CS4221
Relational Databases I. Concepts

Yao LU

2024 Semester 2

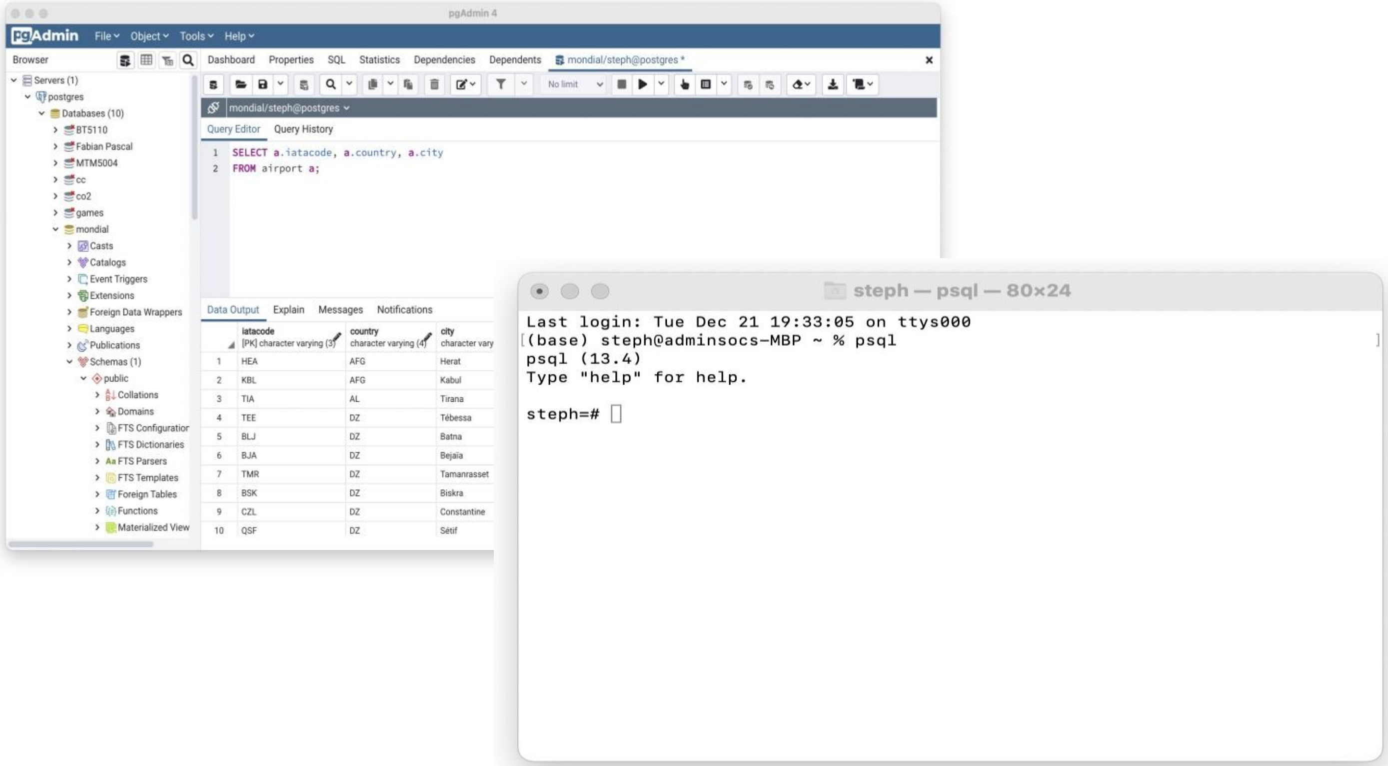
National University of Singapore
School of Computing

Today's agenda

- Refresher
  - SQL
  - Entity-Relationship model
- Design theory
  - Normal forms & functional dependencies
  - Boyce-Codd normal form
  - 3NF

PostgreSQL

- We use the relational database management system PostgreSQL with pgAdmin 4 or psql.



TPC-C

- A wholesale supplier company operates out of a number of warehouses. The warehouses maintain stocks for the items sold by the company. For each item available we record the quantity in stock in each warehouse.

TPC-C

- Warehouses have a unique identifier, a name and a location defined by street, city and country.

```
CREATE TABLE warehouses (
 w_id INTEGER PRIMARY KEY,
 w_name VARCHAR(10) NOT NULL,
 w_street VARCHAR(20) NOT NULL,
 w_city VARCHAR(20) NOT NULL,
 w_country CHAR(9) NOT NULL);

 INSERT INTO warehouses
 (w_id, w_name, w_street, w_city, w_country)
 VALUES
 (301, 'Schmedeman', 'Sunbrook', 'Singapore', 'Singapore');
```

TPC-C

- Items have a unique identifier, a unique image identifier, a name and a price.

```
CREATE TABLE items (
 i_id INTEGER PRIMARY KEY,
 i_im_id VARCHAR(50) UNIQUE NOT NULL,
i_name VARCHAR(50)  NOT NULL,
 i_price NUMERIC(5, 2)  NOT NULL CHECK(i_price > 0));

INSERT INTO items
(i_id , i_im_id , i_name, i_price)
VALUES
(1, '35356226', 'Indapamide', 95.23);
```

TPC-C

- For each item available we record the quantity in stock in each warehouse. If an item is not available in a warehouse, then there is no entry for this pair. The quantity is always equal to or greater than 1.

```sql
CREATE TABLE stocks (
w_id INTEGER REFERENCES warehouses(w_id),
i_id INTEGER REFERENCES items(i_id),
s_qty SMALLINT NOT NULL,
PRIMARY KEY (w_id, i_id));

INSERT INTO stocks VALUES (301, 1, 338);
INSERT INTO stocks VALUES (301, 1, 12);
INSERT INTO stocks VALUES (301, 4, 938);
```

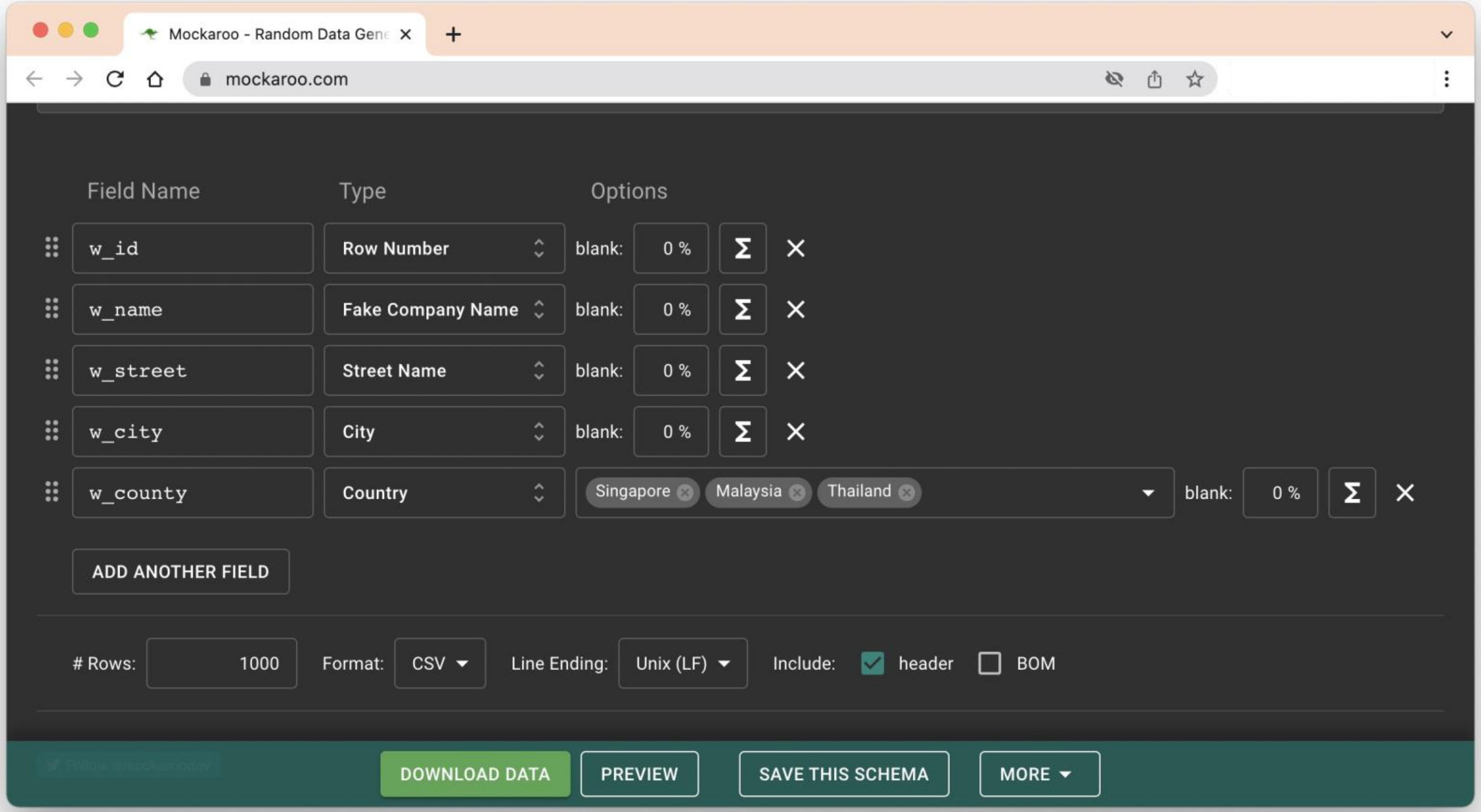
Integrity constraints

- Primary key
- Unique
- Not NULL
- Foreign Key
- Check

Table CHECK constraints, CHECK constraints with general SQL statement and ASSERTION constraints are not available in database management systems.

Creating tables

- We can generate the SQL scripts to create and populate the tables with Mockaroo (www.mockaroo.com).



Creating tables

- We can use the files:

  TPCCSchema.sql,

  TPCCItems.sql,

  TPCCStocks.sql,

  TPCCWarehouses.sql,

  TPCCClean.sql,

  TPCCQueries.sql.

- Does order matter?

Querying tables

- A point query returns at most one record based on an equality condition.

```
SELECT w.w_name
FROM warehouses w
WHERE w.w_id = 123;
```

Querying tables

- A multipoint query returns at several record based on an equality condition.

```
SELECT w.w_name
FROM warehouses w
WHERE w.w_city = 'Singapore';
```

Querying tables

- A range query returns several records based on inequality conditions on some attribute(s).

```sql
SELECT s.i_id
FROM stocks s
WHERE s.s_qty BETWEEN 0 AND 10;
```

```sql
SELECT s.i_id
FROM stocks s
WHERE s.s_qty <= 10;
```

Querying tables

- A prefix match query returns several records based on prefix condition on some attributes.

```
SELECT w.w_city, w.w_name
FROM warehouses w
WHERE w.w_city LIKE 'Si%';
```

Querying tables

- A extremal query returns several records based on an equlity condition comparing with an maxium or a minimum.

```
SELECT s1.i_id
FROM stocks s1
WHERE s1.s_qty = ALL (
 SELECT MAX(s2.s_qty)
 FROM stocks s2);
```

Querying tables

- An ordering query returns several records in a prescribed order.

```
SELECT w.w_id, w.w_name, w.w_city
FROM warehouses w
ORDER BY w.w_name;
```

```
SELECT w.w_id, w.w_name, w.w_city
FROM warehouses w
ORDER BY w.w_city, w.w_name;
```

```
SELECT w.w_id, w.w_name
FROM warehouses w
ORDER BY w.w_city;
```

Querying tables

- A grouping query partitions the records into groups. It is used to express aggregate queries that involve aggregate functions (MAX, MIN AVG, SUM etc.).

```
SELECT s.i_id
FROM stocks s
GROUP BY s.i_id;
```

```
SELECT s.i_id, FLOOR(AVG(s.s_qty)) AS average_qty
FROM stocks s
GROUP BY s.i_id;
```

```
SELECT s.i_id, FLOOR(AVG(s.s_qty)) AS average_qty
FROM stocks s
GROUP BY s.i_id, s.w_id;
```

```
SELECT s.w_id
FROM stocks s
GROUP BY s. w_id
HAVING AVG(s.s_qty) >= 550;
```

Querying tables

- A join query combines several tables.

```sql
SELECT s.i_id
FROM stocks s, warehouses w
WHERE s.w_id = w.w_id
 AND w.w_city = 'Singapore';
```

```sql
SELECT s.i_id
FROM stocks s CROSS JOIN warehouses w
WHERE s.w_id = w.w_id
 AND w.w_city = 'Singapore';
```

```sql
SELECT s.i_id
FROM stocks s JOIN warehouses w
ON s.w_id = w.w_id
WHERE w.w_city = 'Singapore';
```

Querying tables

- A natural join query combines several tables on equality of attributes with the same name.

```sql
SELECT s.i_id
FROM stocks s NATURAL JOIN warehouses w
WHERE w.w_city = 'Singapore';
```

```sql
SELECT s.i_id
FROM stocks s INNER JOIN warehouses w ON s.w_id = w.w_id
WHERE w.w_city = 'Singapore';
```

Querying tables

- A outer join query combines several tables and pads the indicated (LEFT/RIGHT/FULL) missing values with nulls.

```
SELECT i.i_id, s.w_id
FROM items i LEFT OUTER JOIN stocks s
ON i.i_id=s.i_id;
```

```
SELECT i.i_id, s.w_id
FROM stocks s RIGHT OUTER JOIN items i
ON i.i_id=s.i_id;
```

```
SELECT i.i_id, s.w_id
FROM stocks s FULL OUTER JOIN items i
ON i.i_id=s.i_id;
```

Querying tables

- A nested join query uses the result of a subquery (inner query) in the WHERE or HAVING clause of an outer query.

```sql
SELECT s.i_id
FROM stocks s
WHERE s.w_id IN (
 SELECT w2.w_id
 FROM warehouses w2
 WHERE w2.w_city = 'Singapore');
```

Querying tables

- A correlated nested query uses the result of a subquery (inner query) in the WHERE or HAVING clause of an outer query. The inner query referes to attributes of the relations in the FROM clause of the outer query.

```
SELECT s.i_id
FROM stocks s
WHERE EXISTS (
  SELECT *
  FROM warehouses w
  WHERE s.w_id = w.w_id
  AND w.w_city = 'Singapore');
```

Querying tables

- There is no theoretical limit on the number of level of nesting. There might be some in individual systems.

```sql
SELECT i.i_id
FROM items i
WHERE NOT EXISTS (
    SELECT *
    FROM warehouses w
    WHERE NOT EXISTS (
        SELECT *
        FROM stocks s
        WHERE s.w_id=w.w_id AND s.i_id=i.i_id));
```

Querying tables

- There can be nested queries in the SELECT and FROM clause.

```
SELECT s.i_id
FROM stocks s, (
 SELECT w.w_id
 FROM warehouses w
 WHERE w.w_city = 'Singapore') AS w1
WHERE s.w_id = w1.w_id;
```

```
SELECT s.i_id
FROM stocks s INNER JOIN (
 SELECT w.w_id
 FROM warehouses w
 WHERE w.w_city = 'Singapore') AS w1
ON s.w_id = w1.w_id;
```

Today’s agenda

- Refresher
  - SQL
  - Entity-Relationship model
- Design theory
  - Normal forms & functional dependencies
  - Boyce-Codd normal form
  - 3NF

Database design process

1. Requirements analysis
    - What is stored?
    - How to use?
    - Who should access the data?

1. Requirements Analysis → 2. Conceptual Design → 3. Logical, Physical, Security, etc.

Database design process

2. Conceptual design
   - A high-level description of the DB
   - Sufficiently precise so that tech people can understand
   - But not too precise so that non-tech people can't understand

This is where Entity/Relations fits in.

