

| | PName | Price | Category | Manufacturer |
|---|---|---|---|---|
| 1 | Gizmo | 19.99 | Gadgets | GizmoWorks |
| 2 | MultiTouch | 203.99 | Household | Hitachi |
| 3 | PowerGizmo | 29.99 | Gadgets | GizmoWorks |
| 4 | SingleTouch | 149.99 | Photography | Canon |

# Referential integrity constraints and NULL



SQLQuery12.sql - j..._Assignment (1220))* | SQLQuery11.sql - (...8OR\Wolfgang (59))

```sql
select *
from product;

insert into product values('hallo', 10, 'Gadgets', NULL);
```

```sql
----------------------------
-- Create the tables
----------------------------
create table Company (
    CName char(20) PRIMARY KEY,
    StockPrice int,
    Country char(20) );

create table Product (
    PName char(20),
    Price decimal(9, 2),
    Category char(20),
    Manufacturer char(20),
PRIMARY KEY (PName),
FOREIGN KEY (Manufacturer) REFERENCES Company(CName) );
```

Results | Messages

| | PName | Price | Category | Manufacturer |
|---|---|---|---|---|
| 1 | Gizmo | 19.99 | Gadgets | GizmoWorks |
| 2 | MultiTouch | 203.99 | Household | Hitachi |
| 3 | PowerGizmo | 29.99 | Gadgets | GizmoWorks |
| 4 | SingleTouch | 149.99 | Photography | Canon |

# Referential integrity constraints and NULL



SQLQuery12.sql - j..._Assignment (1220))* | SQLQuery11.sql - (...8OR\Wolfgang (59))

```sql
select *
from product;

insert into product values('hallo', 10, 'Gadgets', NULL);

select *
from product;

delete from product
where manufacturer is null;
```

100 %

Results | Messages

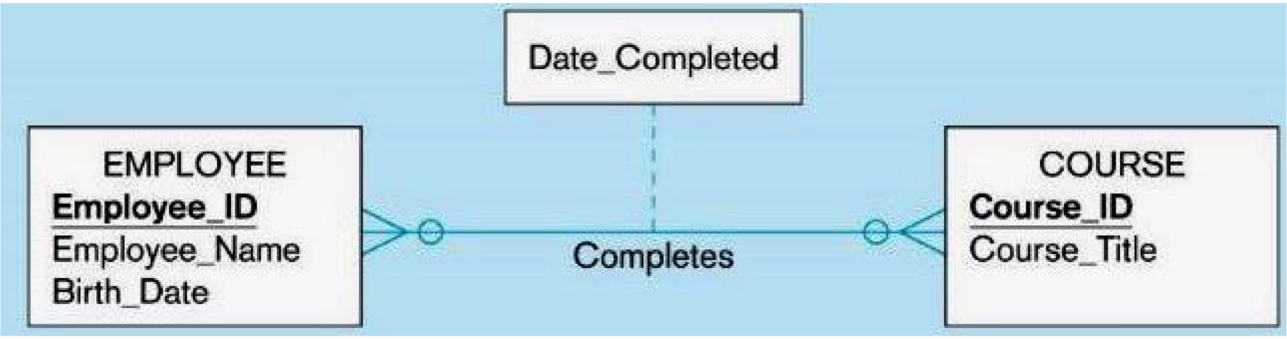
| | PName | Price | Category | Manufacturer |
|---|---|---|---|---|
| 1 | Gizmo | 19.99 | Gadgets | GizmoWorks |
| 2 | MultiTouch | 203.99 | Household | Hitachi |
| 3 | PowerGizmo | 29.99 | Gadgets | GizmoWorks |
| 4 | SingleTouch | 149.99 | Photography | Canon |

| | PName | Price | Category | Manufacturer |
|---|---|---|---|---|
| 1 | Gizmo | 19.99 | Gadgets | GizmoWorks |
| 2 | hallo | 10.00 | Gadgets | NULL |
| 3 | MultiTouch | 203.99 | Household | Hitachi |
| 4 | PowerGizmo | 29.99 | Gadgets | GizmoWorks |
| 5 | SingleTouch | 149.99 | Photography | Canon |

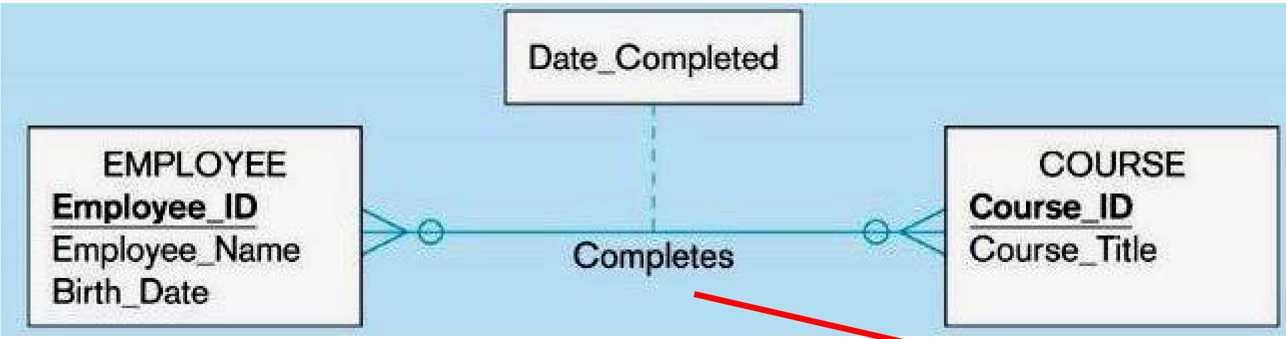
```sql
---------------------------
-- Create the tables
---------------------------
create table Company (
    CName char(20) PRIMARY KEY,
    StockPrice int,
    Country char(20) );

create table Product (
    PName char(20),
    Price decimal(9, 2),
    Category char(20),
    Manufacturer char(20),
PRIMARY KEY (PName),
FOREIGN KEY (Manufacturer) REFERENCES Company(CName) );
```

```sql
create table Product (
    PName char(20),
    Price decimal(9, 2),
    Category char(20),
    Manufacturer char(20) not null,
PRIMARY KEY (PName),
FOREIGN KEY (Manufacturer) REFERENCES Company(CName) );
```