1. Requirements Analysis → 2. Conceptual Design → 3. Logical, Physical, Security, etc.

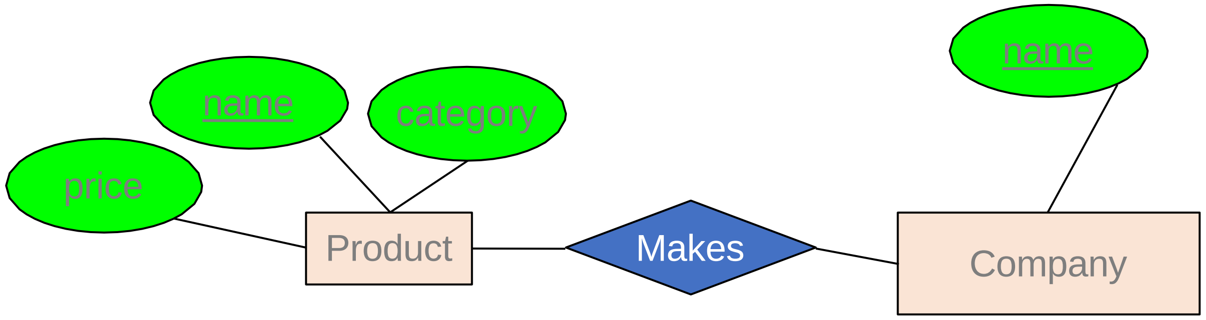
Database design process

3. More

    - Logical DB design
    - Physical DB design
    - Security design

E/R model & diagrams used

E/R is a visual syntax for DB design which is *precise enough* for technical points, but *abstracted enough* for non-technical people



Iterative process

1. Requirements Analysis
2. Conceptual Design
3. Logical, Physical, Security, etc.

E: Entities and Entity Set

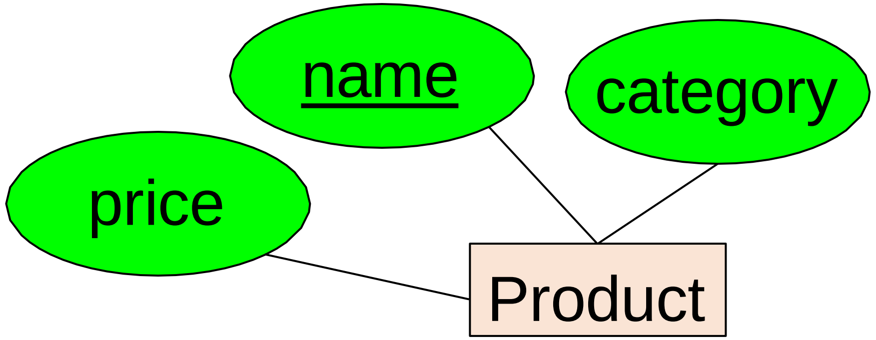
Entities are the individual objects (no associated methods)

Entity sets are collections of similar entities

- Represented by rectangles

An entity set has attributes

- Represented by ovals attached to an entity set

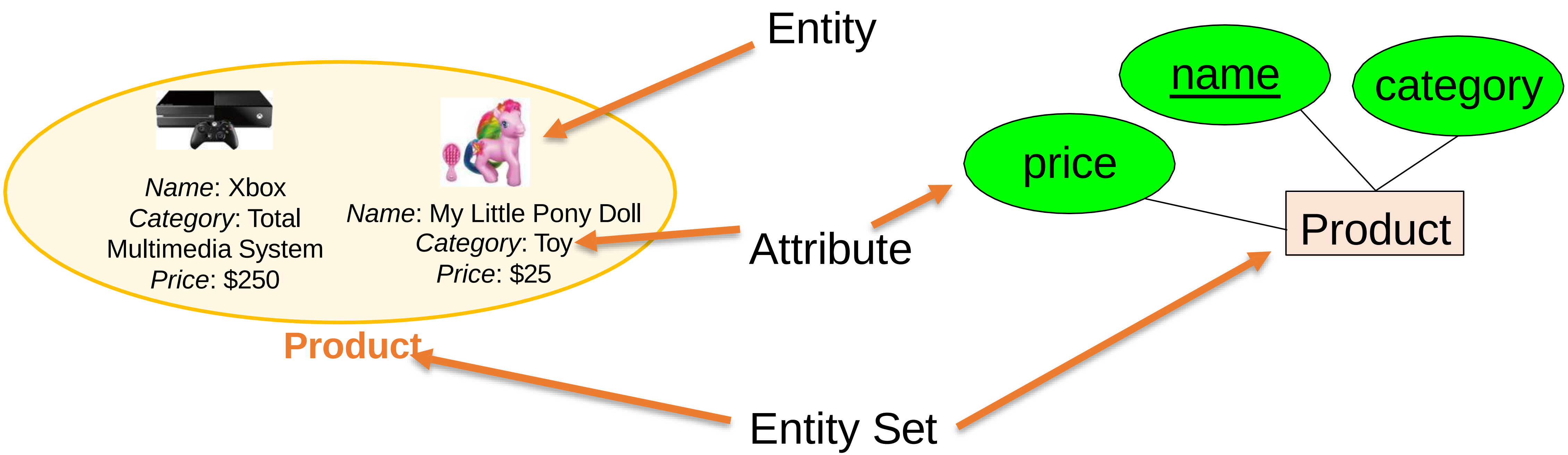


Shapes are important.
Colors are not.

Entities vs. Entity Sets

Example:

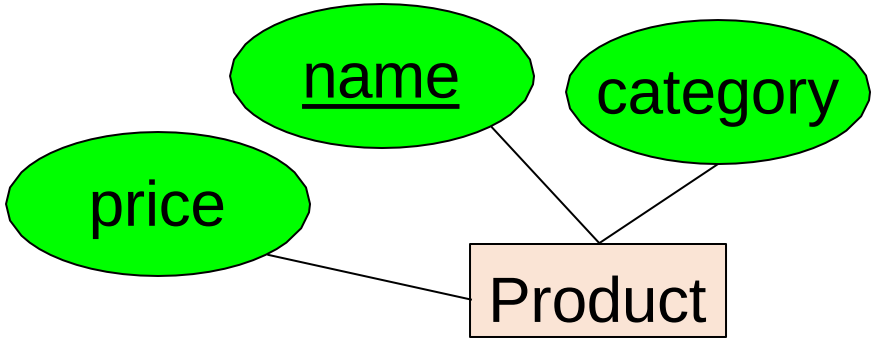
Entities are not explicitly represented in E/R diagrams!



Keys

A key is a minimal set of attributes that uniquely identifies an entity.

Denote elements of the primary key by underlining.

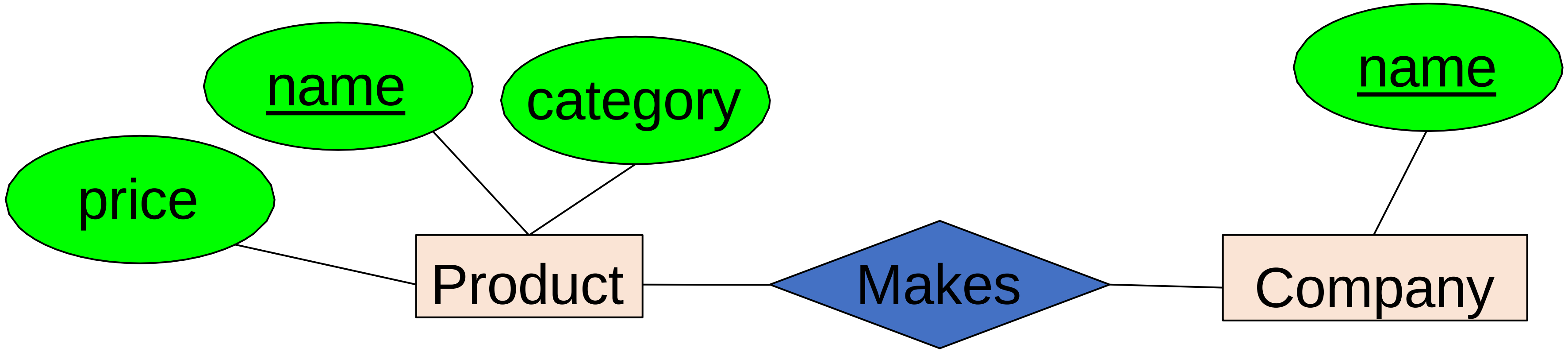


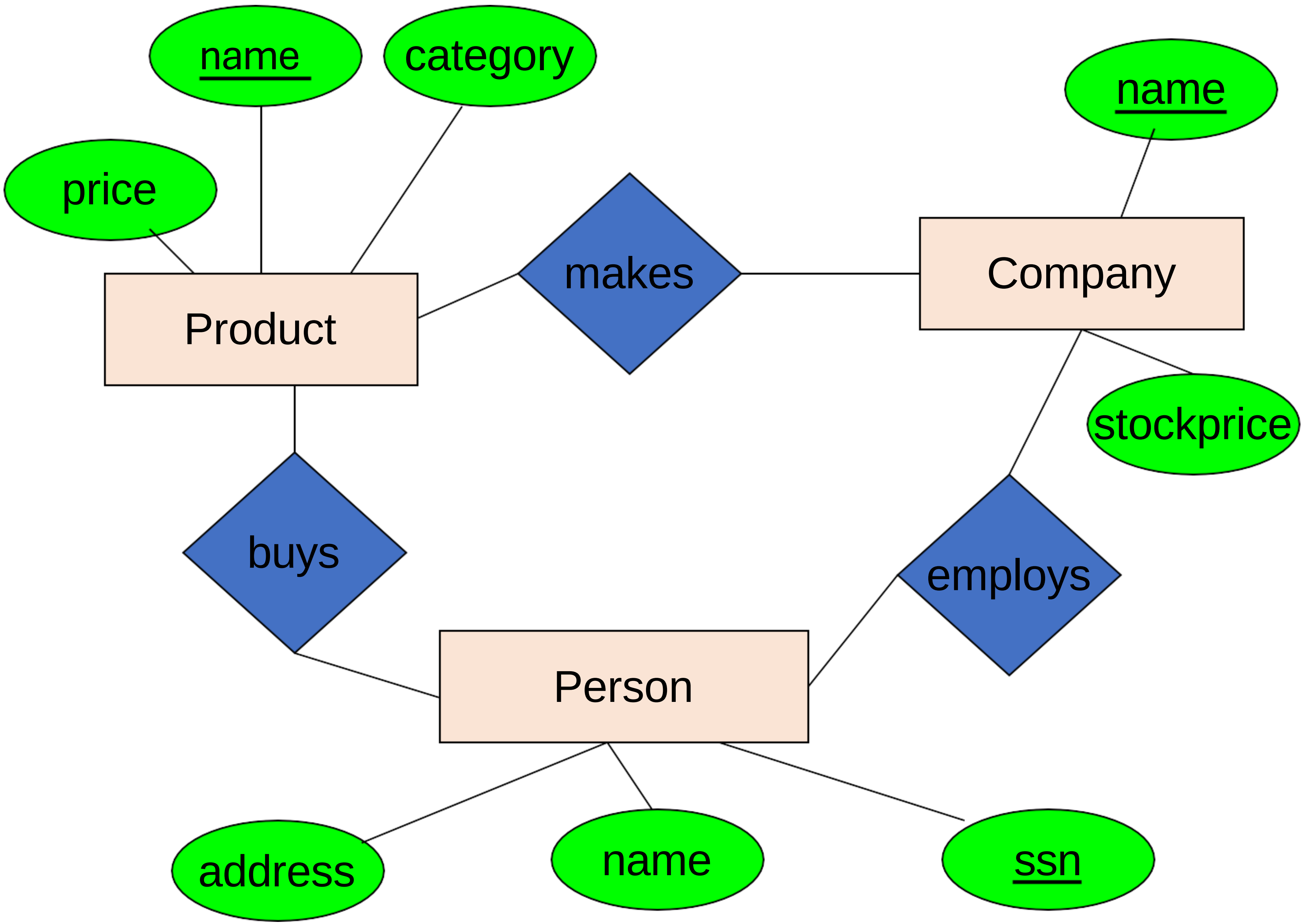
The E/R model forces us to designate a single primary key, though there may be multiple candidate keys

R: Relationships

A relationship is between two entities

- Represented by diamonds

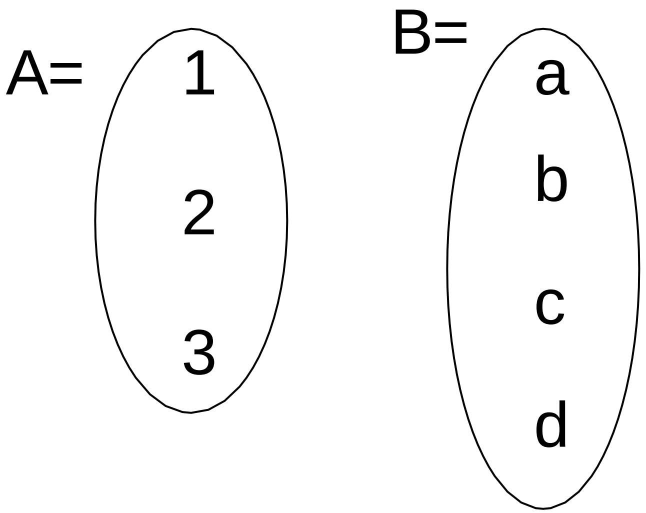



name
category
price
name
makes
Product
Company
stockprice
buys
employs
Person
address
name
ssn

What is a relationship?