# 2) Mapping An M:N Relationship

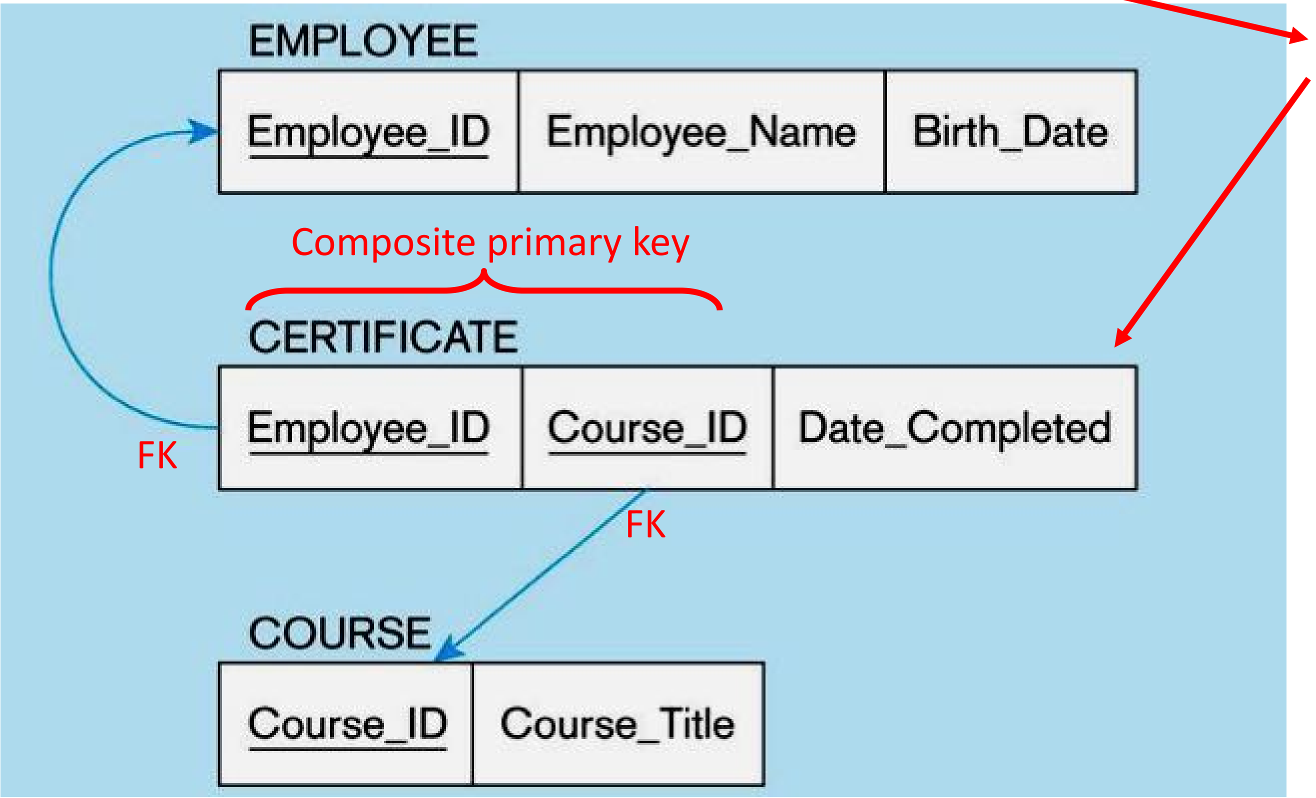


# 2) Mapping An M:N Relationship




The *Completes* relationship will need to become a separate relation

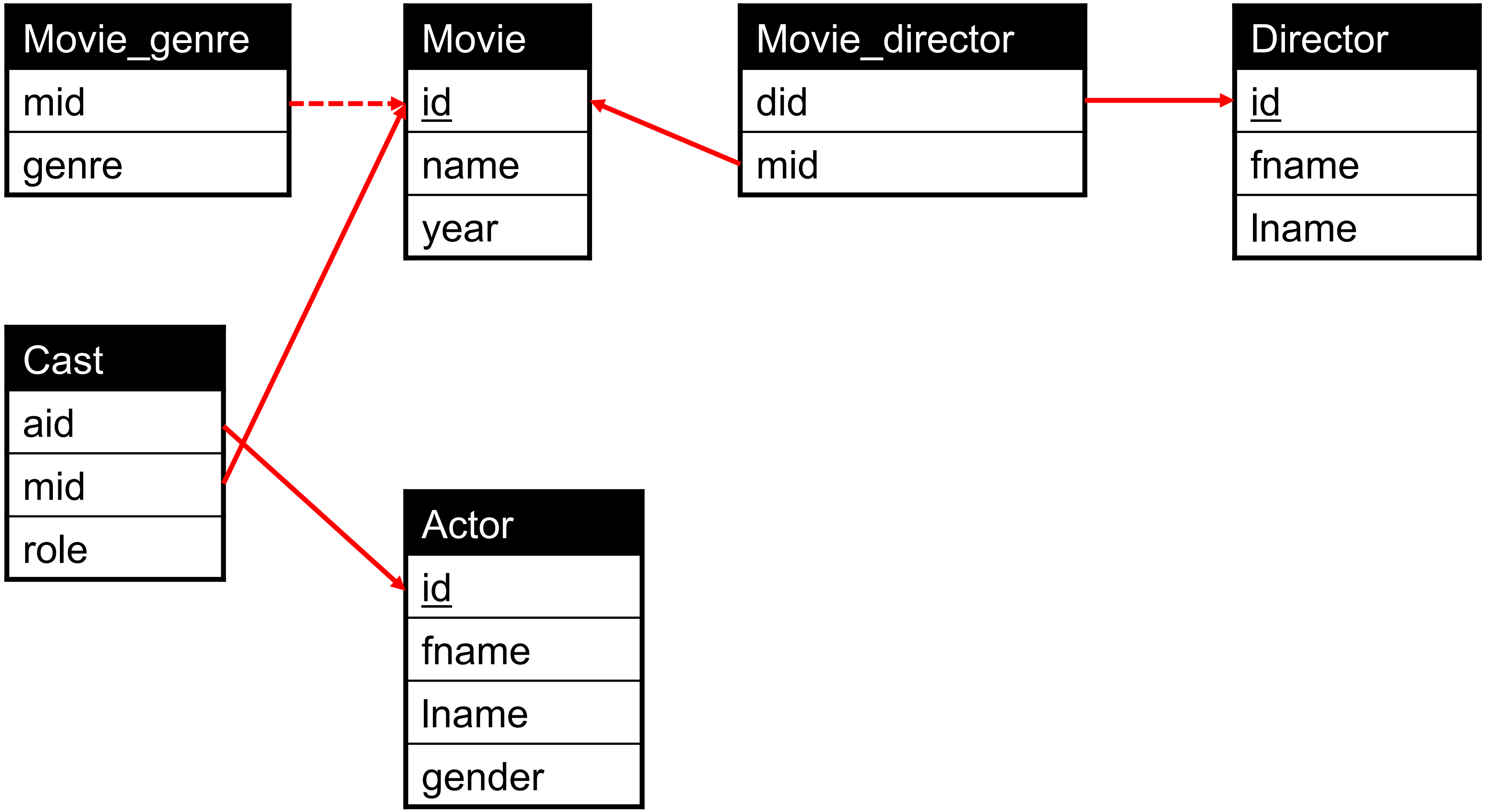
Rule: Create a new relation with the primary keys of the two entities as its primary key




*Difference to IMDB cast table?*

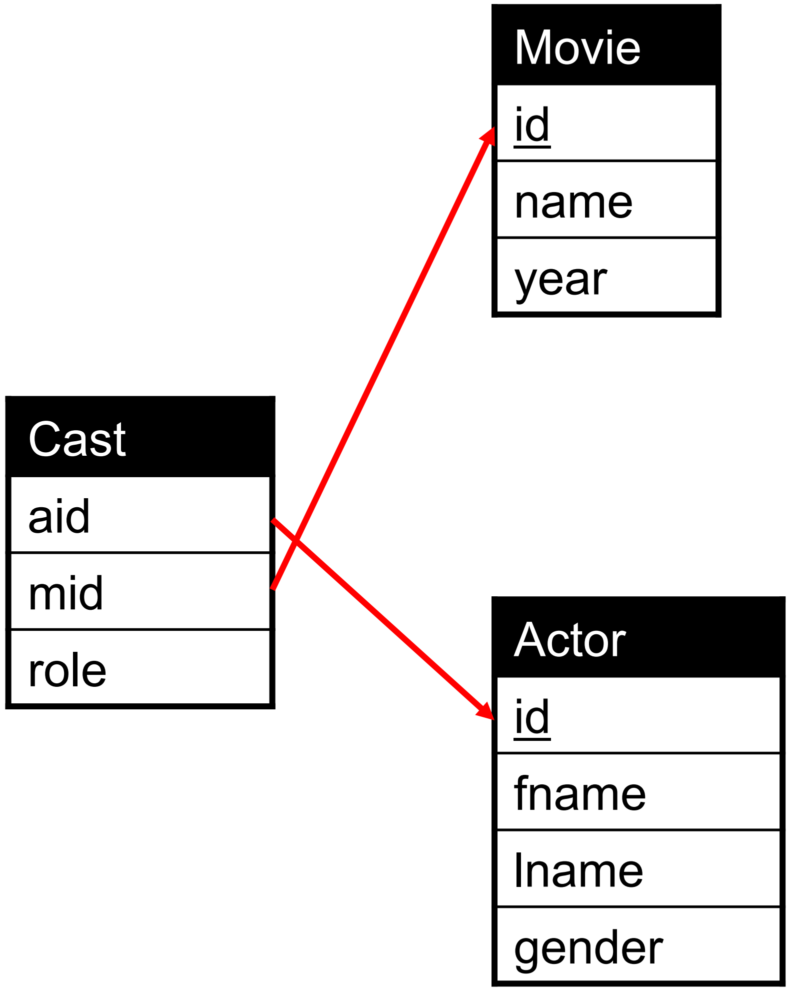
# Small IMDB Movie Database: Schema

301

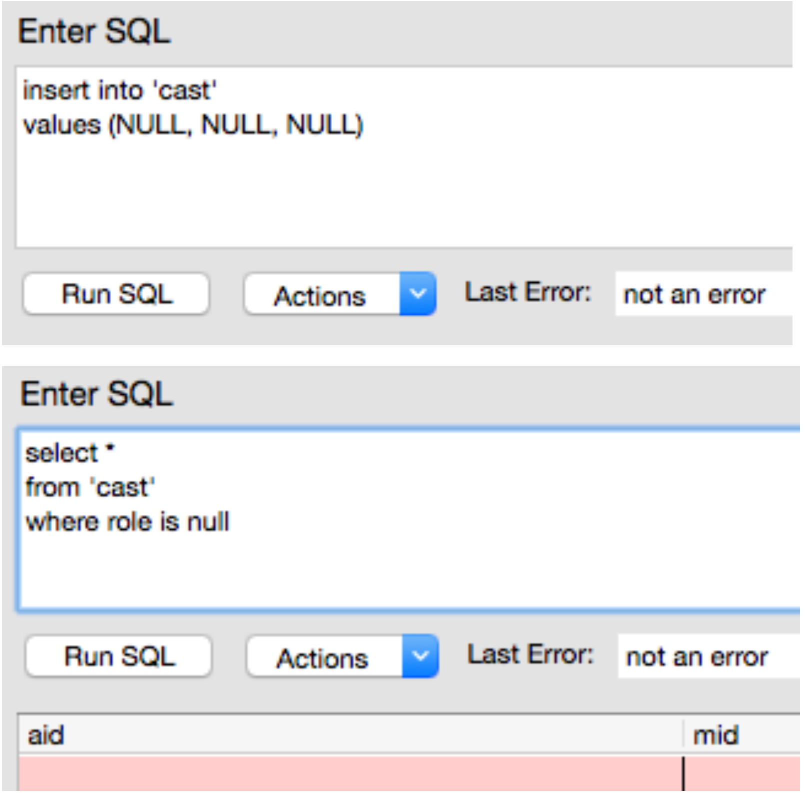


# Small IMDB Movie Database: Schema

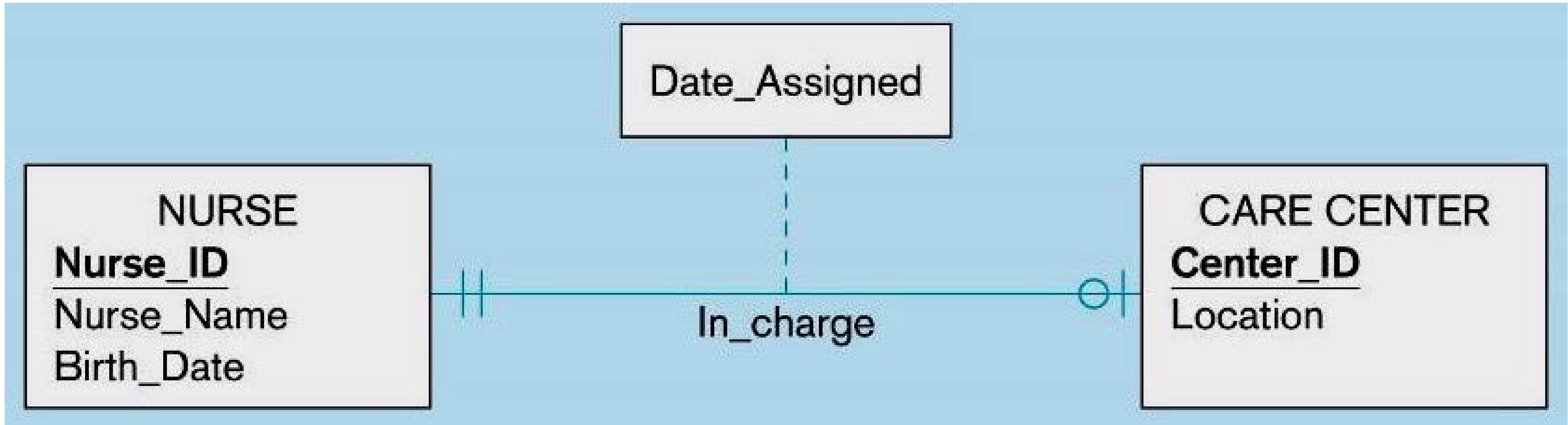
301




```
-- Table structure for table 'cast'
DROP TABLE IF EXISTS 'Cast';
CREATE TABLE 'Cast' (
  'aid' int(11) default NULL,
  'mid' int(11) default NULL,
  'role' varchar(100) default NULL,
  FOREIGN KEY(aid) REFERENCES actor(id),
  FOREIGN KEY(mid) REFERENCES movie(id)
);
```