A mathematical definition:

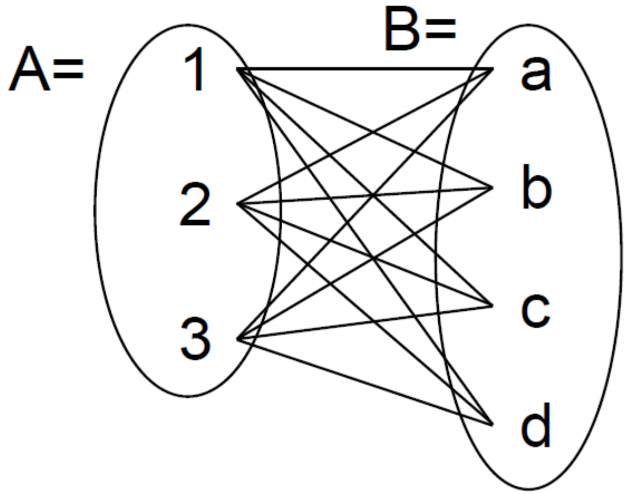
- Let A, B be sets
  - A={1,2,3},   B={a,b,c,d}



What is a relationship?

A mathematical definition:

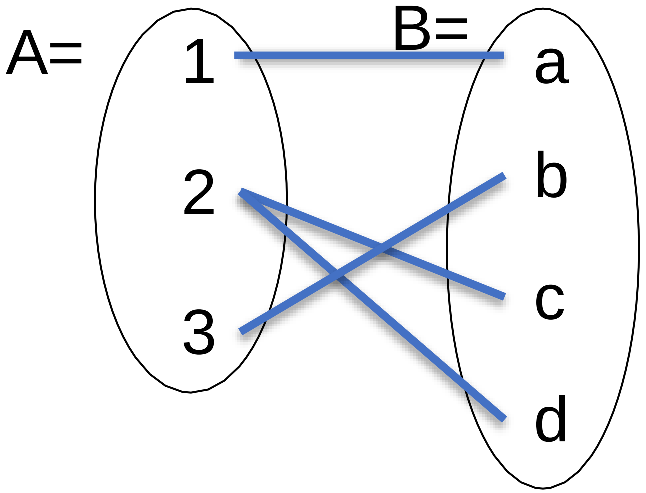
- Let A, B be sets
  - A={1,2,3},   B={a,b,c,d}
- A x B (the cross-product) is the set of all pairs (a,b)
  - A x B = {(1,a), (1,b), (1,c), (1,d), (2,a), (2,b), (2,c), (2,d), (3,a), (3,b), (3,c), (3,d)}



What is a relationship?

A mathematical definition:

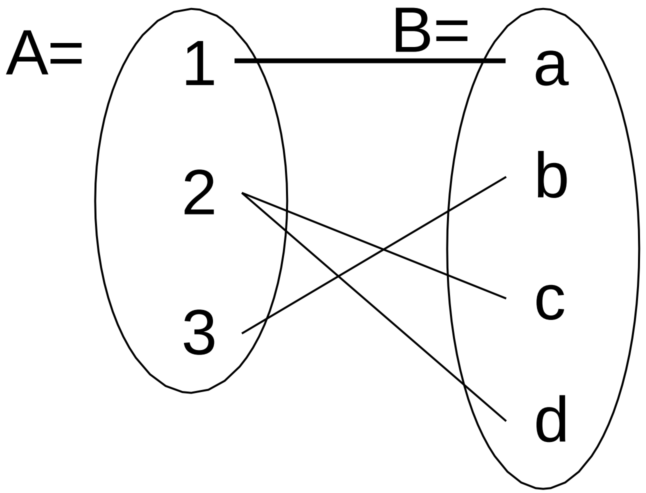
- Let A, B be sets
  - A={1,2,3}, B={a,b,c,d},
- A x B (the cross-product) is the set of all pairs (a,b)
  - A x B = {(1,a), (1,b), (1,c), (1,d), (2,a), (2,b), (2,c), (2,d), (3,a), (3,b), (3,c), (3,d)}
- We define a relationship to be a subset of A x B
  - R = {(1,a), (2,c), (2,d), (3,b)}



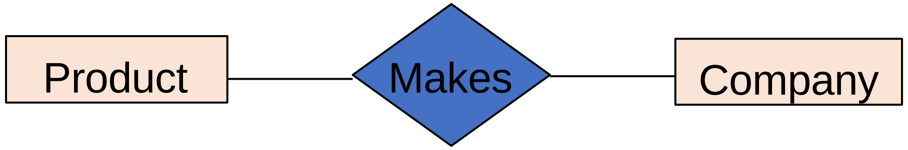
What is a relationship?

A mathematical definition:

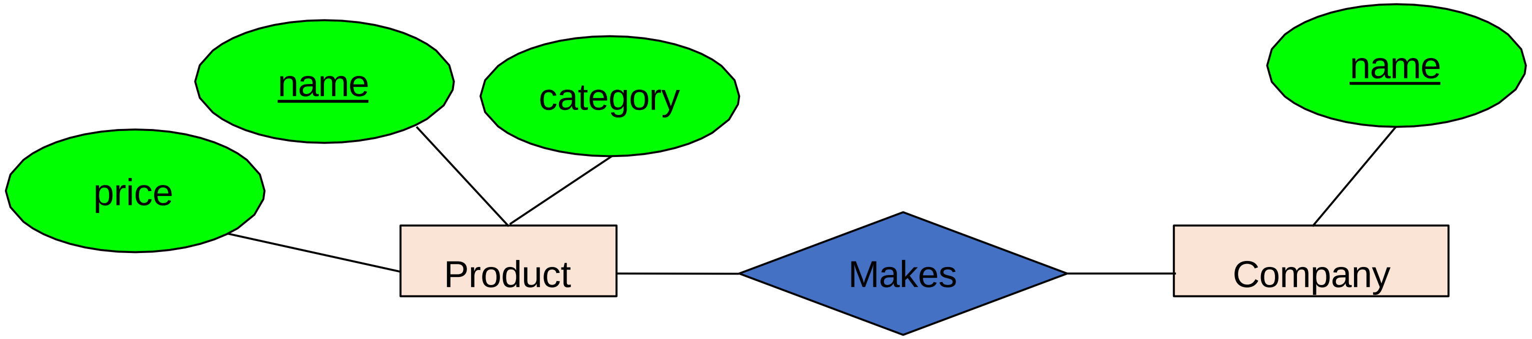
- Let A, B be sets
- A x B (the cross-product) is the set of all pairs
- A relationship is a subset of A x B



Makes is relationship: it is a subset of Product x Company:



What is a relationship?



A relationship between entity sets P and C is a subset of all possible pairs of entities in P and C, with tuples uniquely identified by P and C's keys

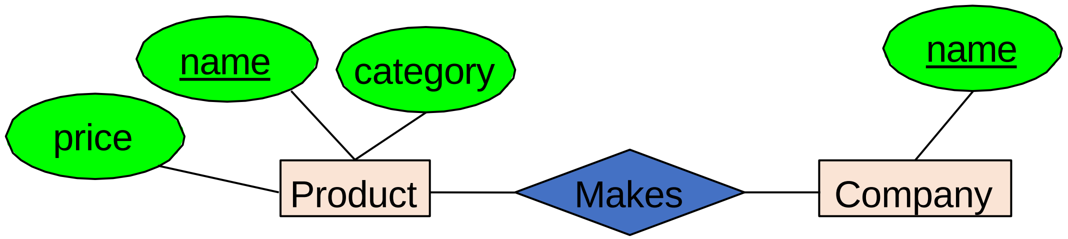
What is a relationship?

Company

name
GizmoWorks
GadgetCorp

Product

name	category	price
Gizmo	Electronics	$9.99
GizmoLite	Electronics	$7.50
Gadget	Toys	$5.50



Company C × Product P

C.name	P.name	P.category	P.price
GizmoWorks	Gizmo	Electronics	$9.99
GizmoWorks	GizmoLite	Electronics	$7.50
GizmoWorks	Gadget	Toys	$5.50
GadgetCorp	Gizmo	Electronics	$9.99
GadgetCorp	GizmoLite	Electronics	$7.50
GadgetCorp	Gadget	Toys	$5.50

Makes

C.name	P.name
GizmoWorks	Gizmo
GizmoWorks	GizmoLite
GadgetCorp	Gadget

A relationship between entity sets P and C is a subset of all possible pairs of entities in P and C, with tuples uniquely identified by P and C's keys

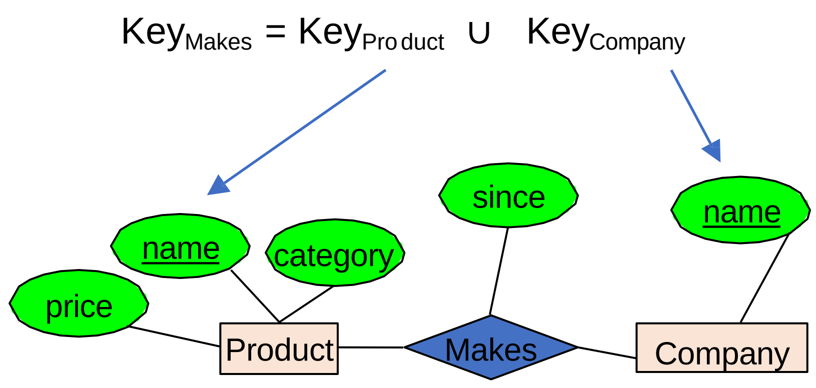
What is a relationship?

There can only be one relationship for every unique combination of entities

This follows from our mathematical definition of a relationship- it's a SET!

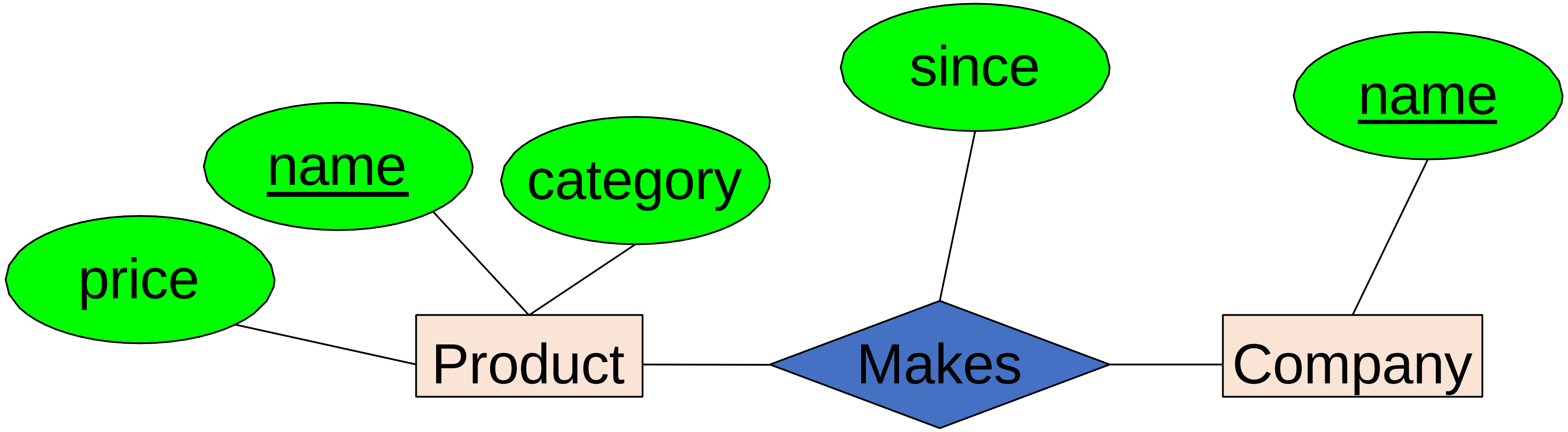
This also means that the relationship is uniquely determined by the keys of its entities

Example: the "key" for Makes (to right) is {Product.name, Company.name}



Relationships and attributes

Relationships may have attributes as well.



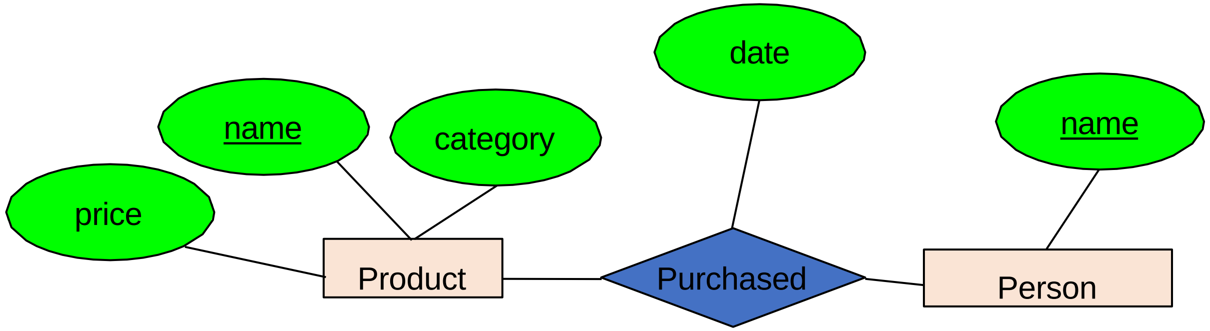
For example: "since" records when company started making a product

Note: "since" is implicitly unique per pair here! Why?

Note #2: Why not "how long"?

Decision: relationship vs. entity?

Q: What does this say?

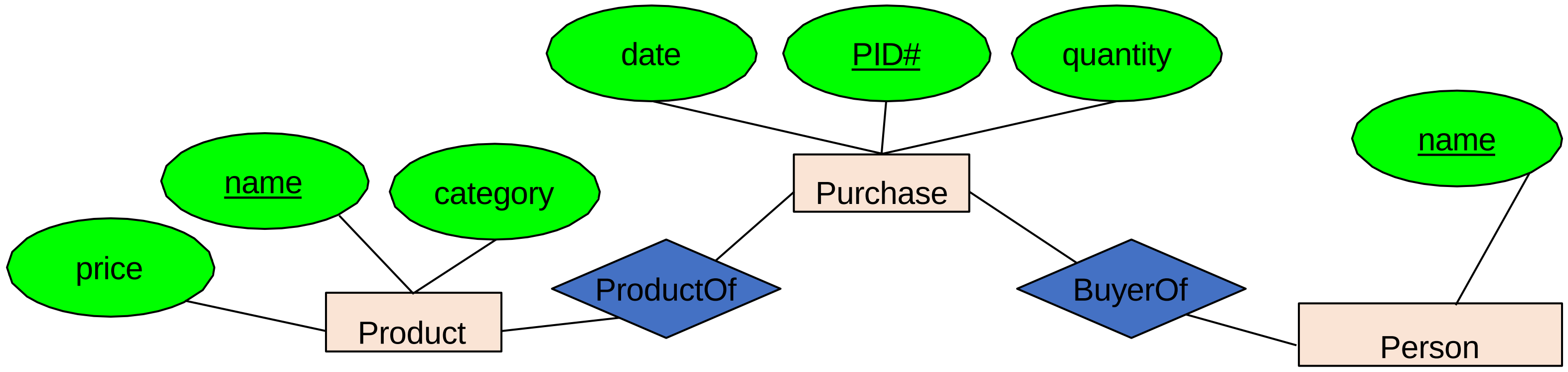


A: A person can only buy a specific product once (on one date)

Modeling something as a relationship makes it unique;
what if not appropriate?