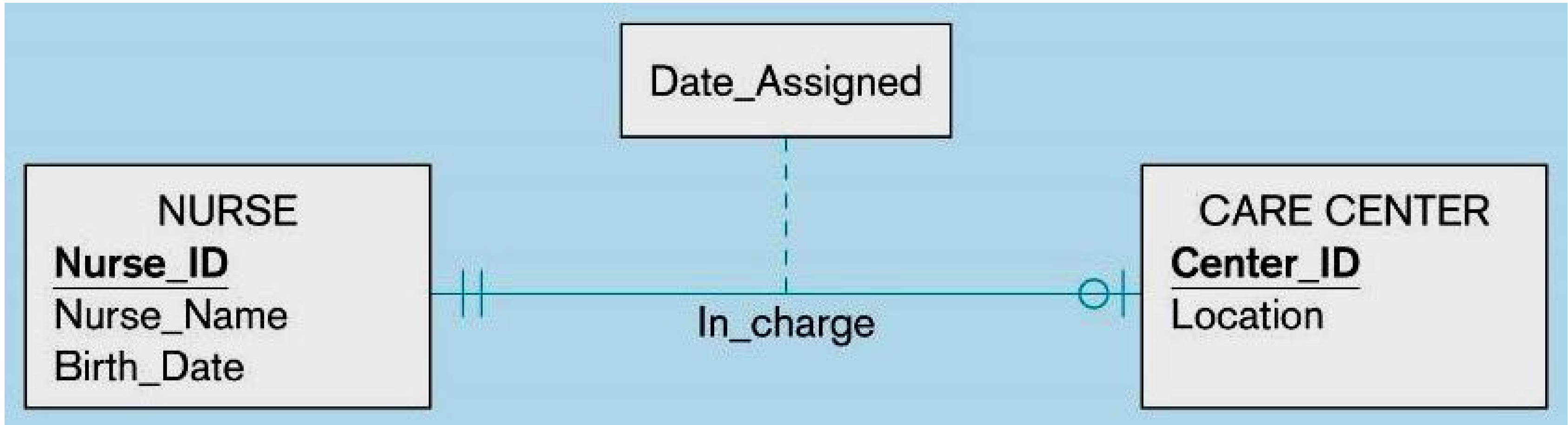
# 3) Mapping A Binary 1:1 Relationship

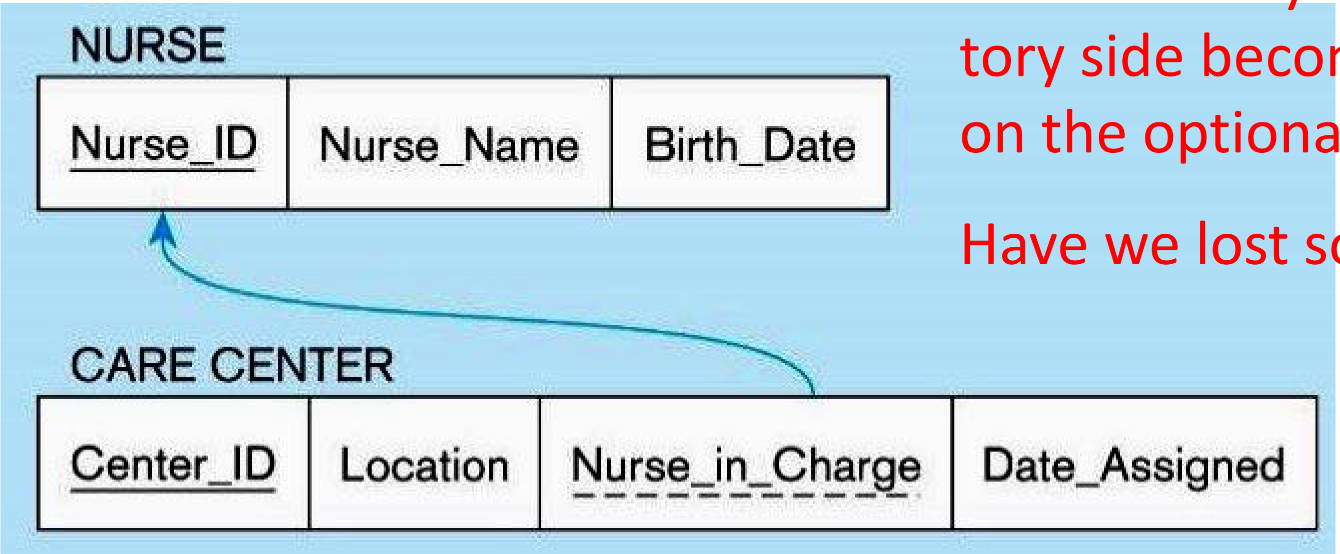



unique

| NID | NN | BD | L | CID | DA |
|---|---|---|---|---|---|
| 1 | A | ... | NULL | . | NULL |
| 2 | B | ... | Boston | . | 2121 |
| 3 | C | ... | Boston | . | |

# 3) Mapping A Binary 1:1 Relationship

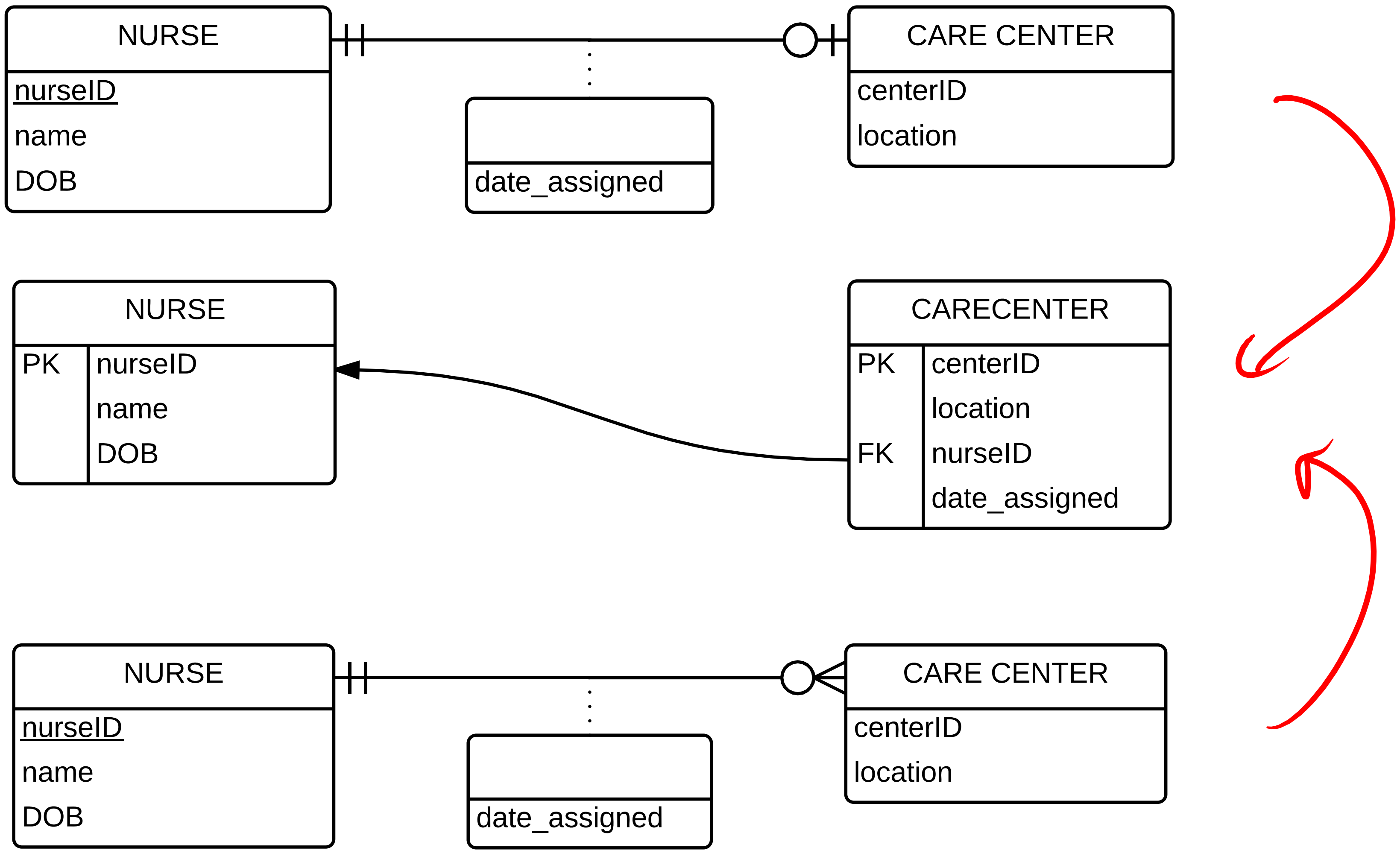


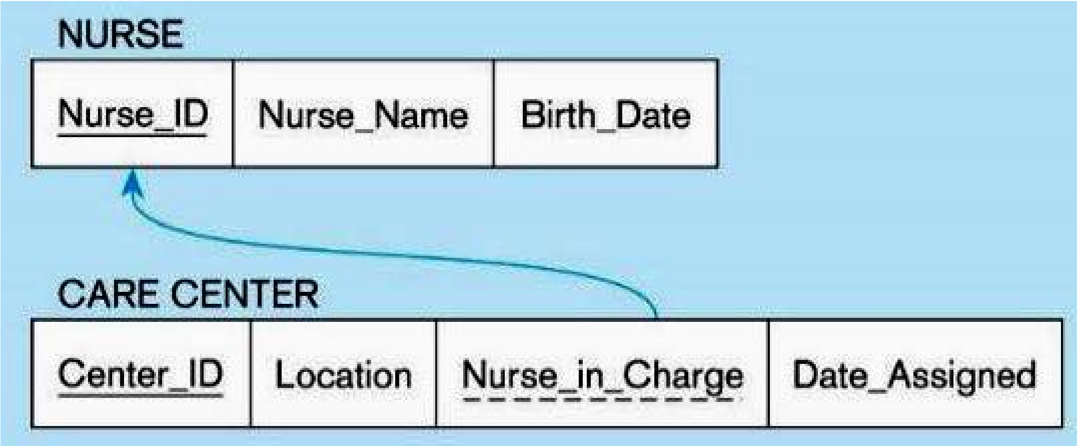

Rule: Primary key on the mandatory side becomes foreign key on the optional side

Have we lost some information?

# Transform the ERD into the appropriate schema



# Nurses: Instance




**Nurse**

| nid | name | birth_date |
|---|---|---|
| 1 | Alice | 1/1/1980 |
| 2 | Beate | 1/1/1970 |
| 3 | Clarissa | 1/1/1975 |
| 4 | Dora | 1/1/1972 |

**Carecenter**

| cid | location | nurseid@ | date_assigned |
|---|---|---|---|
| 1 | Boston | 1 | 1/1/2016 |
| 2 | New York | 3 | 3/1/2016 |

**Nurse**

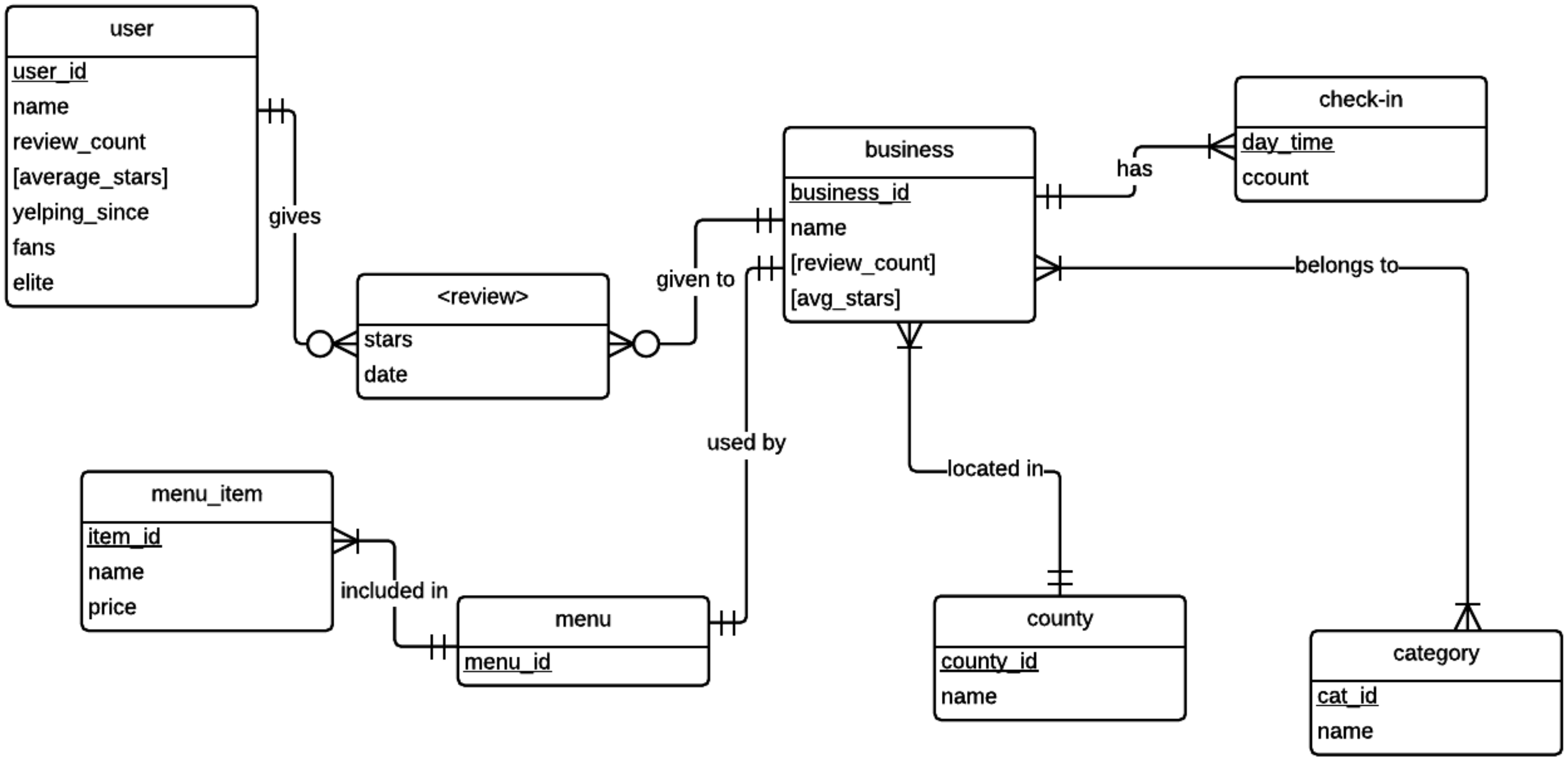
| nid | name | birth_date | carecenter@ | date_assigned |
|---|---|---|---|---|
| 1 | Alice | 1/1/1980 | 1 | 1/1/2016 |
| 2 | Beate | 1/1/1970 | NULL | NULL |
| 3 | Clarissa | 1/1/1975 | 3 | 3/1/2016 |
| 4 | Dora | 1/1/1972 | NULL | NULL |

**Carecenter**

| cid | location |
|---|---|
| 1 | Boston |
| 2 | New York |

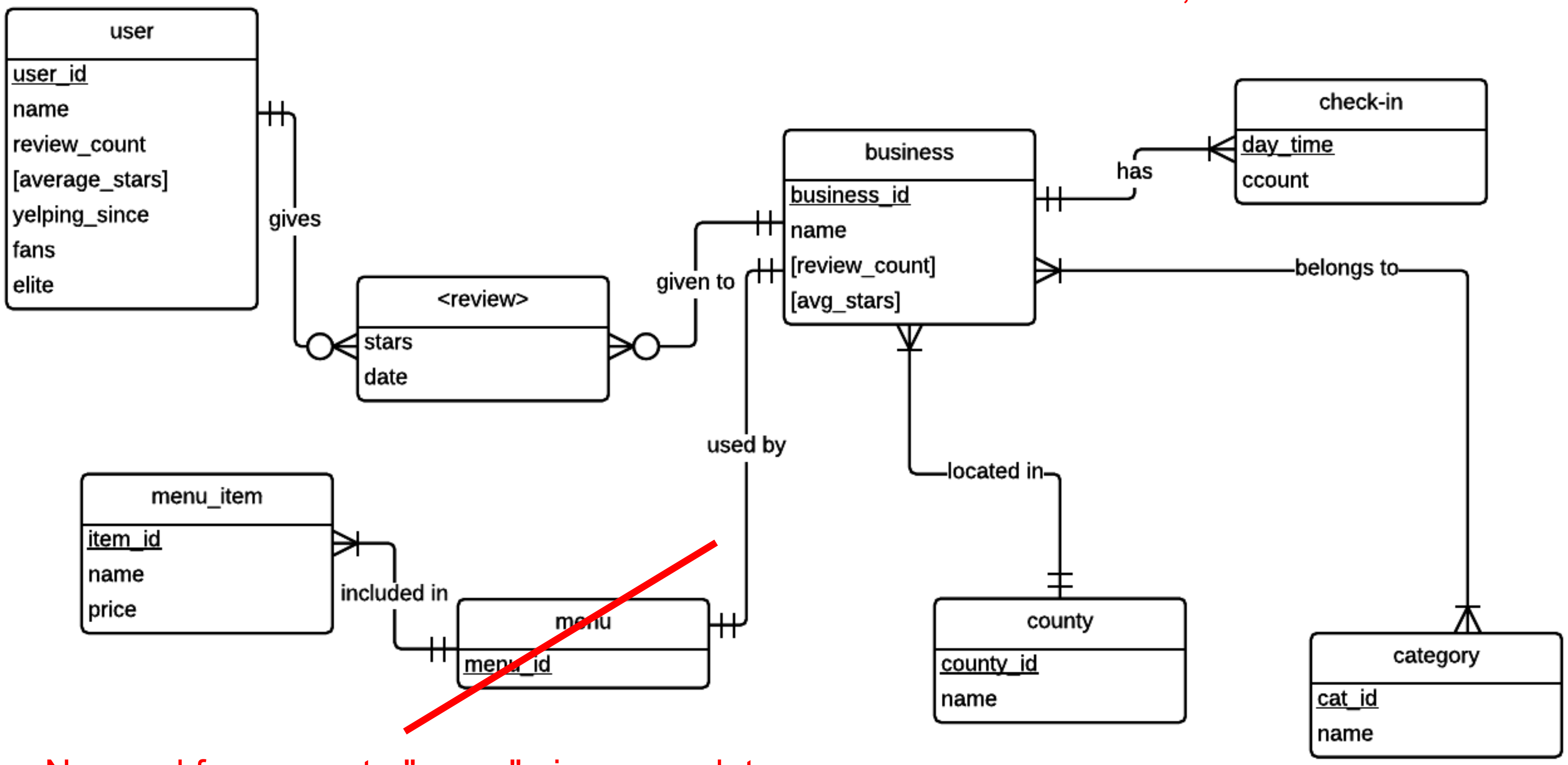
# Yelp: 1:1 relationships

?



# Yelp: 1:1 relationships

Also, who checks in?



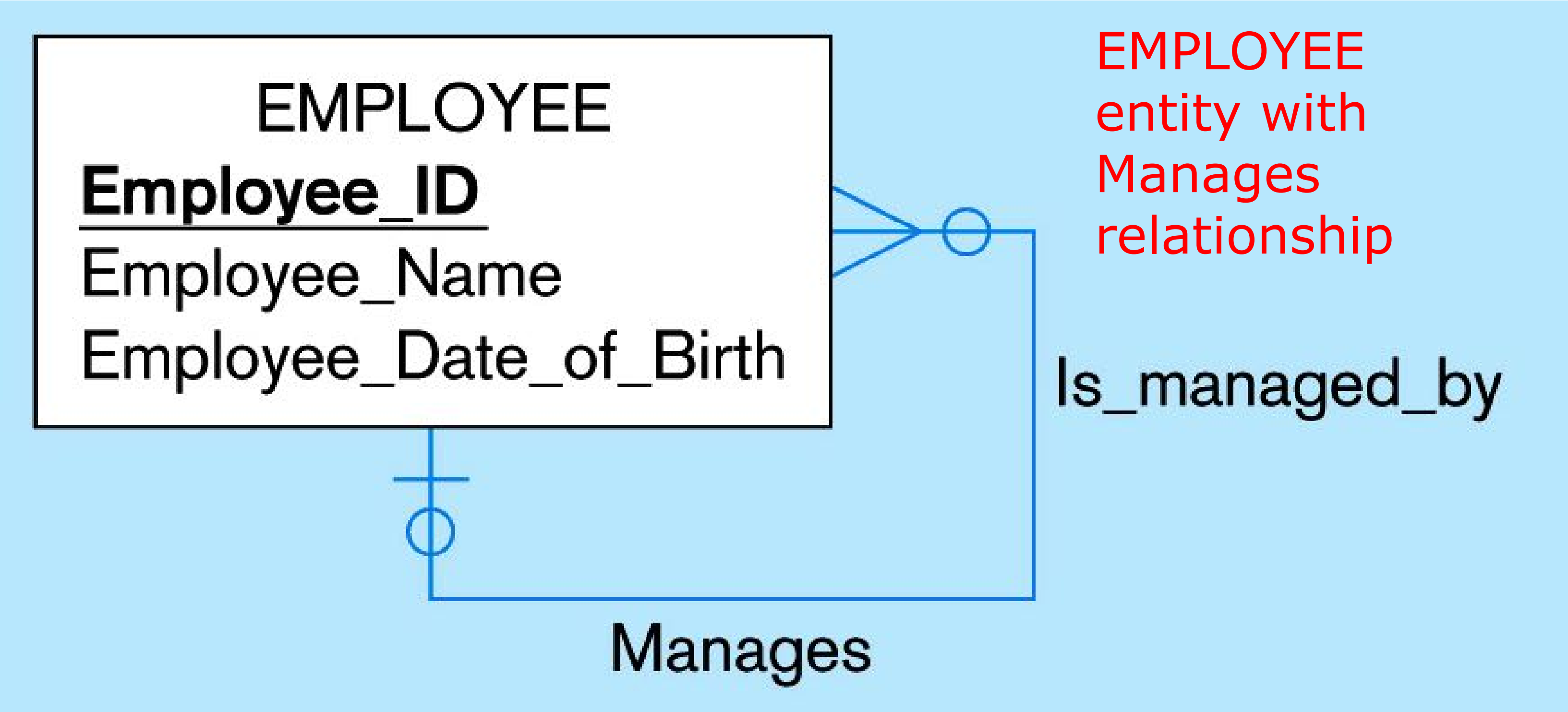

No need for separate "menu" given mandatory 1:1 relationship with business. In other words, you will never have a (1,1)-to-(1,1) relationship