Decision: relationship vs. entity?

What about this way?



Now we can have multiple purchases per product, person pair!

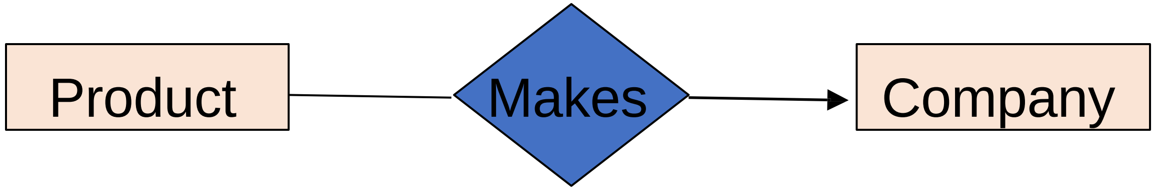
We can always use a new entity instead of a relationship. For example, to permit multiple instances of each entity combination!

Multiplicity of binary relationships

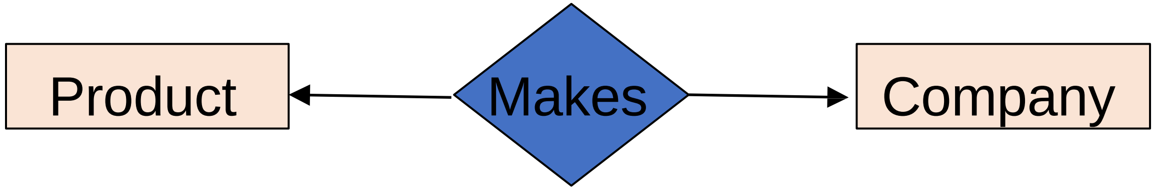
Relationships can be one-one, one-many, or many-many

An arrow indicates “related to at most one entity”

- Different than “exactly one”



A product has at most one company

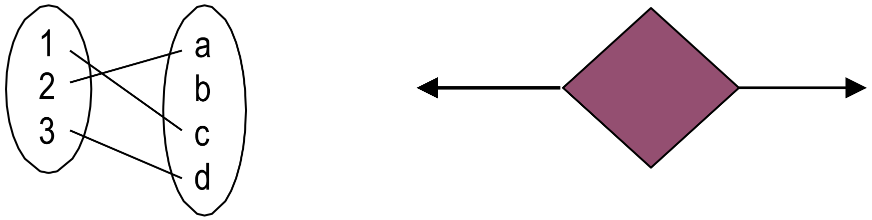


A product has at most one company, and a company has at most one product

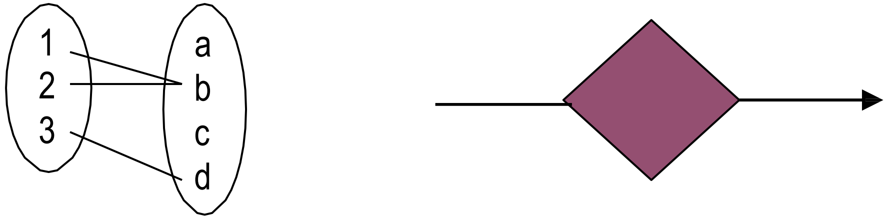
What kind of relationship does the bottom diagram imply?

Multiplicity of binary relationships

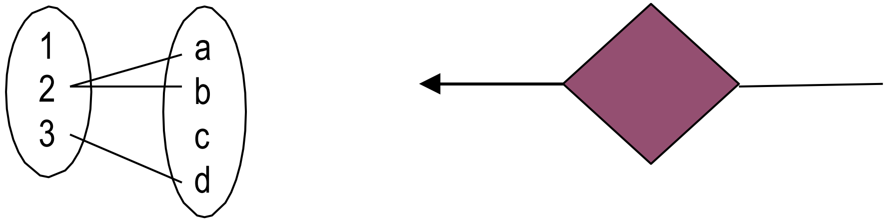
One-to-one:



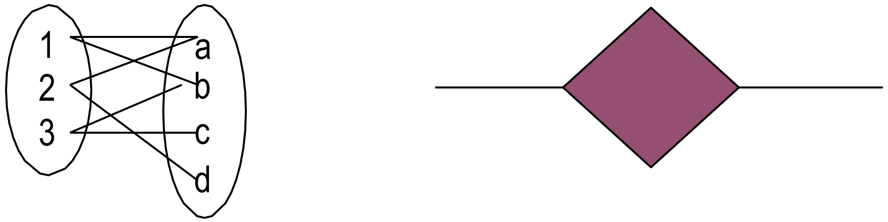
Many-to-one:



One-to-many:

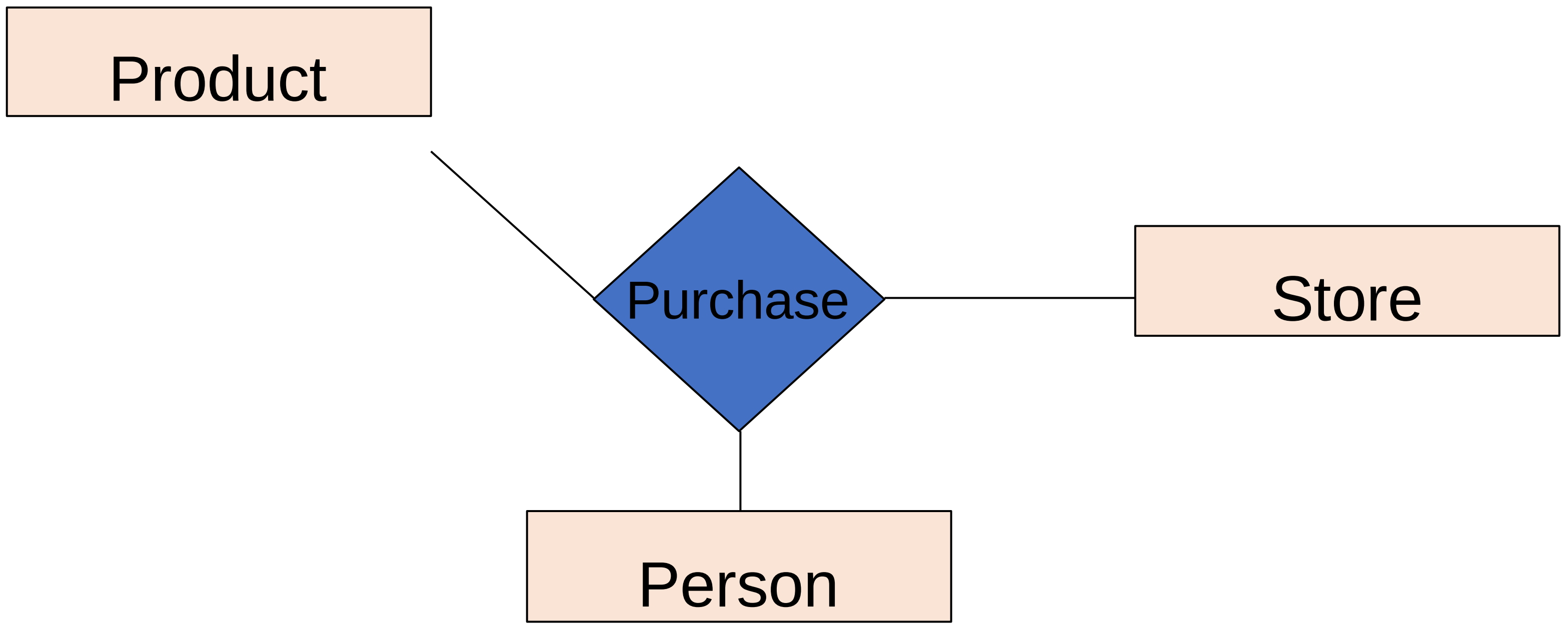


Many-to-many:



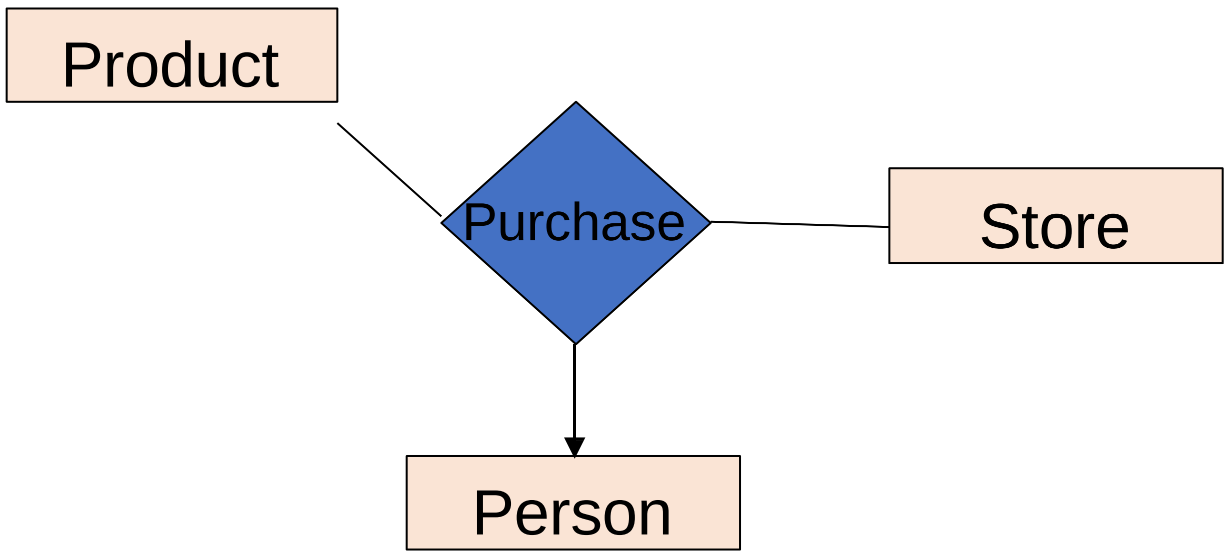
Multiway relationships

How do we model a purchase relationship between buyers, products and stores?



Arrows in multiway relationships

Q: What does the arrow mean ?



Arrow: if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E.

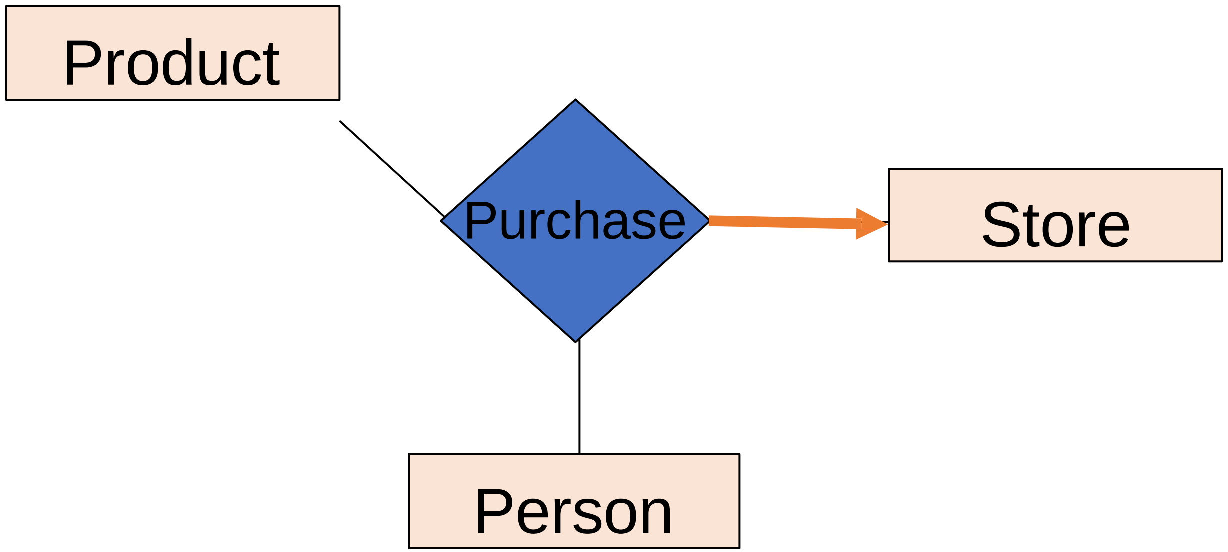
For each (product, store), there is at most one person who have made that purchase

Q: Can a person purchase two different products the same store?

Q: Can a person purchase the same product at two different stores?

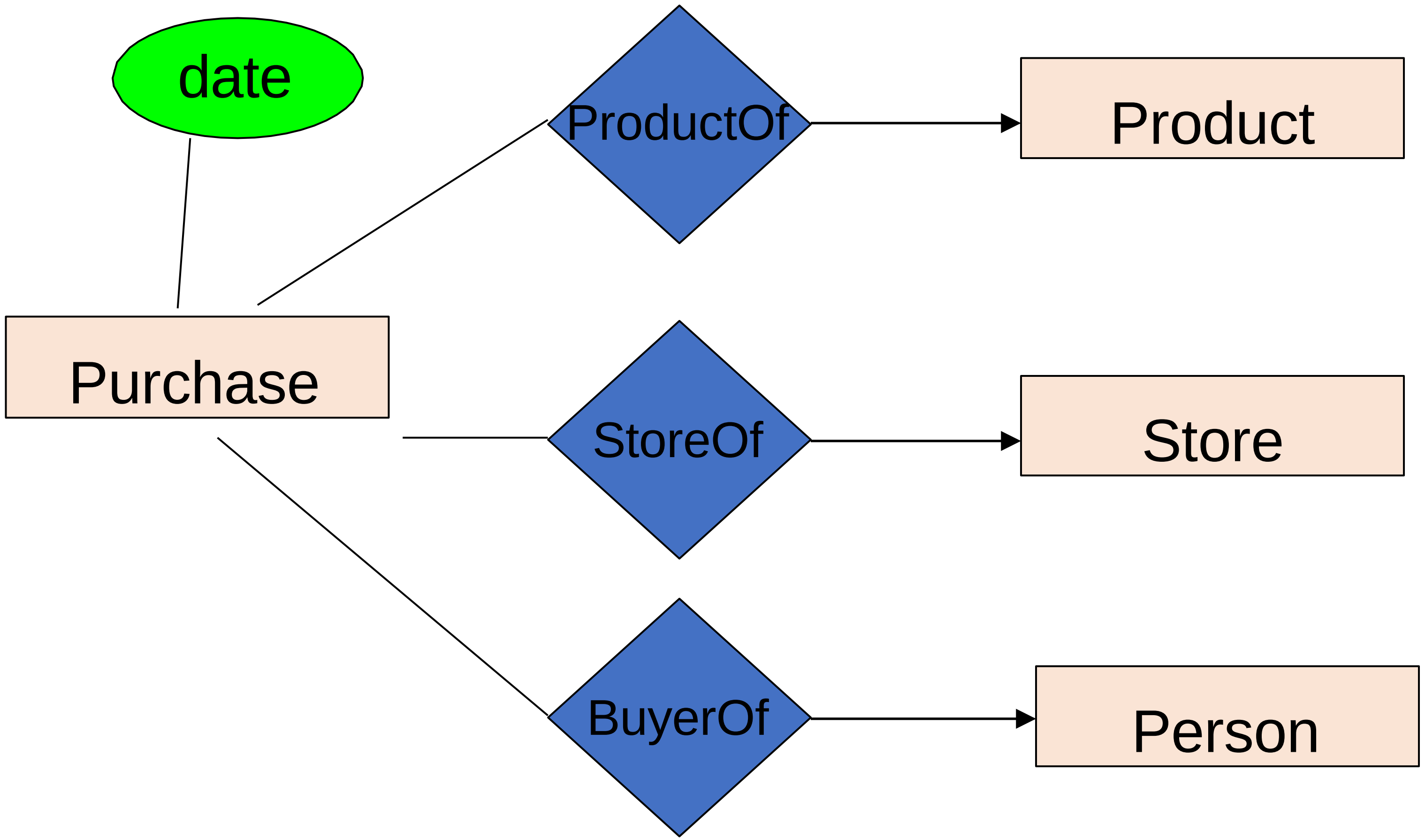
Arrows in multiway relationships

Q: How do we say that every person shops in at most one store ?