# Mapping Unary Relationships

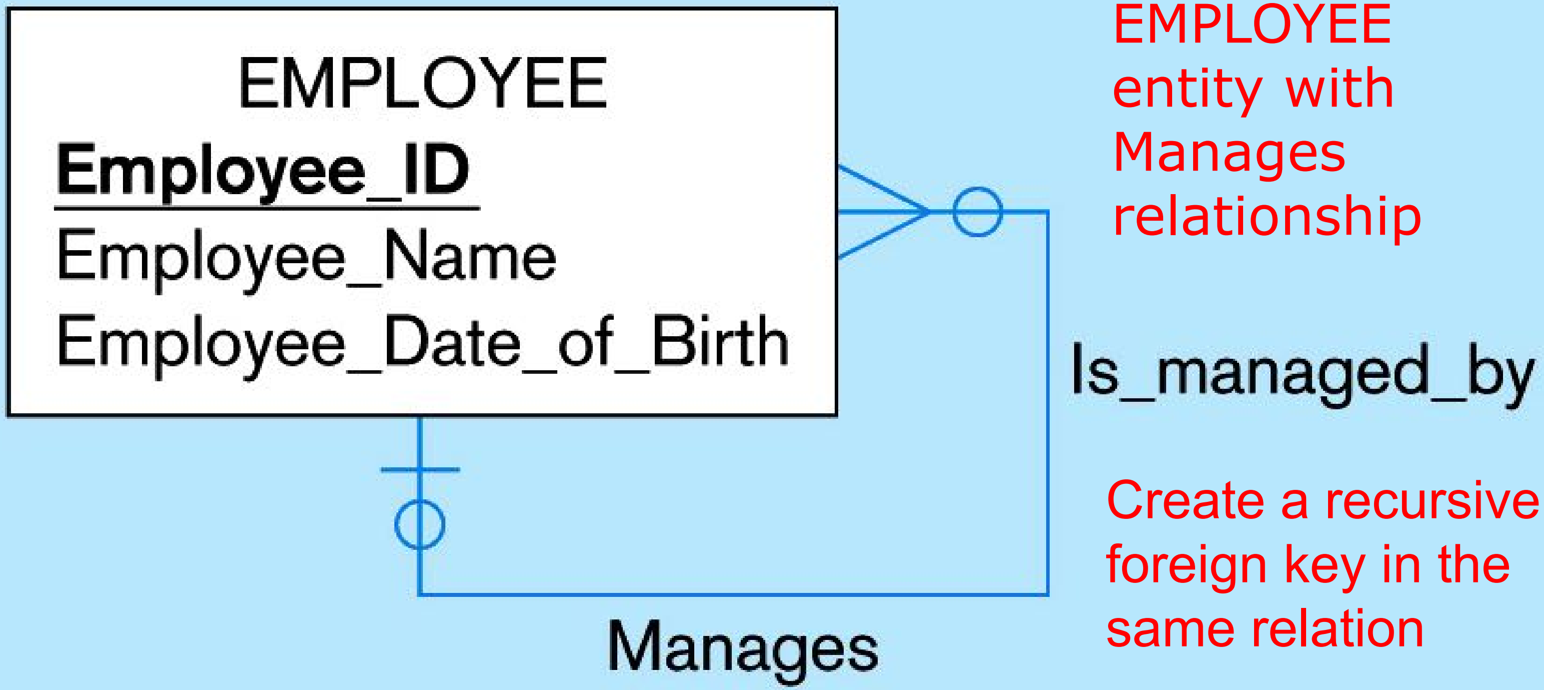
- 1) One-to-Many
  - Create a recursive foreign key in the same relation

- 2) Many-to-Many – Create two relations:
  - One for the entity type
  - One for an associative relation in which the primary key has two attributes, both taken from the primary key of the entity

# 1) Mapping a Unary 1:N Relationship



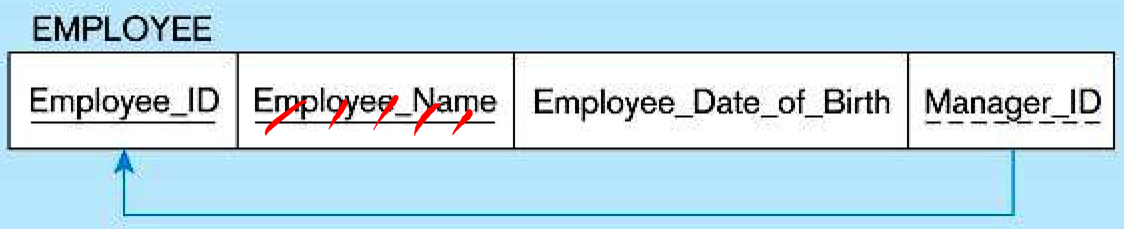
# 1) Mapping a Unary 1:N Relationship




EMPLOYEE entity with Manages relationship

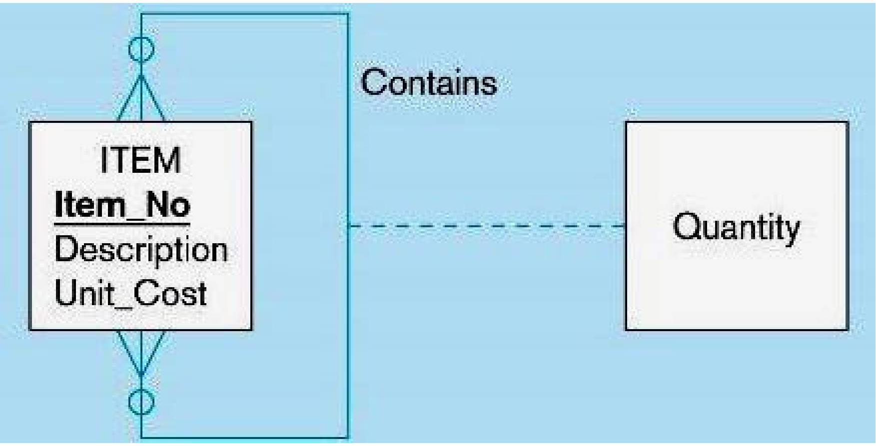
Create a recursive foreign key in the same relation

EMPLOYEE relation with recursive foreign key



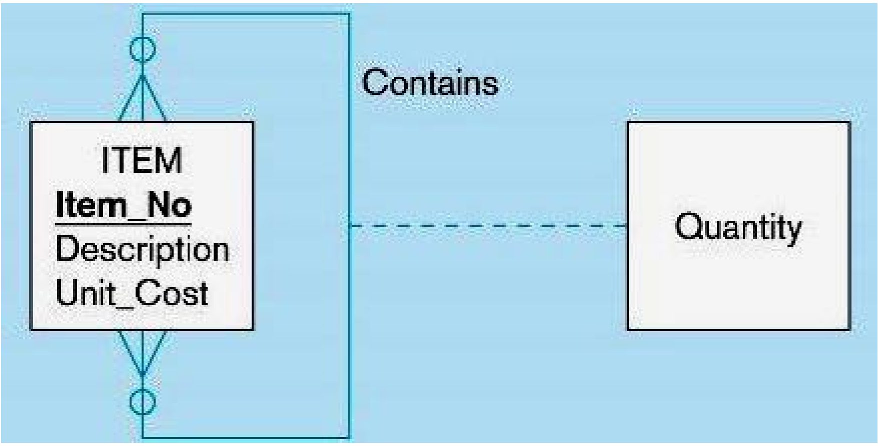
# 2) Mapping a Unary M:N Relationship

Bill-of-materials relationships (M:N)

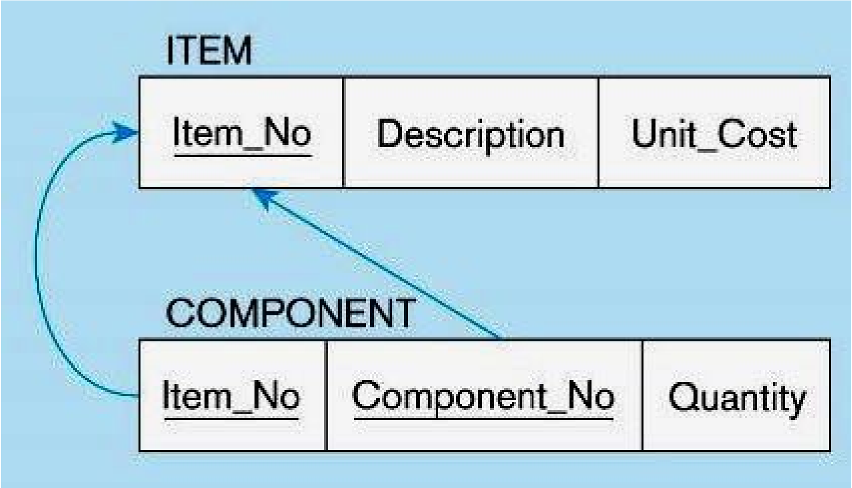


# 2) Mapping a Unary M:N Relationship

Bill-of-materials relationships (M:N)




ITEM and COMPONENT relations




Create Two relations:

- One for the entity type
- One for an associative relation in which the primary key has two attributes, both taken from the primary key of the entity