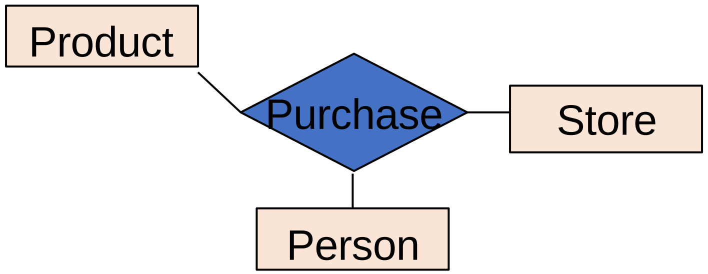


A: Cannot. This is the best approximation.
(Why only approximation ?)

Converting multi-way relationships to binary

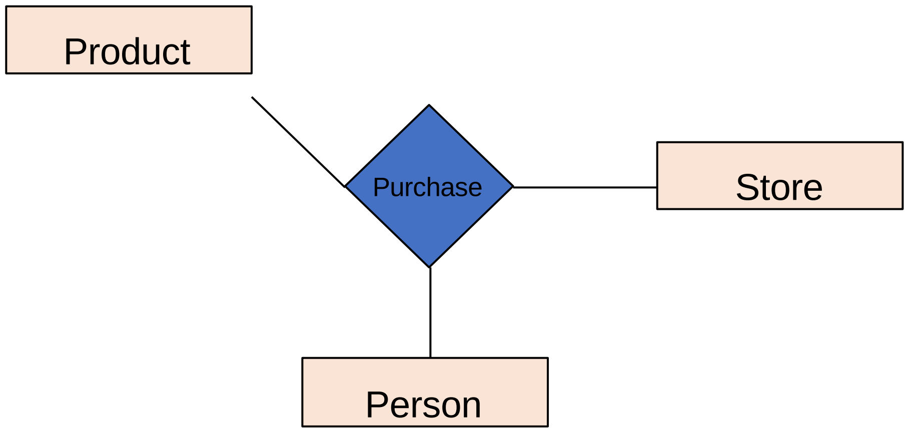


- Create a connecting entity set
- Introduce many-one relationships

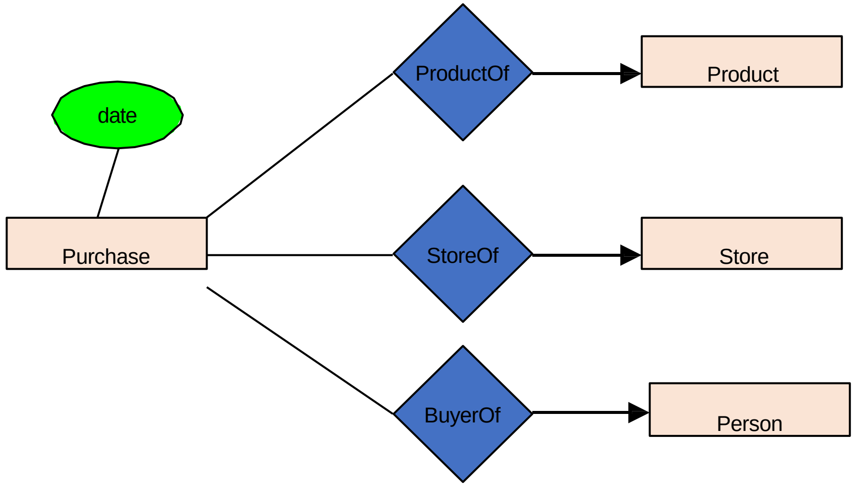


Decision: multi-way or new entity + binary?

(A) Multi-way Relationship



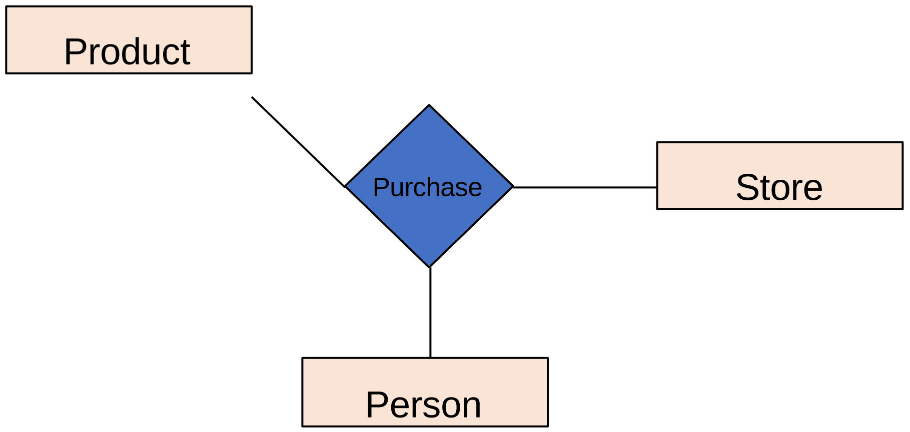
(B) Entity + Binary



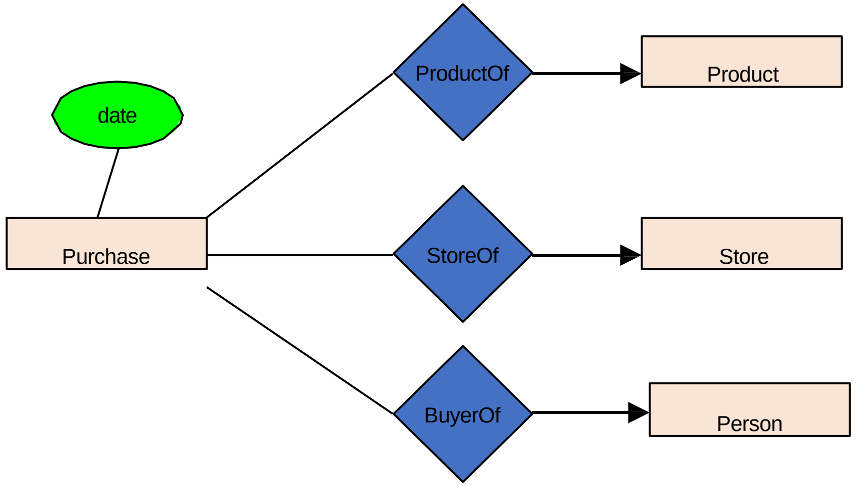
Should we use a single multi-way relationship or a new entity with binary relations?

Decision: multi-way or new entity + binary?

(A) Multi-way Relationship



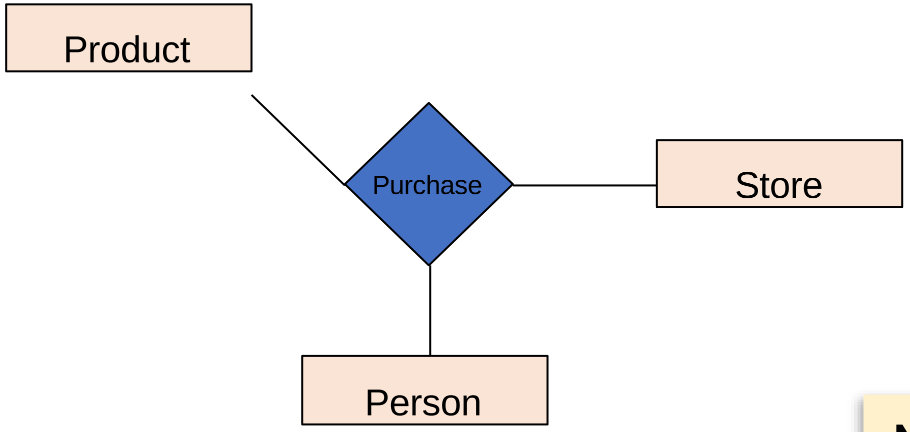
(B) Entity + Binary



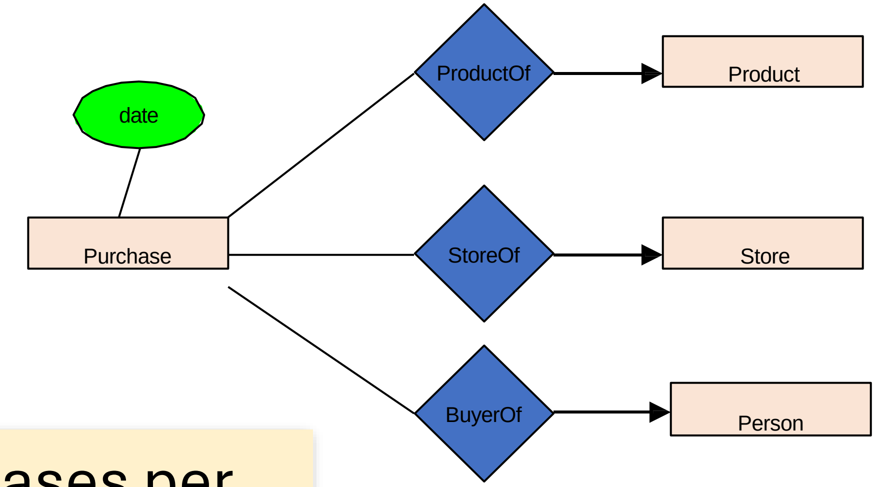
- (A) is useful when a relationship really is between multiple entities
  - Ex: A three-party legal contract

Decision: multi-way or new entity + binary?

(A) Multi-way Relationship



(B) Entity + Binary

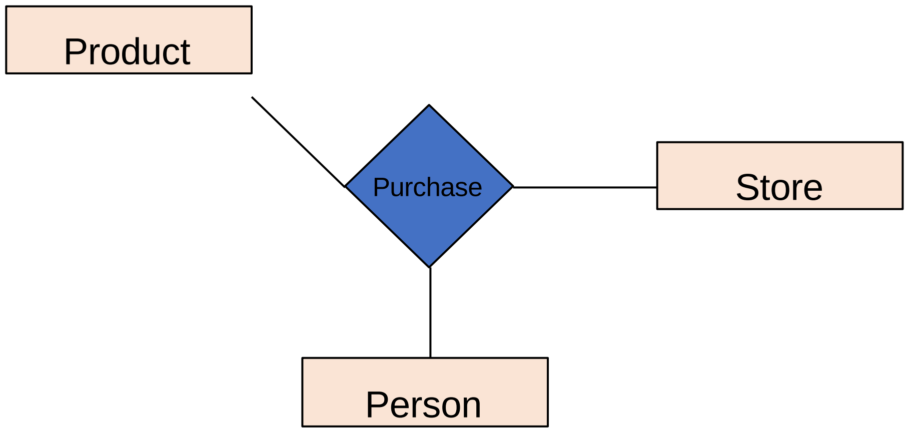


Multiple purchases per (product, store, person) combo possible here!

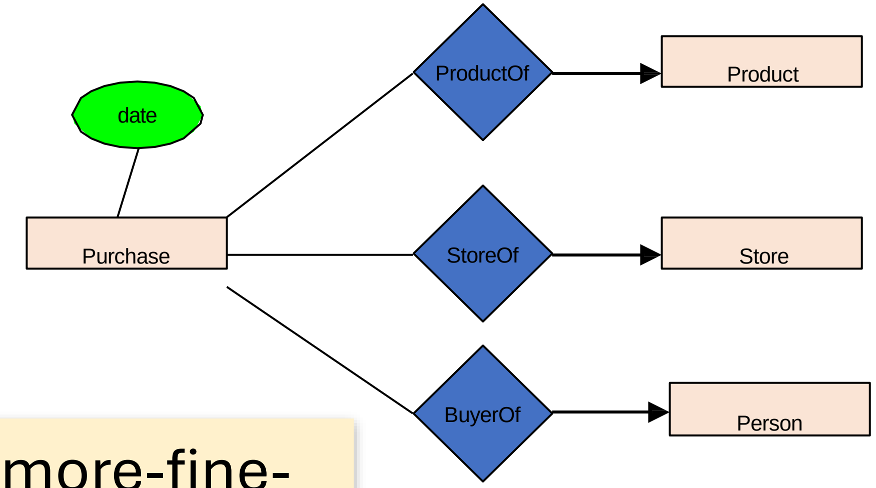
- Covered earlier: (B) is useful if we want to have multiple instances of the "relationship" per entity combination

Decision: multi-way or new entity + binary?

(A) Multi-way Relationship



(B) Entity + Binary

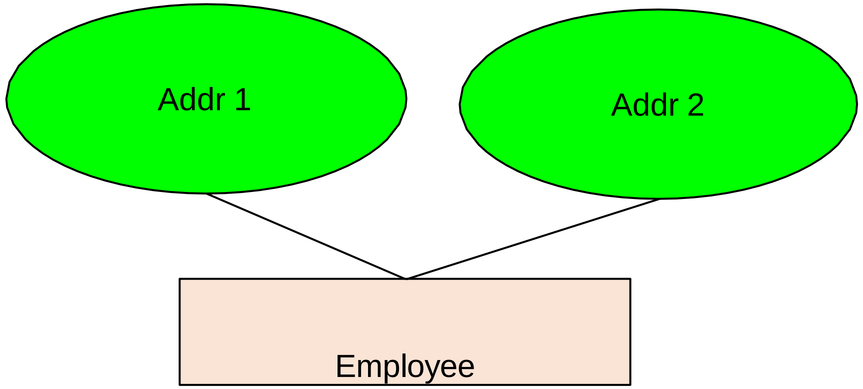


We can add more-fine-grained constraints here!