# Relational Modeling: Associative Entities

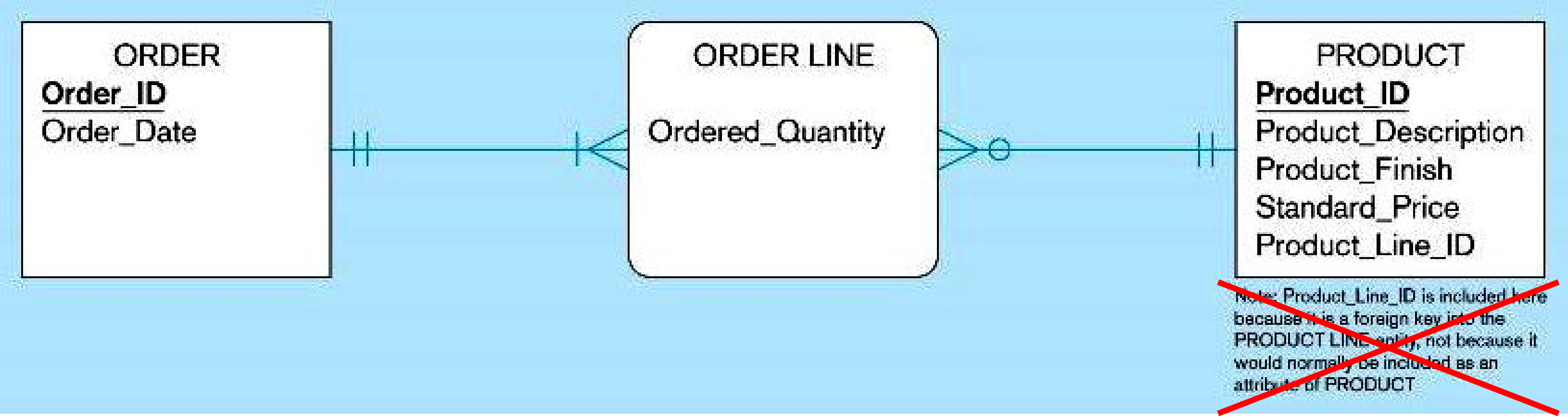
# Mapping Associative Entities

- Rules for two scenarios:

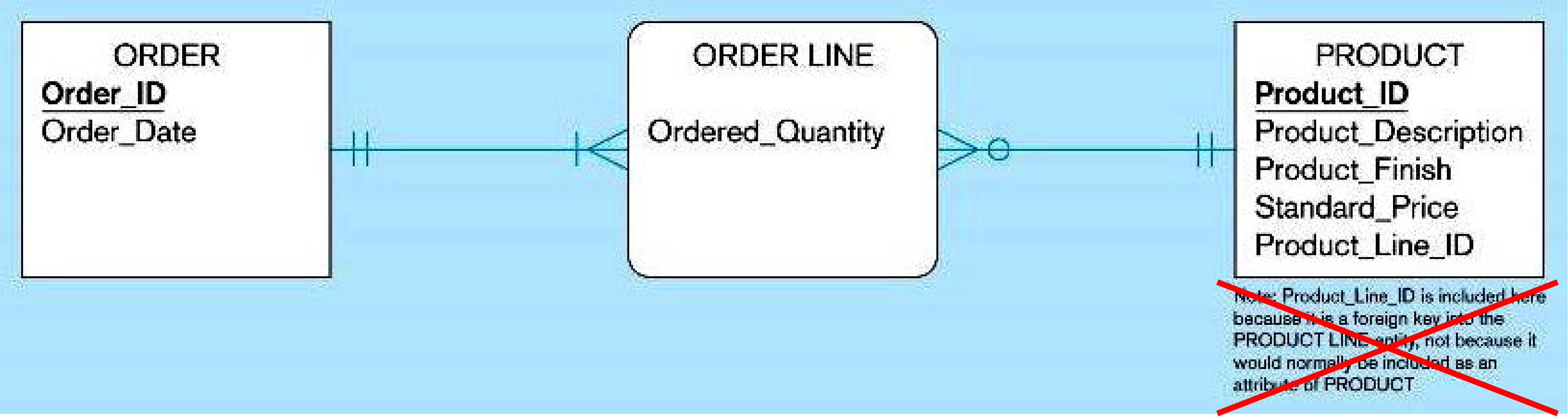
- A) Identifier Not Assigned
  - Default primary key for the association relation is composed of the primary keys of the two entities (as in M:N relationship)

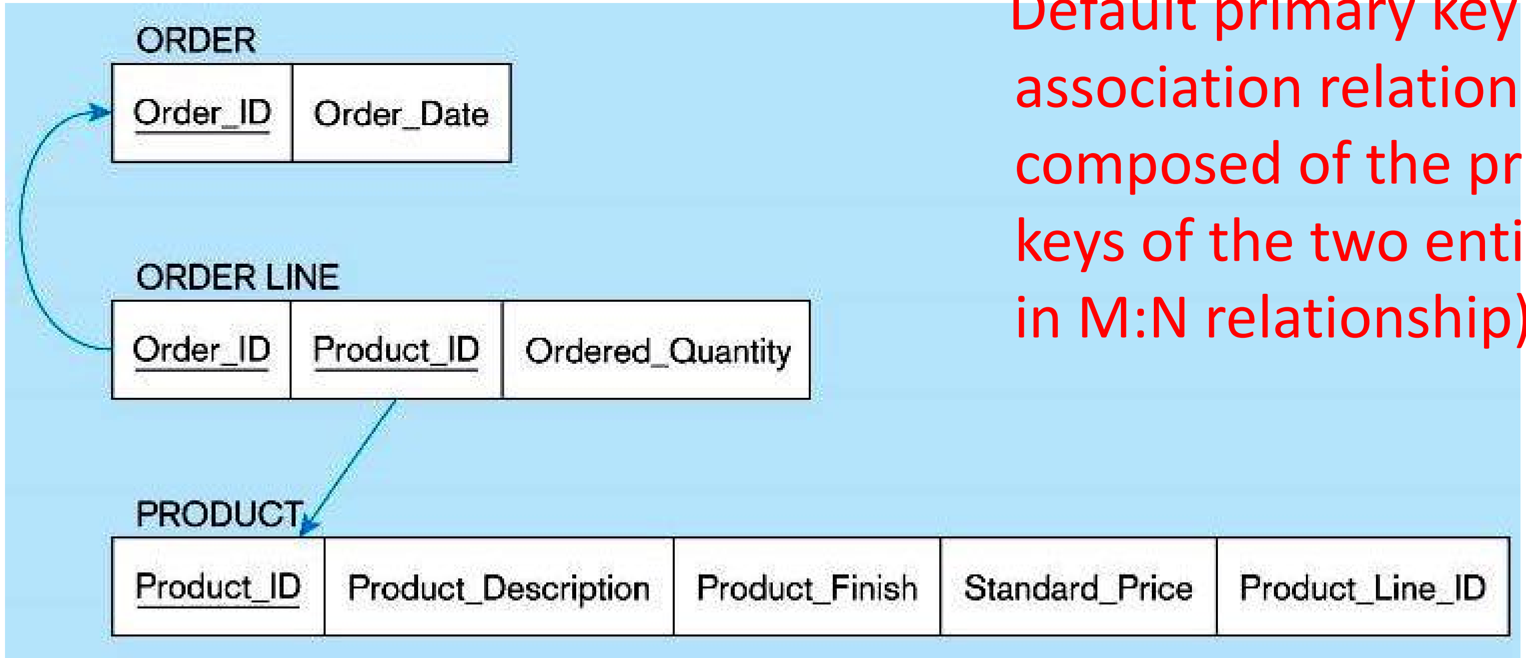
- B) Identifier Assigned
  - It is natural and familiar to end-users
  - Default identifier may not be unique

# A) Associative Entity Relations (No Identifier)



# A) Associative Entity Relations (No Identifier)

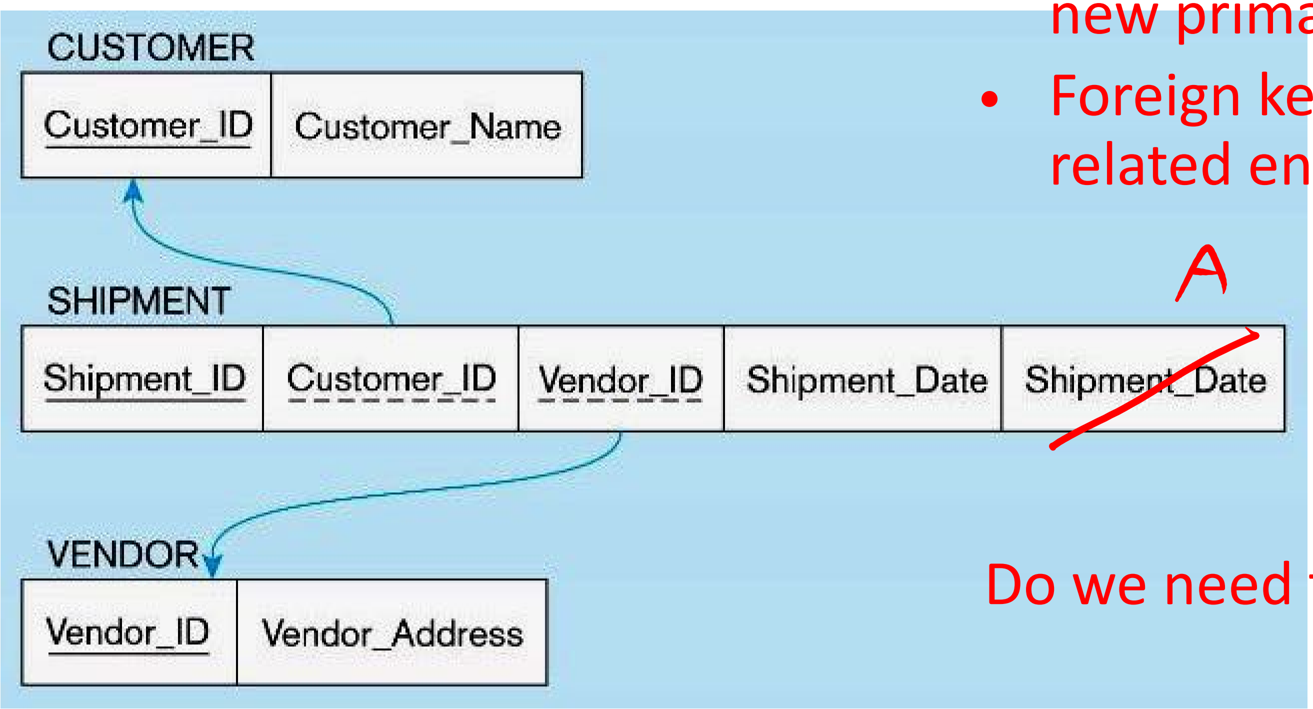



Default primary key for the association relation is composed of the primary keys of the two entities (as in M:N relationship)