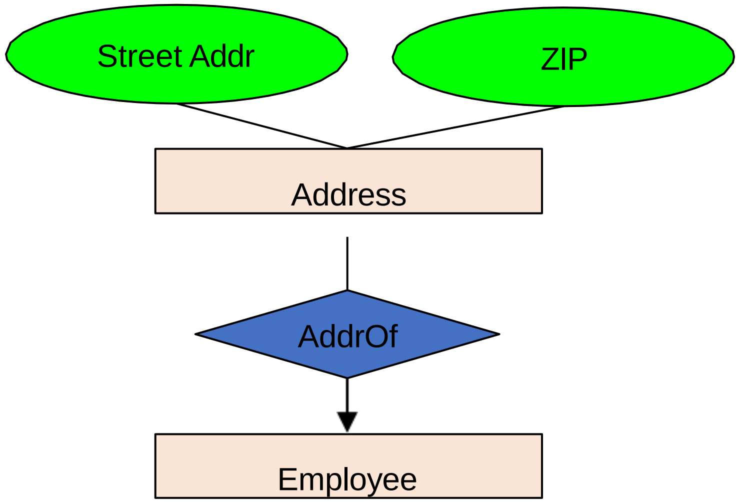
- (B) is also useful when we want to add details (constraints or attributes) to the relationship
  - "A person who shops in only one store"
  - "How long a person has been shopping at a store"

Examples: entity vs. attribute

Should address (A) be an attribute?

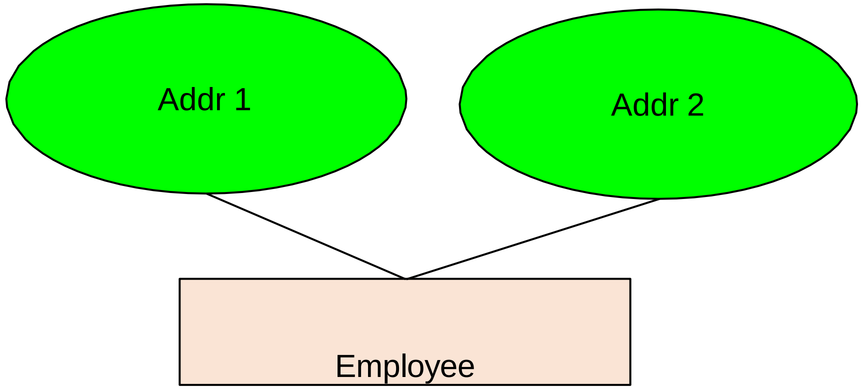


Or (B) be an entity?



Examples: entity vs. attribute

Should address (A) be an attribute?

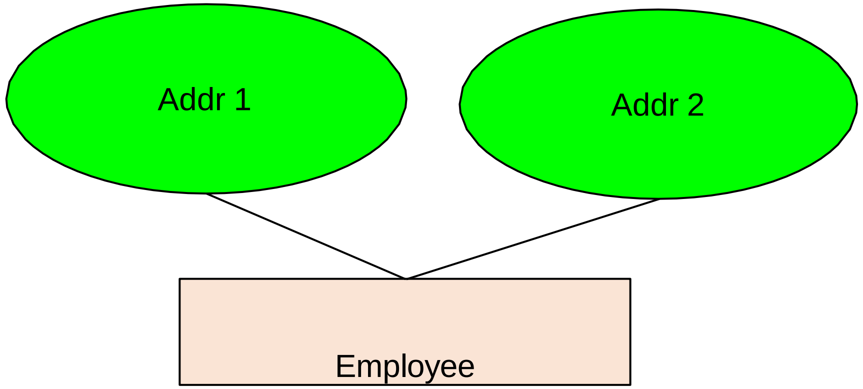


How do we handle employees with multiple addresses here?

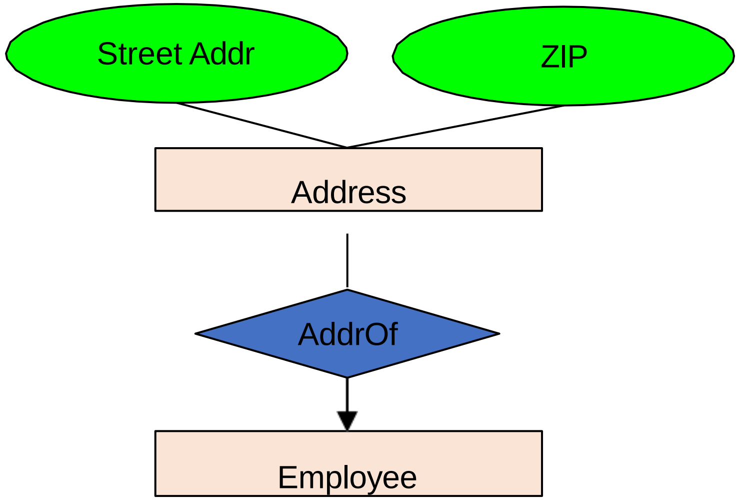
How do we handle addresses where internal structure of the address (e.g. zip code, state) is useful?

Examples: entity vs. attribute

Should address (A) be an attribute?



Or (B) be an entity?



In general, when we want to record several values, we choose new entity

Constraints in E/R Diagrams

Commonly used constraints are:

Keys: Implicit constraints on uniqueness of entities

- Ex: An SSN uniquely identifies a person

Single-value constraints:

- Ex: a person can have only one father

Referential integrity constraints: Referenced entities must exist

- Ex: if you work for a company, it must exist in the database

Recall FOREIGN KEYs!

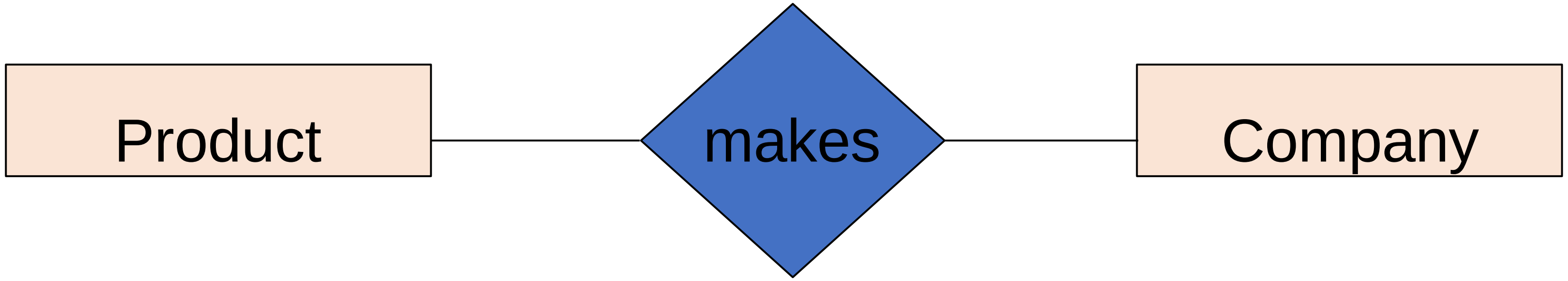
Participation constraints:

- Ex: every student must enroll in a class

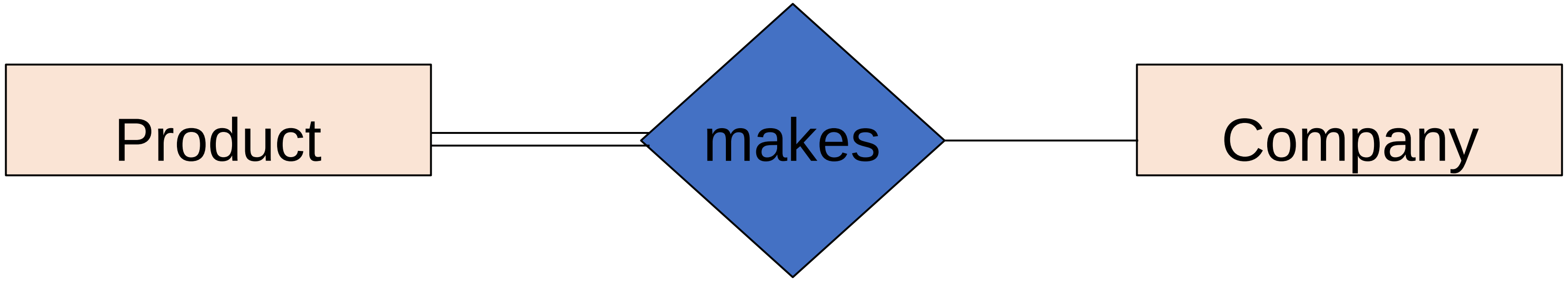
Other constraints:

- Ex: peoples' ages are between 0 and 150

Participation constraints: partial v. total



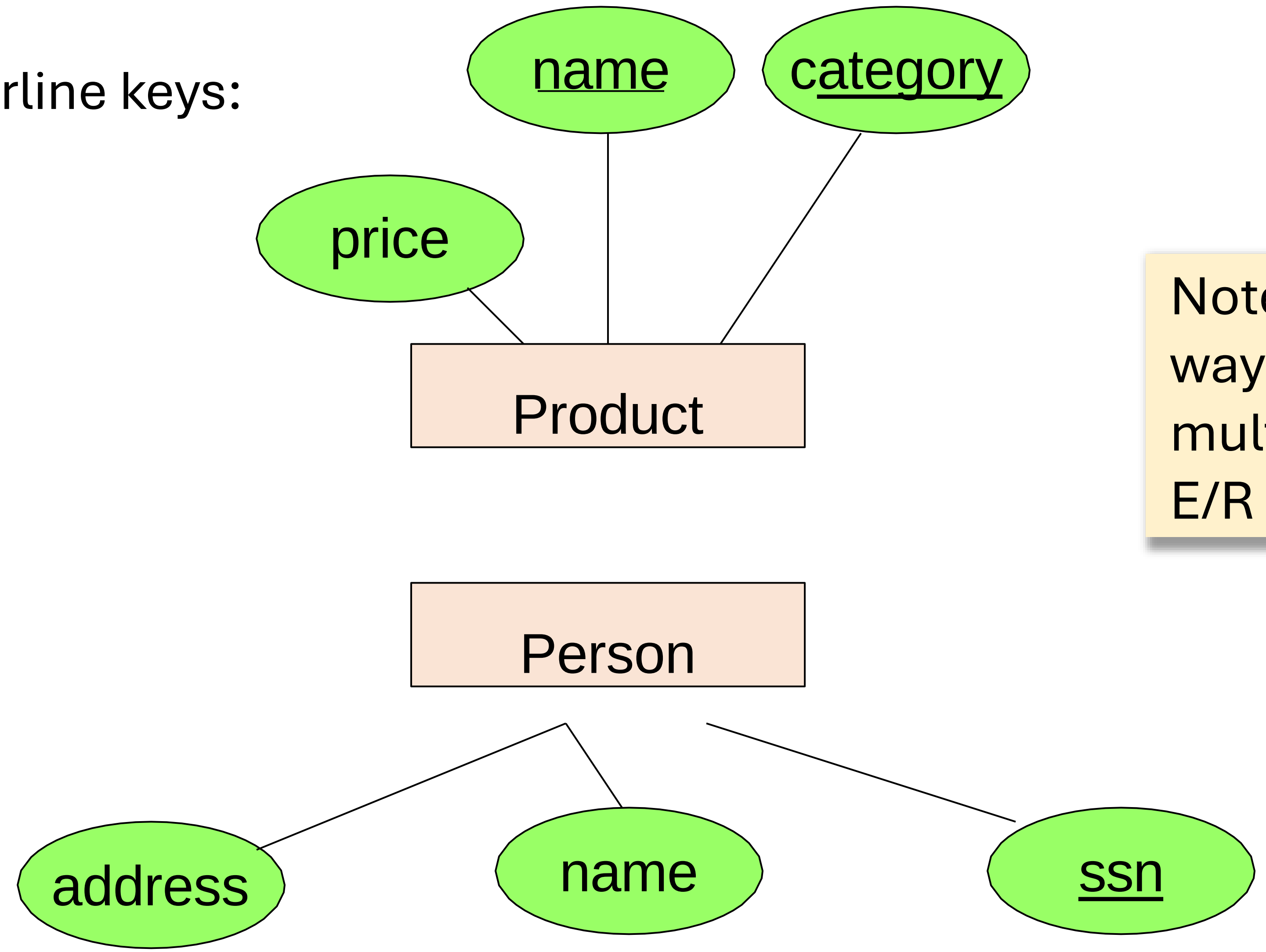
Are there products made by no company?
Companies that don't make a product?



Double line indicates <u>total participation</u> (i.e. here: all products are made by a company)

Key constraints

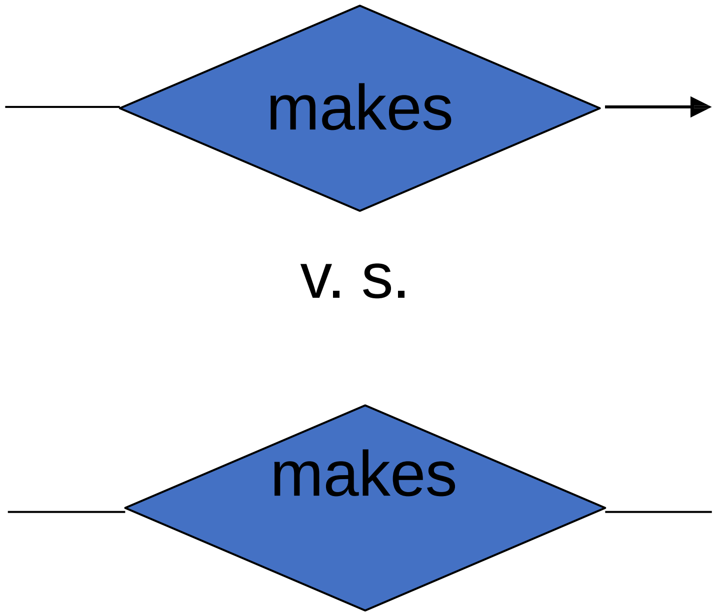
Underline keys:



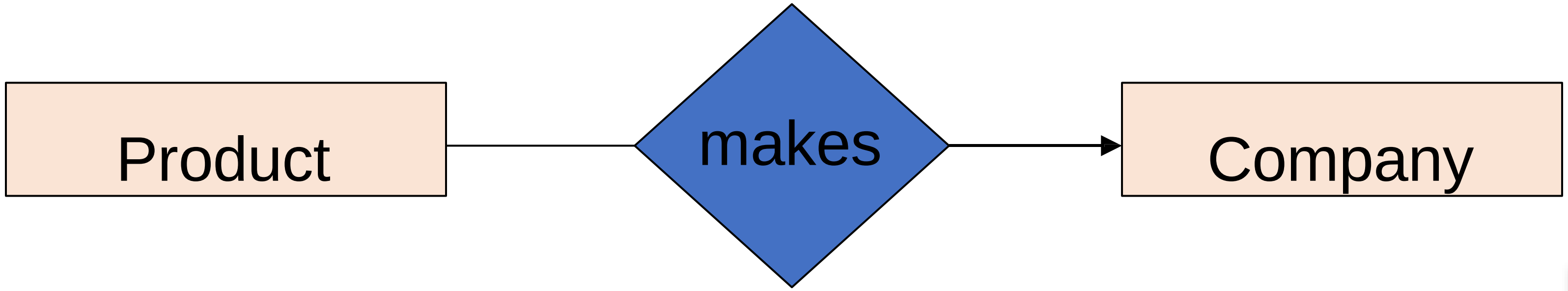
Note: no formal way to specify multiple keys in E/R diagrams…

Single value constraints

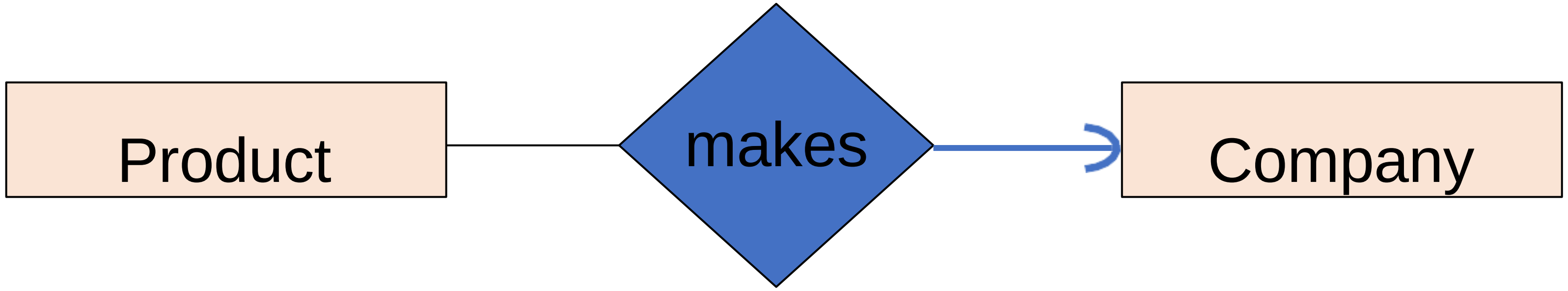
See previous section!



Referential integrity constraints



Each product made by at most one company.
Some products made by no company?

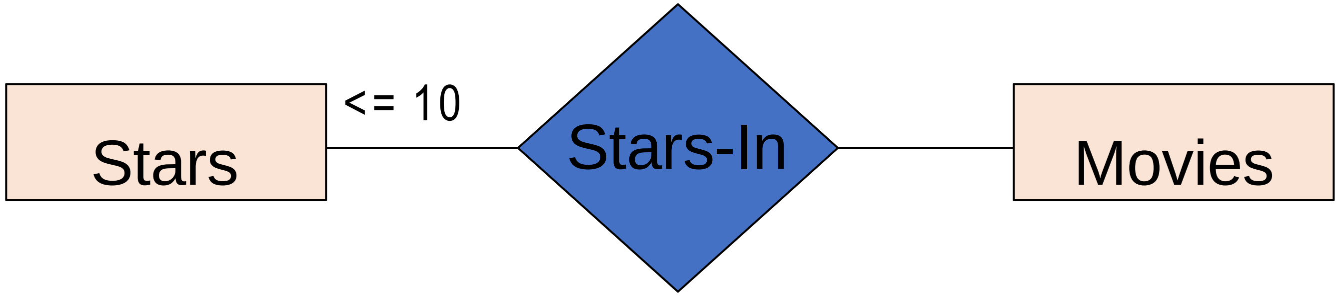


Each product made by <u>exactly one</u> company.

A rounded arrow to F means
- The relationship is many-one and
- The entity of set F related to an entity of E must exist

Degree constraints

- Limit the number of entities connected to any one entity of the related entity set
  - Arrow is same as “<=1” constraint
  - Rounded arrow is same as “=1” constraint



Every movie can be connected to at most 10 stars

From E/R diagrams to relational schema

- Key concept:

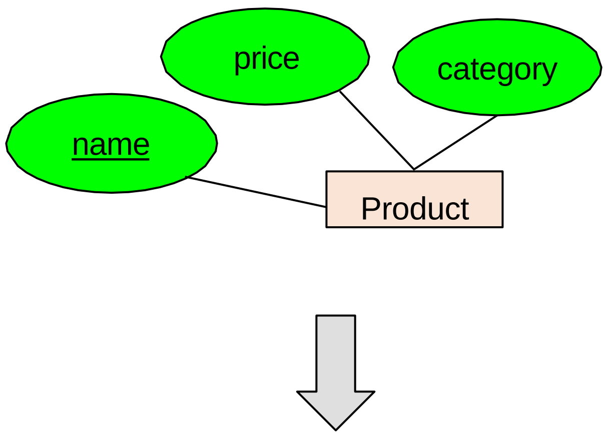
Both <u>Entity sets</u> and <u>Relationships</u> become relations (tables in RDBMS)

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

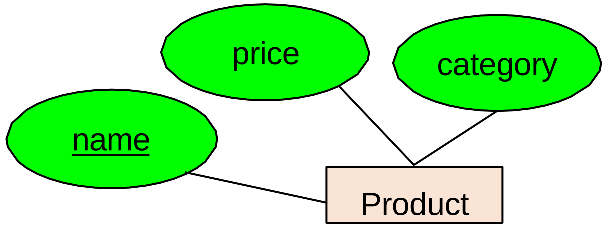


Product

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    DOUBLE,
  category VARCHAR(30)
)
```



Product

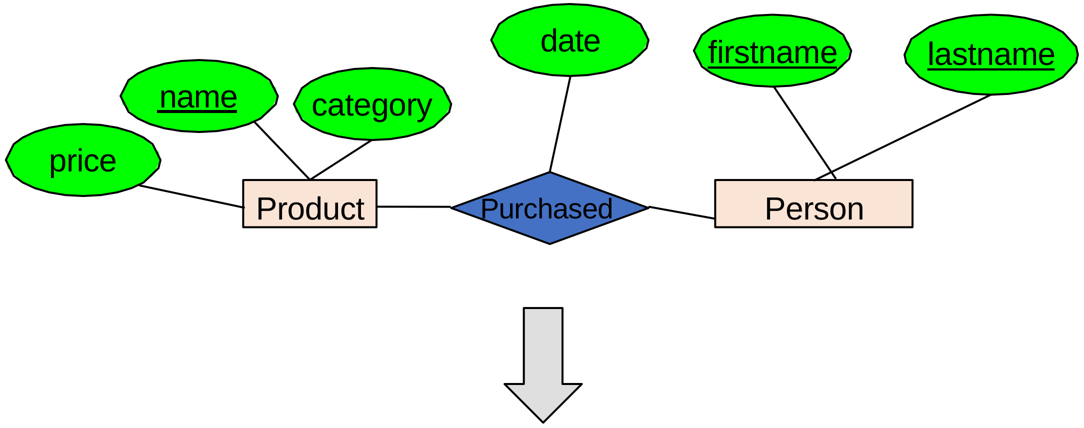
name	price	category
Gizmo1	99.99	Camera
Gizmo2	19.99	Edible

From E/R diagrams to relational schema

A relation between entity sets $A_1, ..., A_N$ 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 union of the entity sets' keys as primary key
  - has the entities' primary keys as foreign keys

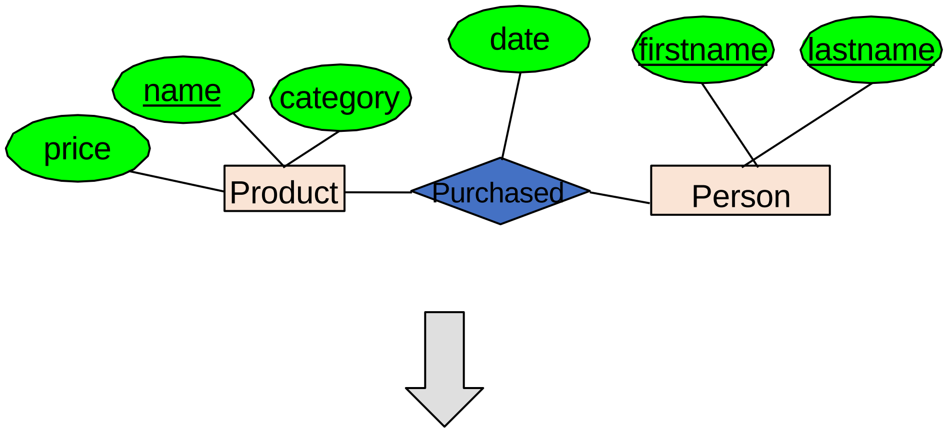


Purchased

name	firstname	lastname	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

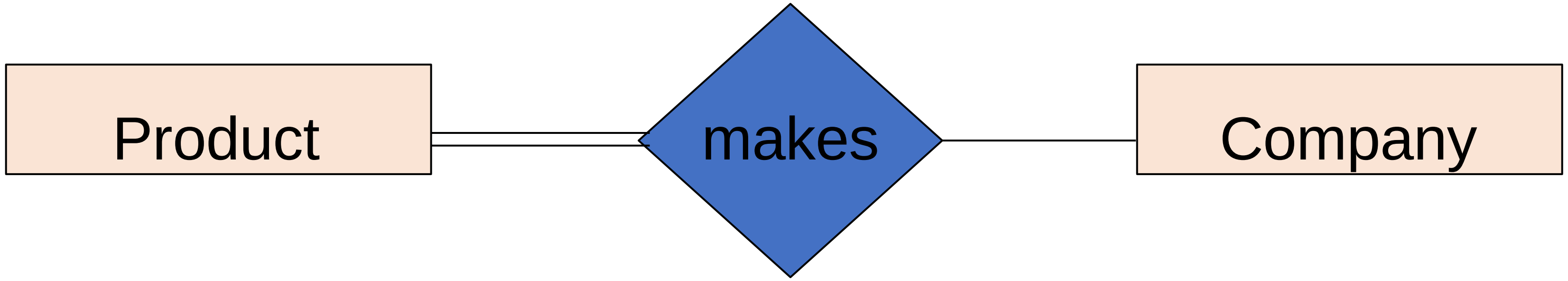
```
CREATE TABLE Purchased(
  name      CHAR(50),
  firstname CHAR(50),
  lastname  CHAR(50),
  date      DATE,
  PRIMARY KEY (name, firstname, lastname),
  FOREIGN KEY (name)
            REFERENCES Product,
  FOREIGN KEY (firstname, lastname)
            REFERENCES Person
)
```



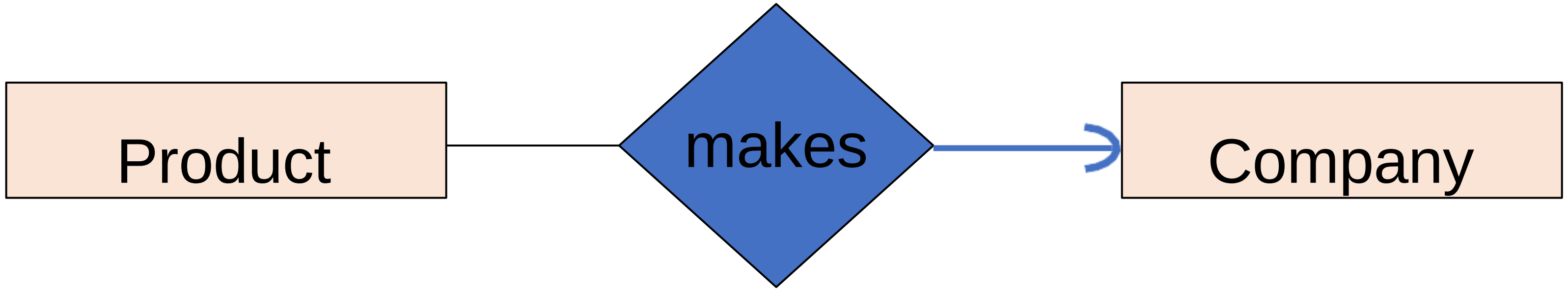
Purchased

<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

Note: total participation vs. referential integrity



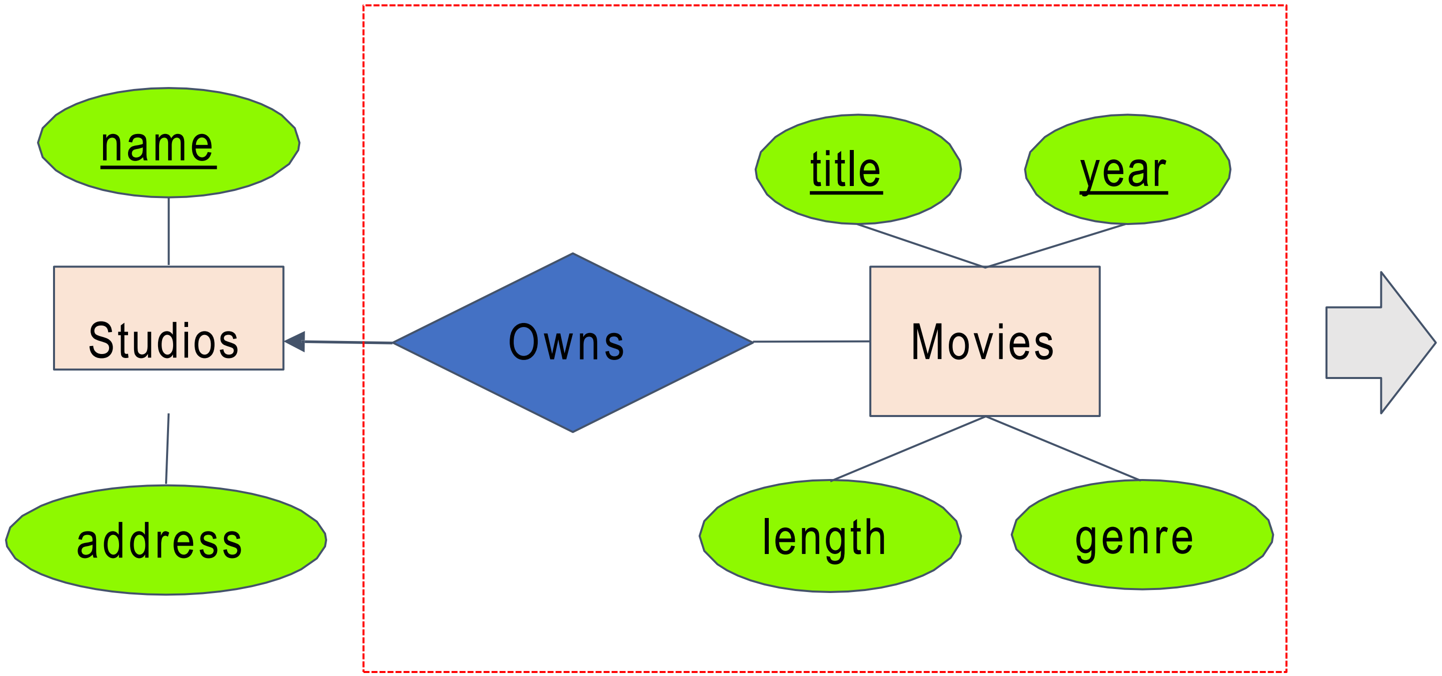
All products need to be in the makes relationship



Each product made by <u>exactly one</u> company;
the company involved must exist in our database.

Combining relations

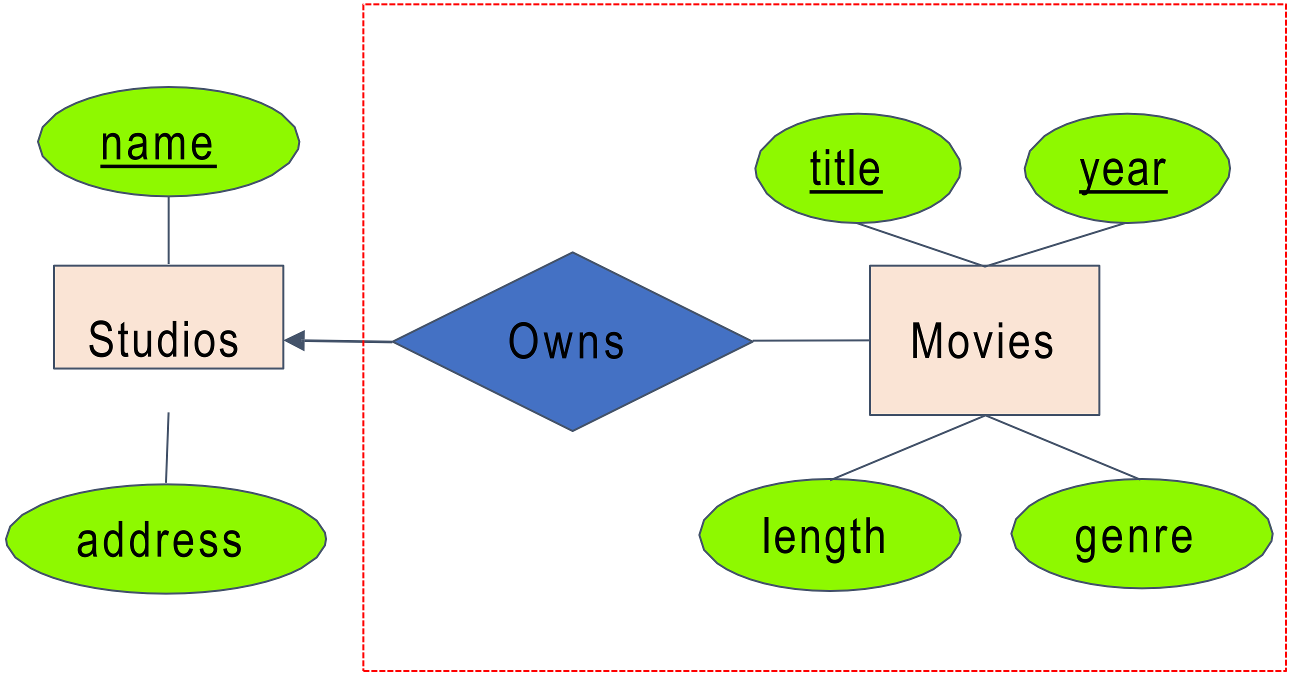
- If E is connected to F through a many-one relationship R, combine E and R
  - Attributes of E and R, and the key attributes of F
- Advantage: querying one relation is faster than querying several relations