# B) Associative Entity Relations (With Identifier)

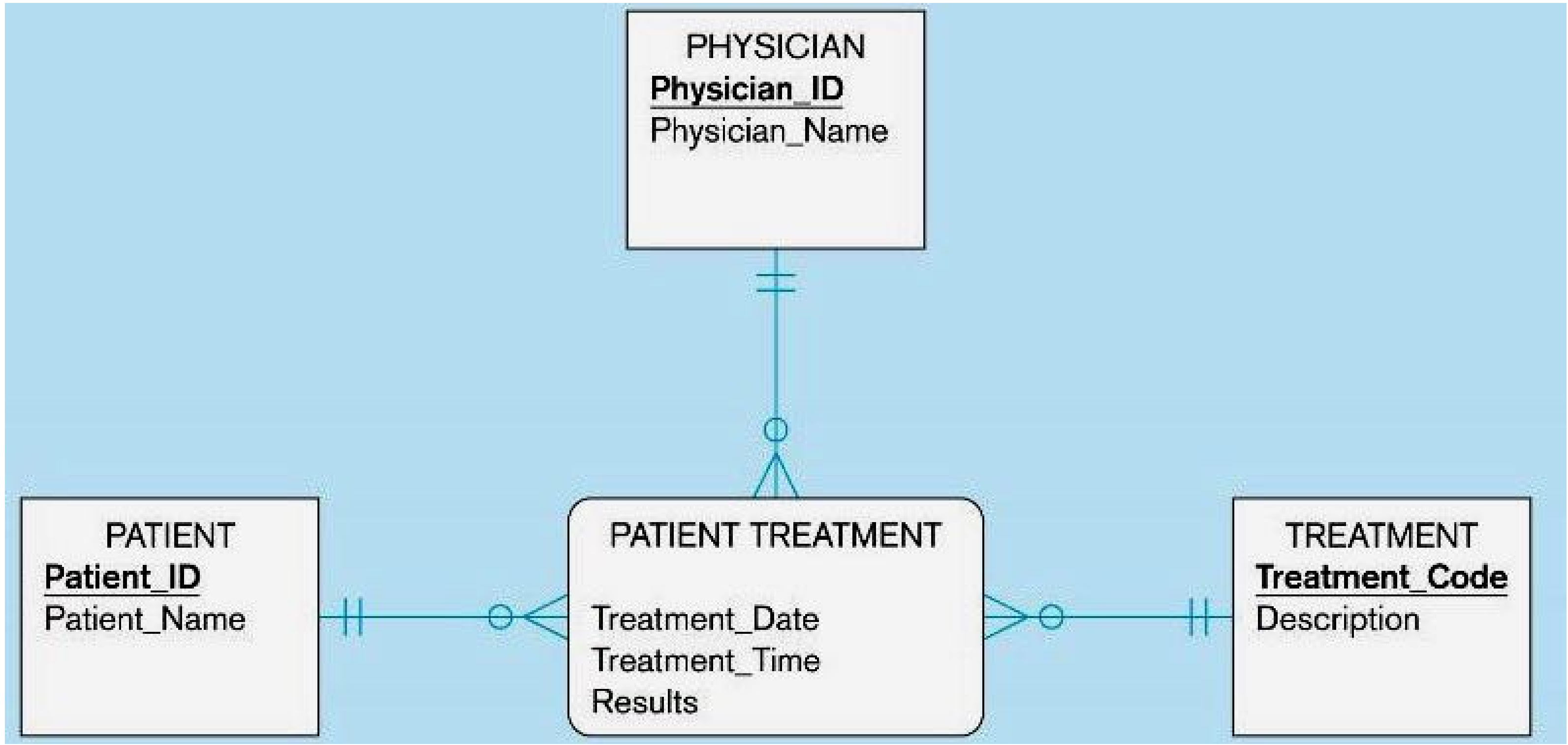
# B) Associative Entity Relations (With Identifier)


- Identifier attribute becomes new primary key in relation
- Foreign keys reference all related entities

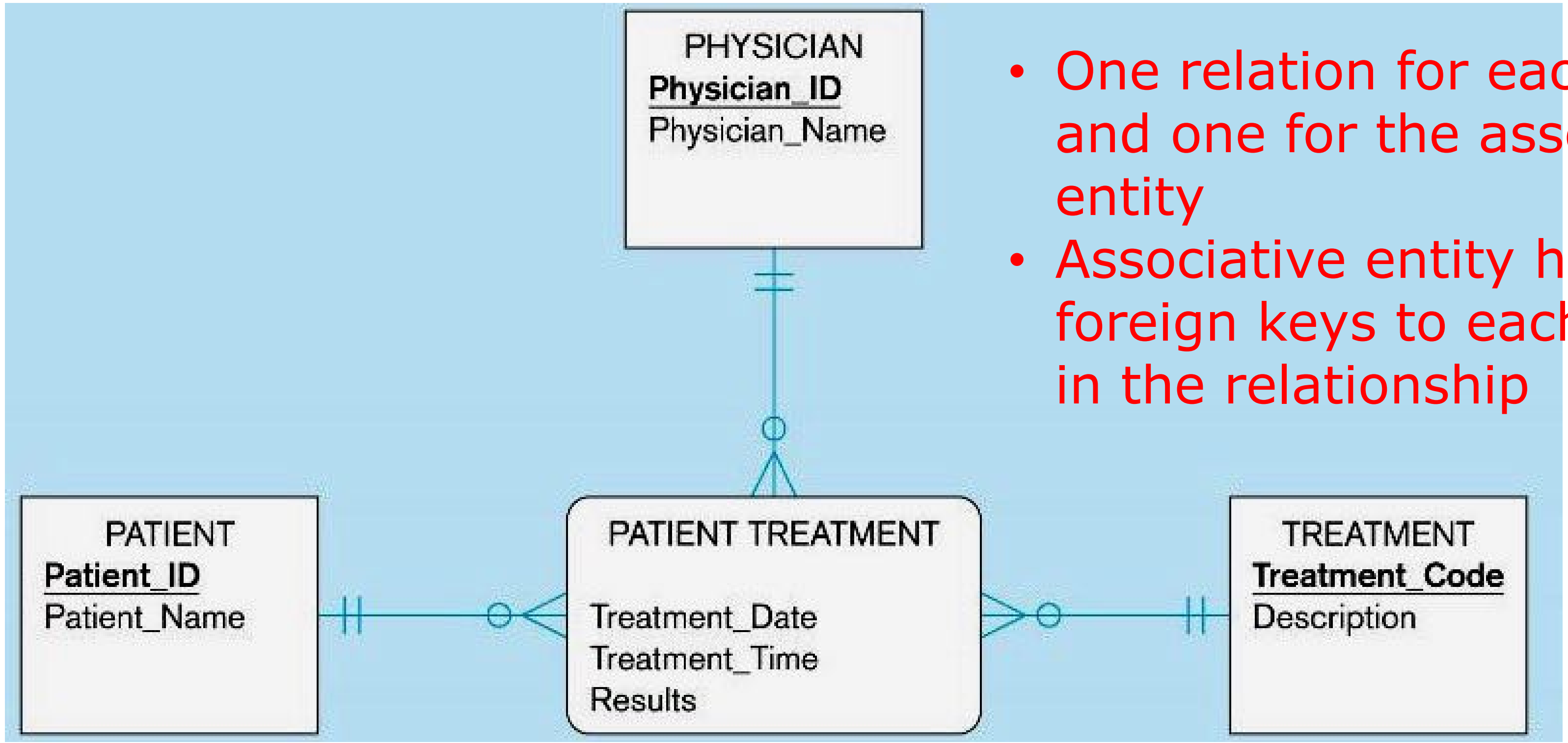
Do we need the key?

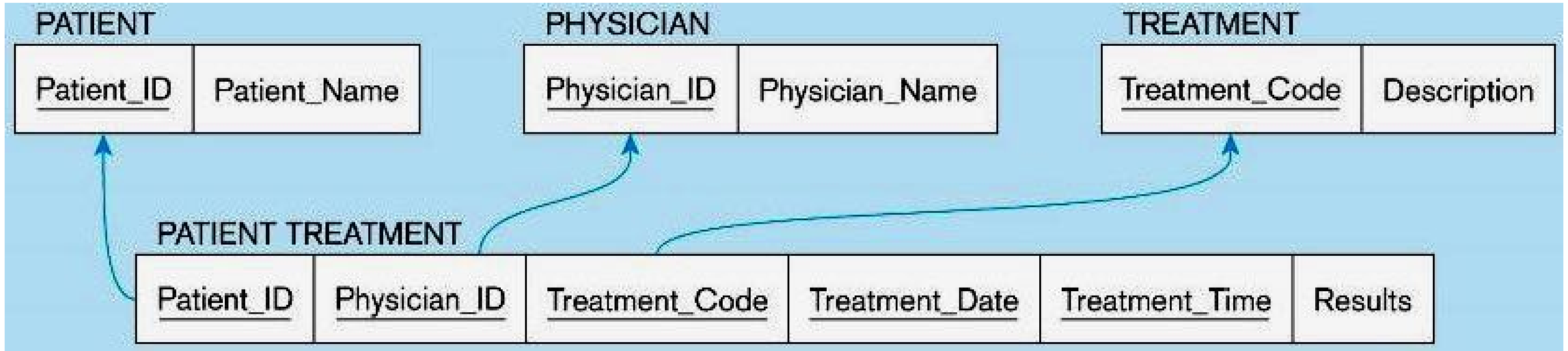
# Mapping ternary relationship w/ associative entity



# Mapping ternary relationship w/ associative entity

- One relation for each entity and one for the associative entity
- Associative entity has foreign keys to each entity in the relationship



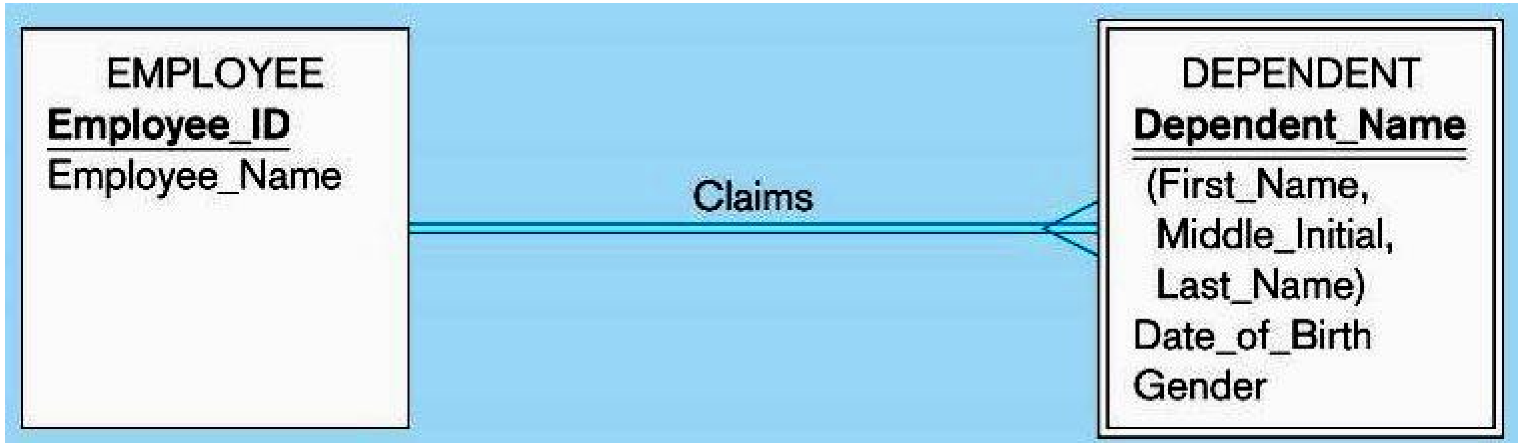
# Relational Modeling: Weak entities

# Mapping Weak Entities

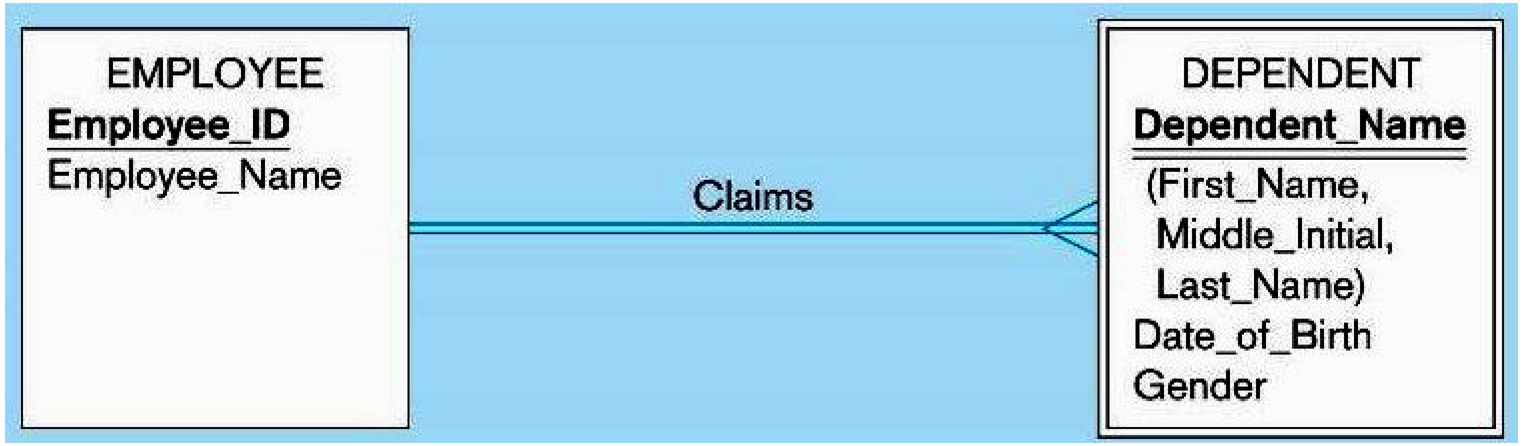
- Weak Entities become separate relations with a foreign key taken from the superior entity

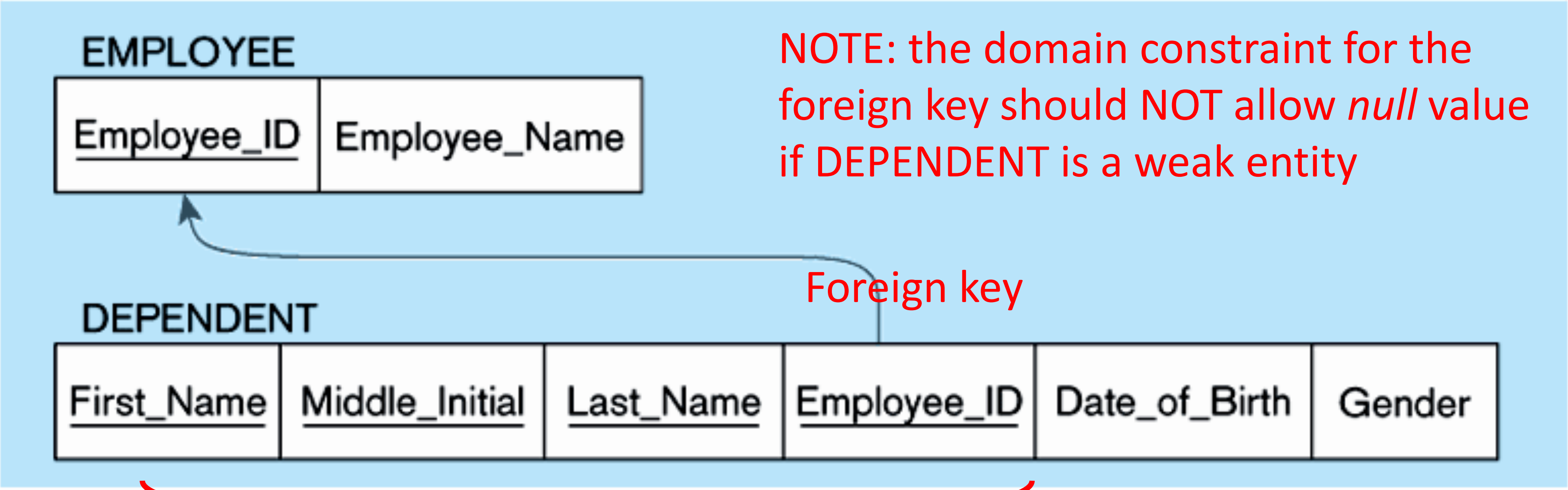
- Primary key composed of:
  - Partial identifier of weak entity
  - Primary key of identifying relation (strong entity)

# Example: Mapping A Weak Entity (Relations)



# Example: Mapping A Weak Entity (Relations)




NOTE: the domain constraint for the foreign key should NOT allow *null* value if DEPENDENT is a weak entity

Composite primary key

or "Surrogate primary key"
*(George Foreman ...)*

DEPENDENT(<u>Dependent#</u>, EmployeeID, FirstName, MiddleInitial, LastName, DateOfBirth, Gender)

# Overview
# Database normalization & Design Theory

# Normalization

- Understand the normalization process and why a normalized data model is desirable (no redundancy)
- Be able to explain normal forms and identify when a relational model is in any of them
- Be able to explain anomalies and how to avoid them
  - Insertion, deletion, and modification
- Actually apply normalization ☺

# Normalization

- Organizing data to minimize redundancy (repeated data)

- This is good for two reasons
  - The database takes up less space
  - You have a lower chance of inconsistencies in your data

- If you want to make a change to a record, you only have to make it in one place
  - The relationships take care of the rest

- But you will usually need to link the separate tables together in order to retrieve information

# First Normal Form (1NF)

- A database schema is in *First Normal Form* if all tables are flat (no "nested relations")

**Student**

| Name | GPA | Course |
|---|---|---|
| Alice | 3.8 | Math<br>DB<br>OS |
| Bob | 3.7 | DB<br>OS |
| Carol | 3.9 | Math<br>OS |

?

# First Normal Form (1NF)

- A database schema is in *First Normal Form* if all tables are flat (no "nested relations")

**Student**

| Name | GPA | Course |
|---|---|---|
| Alice | 3.8 | Math<br>DB<br>OS |
| Bob | 3.7 | DB<br>OS |
| Carol | 3.9 | Math<br>OS |

→

**Student**

| Name | GPA | Course |
|---|---|---|
| Alice | 3.8 | Math |
| Alice | 3.8 | DB |
| Alice | 3.8 | OS |
| Bob | 3.7 | DB |
| Bob | 3.7 | OS |
| Carol | 3.9 | Math |
| Carol | 3.9 | OS |

# First Normal Form (1NF)

- A database schema is in *First Normal Form* if all tables are flat (no "nested relations")

**Student**

| Name | GPA | Course |
|---|---|---|
| Alice | 3.8 | Math<br>DB<br>OS |
| Bob | 3.7 | DB<br>OS |
| Carol | 3.9 | Math<br>OS |

May need to add keys

**Student**

| <u>Name</u> | GPA |
|---|---|
| Alice | 3.8 |
| Bob | 3.7 |
| Carol | 3.9 |

**Takes**

| Student | Course |
|---|---|
| Alice | Math |
| Carol | Math |
| Alice | DB |
| Bob | DB |
| Alice | OS |
| Carol | OS |

**Course**

| <u>Course</u> |
|---|
| Math |
| DB |
| OS |

# Data Anomalies

- When a database is poorly designed we get anomalies (those are bad) resulting from redundancies:
  - <u>Update anomalies</u>: need to change in several places
  - <u>Insert anomalies</u>: need to repeat data for new inserts
  - <u>Deletion anomalies</u>: may lose data when we don't want

# Relational Schema Design

Recall set attributes (persons with several phones):

**Employee**

| Name | SSN | PhoneNumber | City |
|---|---|---|---|
| Fred | 123-45-6789 | 412-555-1234 | Boston |
| Fred | 123-45-6789 | 412-555-6543 | Boston |
| Joe | 987-65-4321 | 908-555-2121 | Westfield |

- One person may have multiple phones, but lives in only one city
- Primary key is thus (SSN, PhoneNumber)

Do you see any anomalies?

# Relational Schema Design

Recall set attributes (persons with several phones):

**Employee**

| Name | SSN | PhoneNumber | City |
|---|---|---|---|
| Fred | 123-45-6789 | 412-555-1234 | Boston |
| Fred | 123-45-6789 | 412-555-6543 | Boston |
| Joe | 987-65-4321 | 908-555-2121 | Westfield |

- One person may have multiple phones, but lives in only one city
- Primary key is thus (SSN, PhoneNumber)

Do you see any anomalies?

- **Update anomalies**: what if Fred moves to "New York"?
- **Insert anomalies**: what if Joe gets a second phone number
- **Deletion anomalies**: what if Joe deletes his phone number?

(what if Joe had no phone #)

What do we do????

# Relation Decomposition

Break the relation into two:

**Employee**

| Name | SSN | PhoneNumber | City |
| --- | --- | --- | --- |
| Fred | 123-45-6789 | 412-555-1234 | Boston |
| Fred | 123-45-6789 | 412-555-6543 | Boston |
| Joe | 987-65-4321 | 908-555-2121 | Westfield |

**Employee**

| Name | SSN | City |
| --- | --- | --- |
| Fred | 123-45-6789 | Boston |
| Joe | 987-65-4321 | Westfield |

**Phone**

| SSN | PhoneNumber |
| --- | --- |
| 123-45-6789 | 412-555-1234 |
| 123-45-6789 | 412-555-6543 |
| 987-65-4321 | 908-555-2121 |

Anomalies have gone:

- No more repeated data
- Easy to move Fred to "New York" (how ?)
- Easy to delete all Joe's phone numbers (how ?)

# Good News / Bad News

- The good news: when you start with solid ER modeling and follow the steps described to create relations then your relations will usually be pretty well normalized

- The bad news: you often don't have the benefit of starting from a good ER model.

- The good news (part 2): the steps we will cover in class will help you convert poorly normalized tables into highly normalized tables