```
Owns(title, year, length,
      genre, studioName)
```

Combining relations

- Why only consider many-one relationships?
  - Otherwise, the combined relation is not good design and may contain anomalies



This information is redundant, and updating one tuple may leave the other one incorrect (update anomaly)

Owns

title	year	length	genre	studioName
t1	y1	l1	g1	n1
t1	y1	l1	g1	n2
t2	y2	l2	g2	n2

Today’s agenda

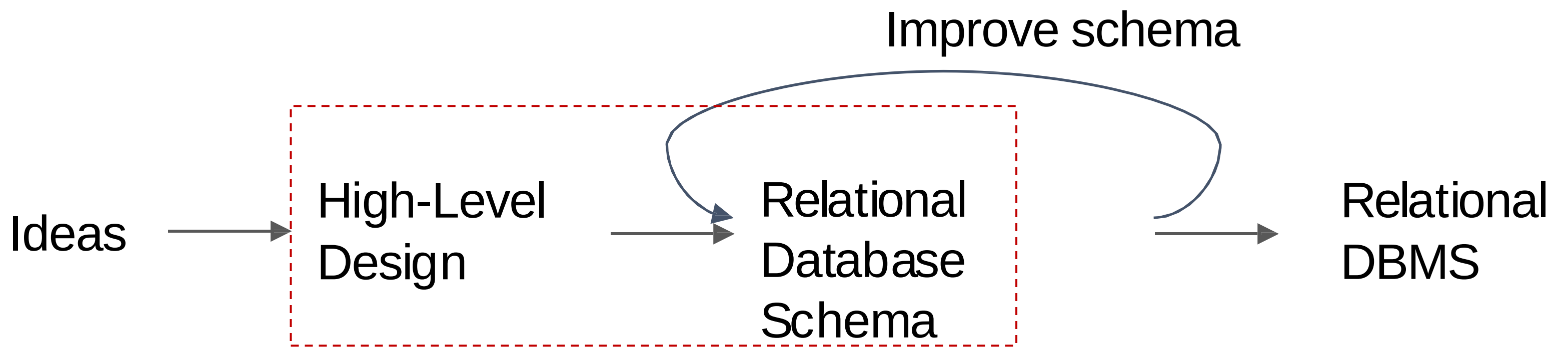
- Refresher
  - SQL
  - Entity-Relationship model
- Design theory
  - Normal forms & functional dependencies
  - Boyce-Codd normal form
  - 3NF

Design theory for relational databases

There are many ways to design a relational database schema
- E.g., we just learned how to use an E/R diagram

It is also common to improve the initial schema (esp. eliminating redundancy)
- Often, the problem is combining too much into one relation

Fortunately, there is a well-developed design theory for good schema design
- Functional dependencies, normalization, multivalued dependencies
- One of the reasons Databases are powerful and so widely used



Normal forms

- 1st Normal Form (1NF) = All tables are flat
- 2nd Normal Form = disused
- Boyce-Codd Normal Form (BCNF)
- 3rd Normal Form (3NF)

DB designs based on functional dependencies, intended to prevent data anomalies

- 4th and 5th Normal Forms = see textbooks

1st Normal Form (1NF)

Student	Courses
Mary	{CS145,CS229}
Joe	{CS145,CS106}
…	…

Violates 1NF.

Student	Courses
Mary	CS145
Mary	CS229
Joe	CS145
Joe	CS106

In 1st NF

1NF Constraint: Types must be atomic!

Data anomalies

A poorly designed database causes *anomalies*:

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

If every course is in only one room, contains redundant information!

Data anomalies

A poorly designed database causes *anomalies*:

Student	Course	Room
Mary	CS145	B01
Joe	CS145	C12
Sam	CS145	B01
..	..	..

If we update the room number for one tuple, we get inconsistent data = an update anomaly

Data anomalies

A poorly designed database causes *anomalies*:

Student	Course	Room
..	..	..

If everyone drops the class, we lose what room the class is in!
= a delete anomaly

Data anomalies

A poorly designed database causes *anomalies*:

...	CS229	C12

Student	**Course**	**Room**
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Similarly, we can't reserve a room without students = an <u>insert anomaly</u>

Data anomalies

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

Course	Room
CS145	B01
CS229	C12

Eliminate anomalies by decomposing relations.

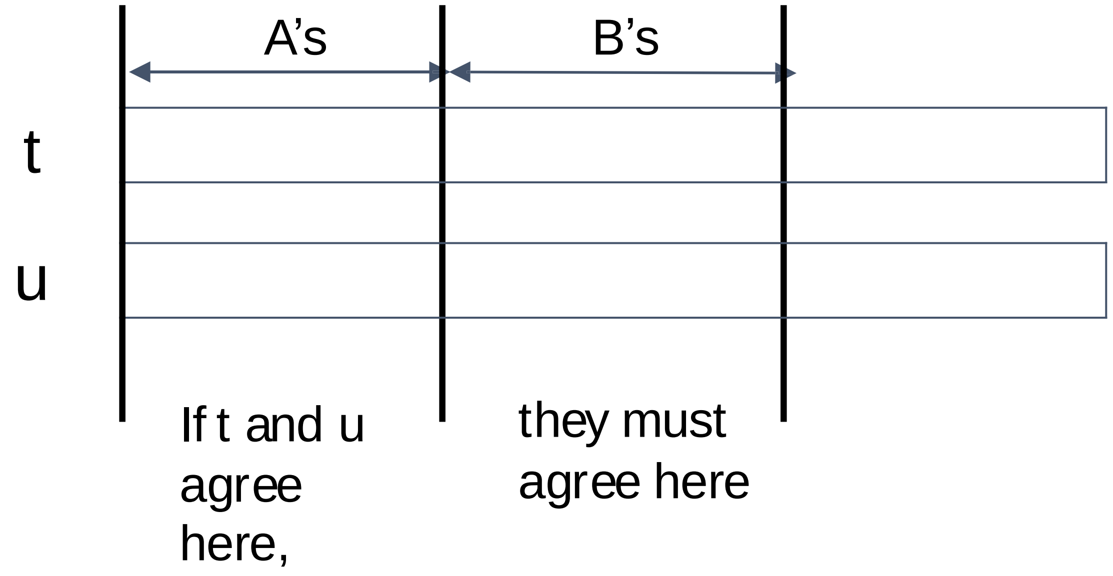
- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

Goal: develop theory to understand why this design may be better and how to find this decomposition…

Functional dependency (FD)

Definition: if two tuples of R agree on all the attributes $A_1, A_2, \ldots, A_n$, they must also agree on (or functionally determine) $B_1, B_2, \ldots, B_m$

- Denoted as $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$



A->B means that "whenever two tuples agree on A then they agree on B."

Splitting/combining rule

- Splitting/combining can be applied to the right sides of FD's

$$A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$$

Splitting rule

Combining rule

$$A_1A_2 \ldots A_n \rightarrow B_1, \quad A_1A_2 \ldots A_n \rightarrow B_2, \quad \ldots, \quad A_1A_2 \ldots A_n \rightarrow B_m$$

Splitting/combining rule

- For example,

$$\text{title, year} \rightarrow \text{length, genre, studioName}$$

$$\|$$

$$\text{title, year} \rightarrow \text{length}$$
$$\text{title, year} \rightarrow \text{genre}$$
$$\text{title, year} \rightarrow \text{studioName}$$

Splitting rule

- Splitting rule does not apply to the left sides of FD's

$$\text{title, year} \rightarrow \text{length}$$

$$\not\Downarrow$$

$$\text{title} \rightarrow \text{length}$$
$$\text{year} \rightarrow \text{length}$$

Functional dependencies as constraints

A functional dependency is a form of <u>constraint</u>

- Holds on some instances (but not others) – can check whether there are violations

- Part of the schema, helps define a valid instance

Recall: an <u>instance</u> of a schema is a multiset of tuples conforming to that schema, i.e. a table

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Note: The FD {Course} -> {Room} holds on this instance

Functional dependencies as constraints

Note that:

- You can check if an FD is violated by examining a single instance;
- However, you cannot prove that an FD is part of the schema by examining a single instance.
  - This would require checking every valid instance

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

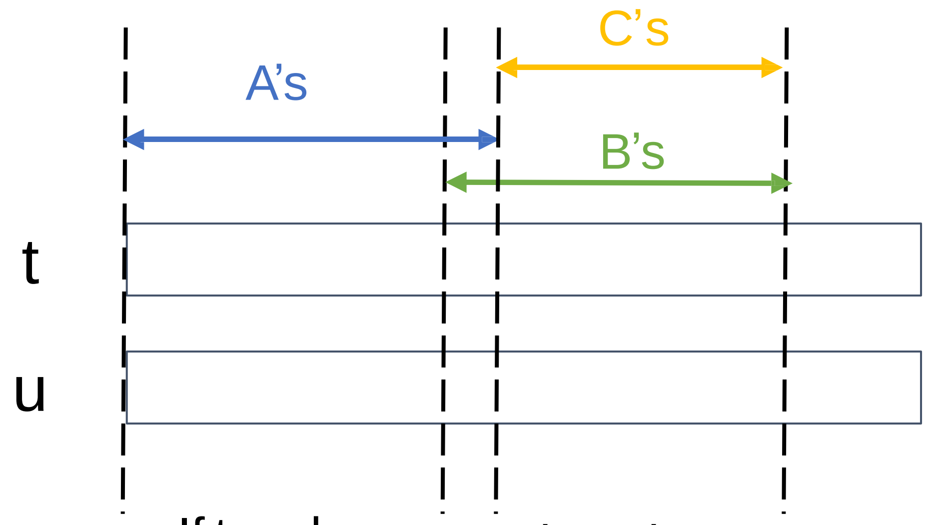
However, cannot prove that the FD {Course} -> {Room} is part of the schema

Trivial functional dependencies

A constraint is *trivial* if it holds for every possible instance of the relation.

Trivial FDs:
$A_1A_2 \ldots A_n \rightarrow B_1\ B_2 \ldots B_m$ such that
$\{B_1, B_2, \ldots B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}$

Trivial dependency rule:
$A_1A_2 \ldots A_n \rightarrow B_1\ B_2 \ldots B_m$ is equivalent to $A_1A_2 \ldots A_n \rightarrow C_1\ C_2 \ldots C_k$ where the C's are the B's that are not also A's



If t and u agree on the As

Then they must agree on the Bs

And surely they agree on the Cs

FDs for relational schema design

High-level idea: why do we care about FDs?

1. Start with some relational schema
2. Find out its functional dependencies (FDs)
3. Use these to design a better schema
    1. One which minimizes possibility of anomalies

This part can be tricky!

Finding functional dependencies

There can be a large number of FDs…

Let's start with this problem:

Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

Three simple rules called <u>Armstrong's Rules</u>.

1. Reflexivity,
2. Augmentation,
3. Transitivity

Armstrong's axioms

You can derive any FDs that follows from a given set using these axioms:

1. Reflexivity:
   If Y is a subset of X, then X → Y

This means that a set of attributes always determines a subset of itself

2. Augmentation:
   If X → Y, then XZ → YZ for any Z

This means we can add the same attributes to both sides of a functional dependency.

3. Transitivity:
   If X → Y and Y → Z, then X → Z

This allows us to chain functional dependencies.

Armstrong's axioms

- Does $AB \rightarrow D$ follow from the FDs below?

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

1. $AB \rightarrow C$ (given)
2. $BC \rightarrow AD$ (given)

Armstrong's axioms

- Does $AB \longrightarrow D$ follow from the FDs below?

$AB \longrightarrow C$

$BC \longrightarrow AD$

$D \longrightarrow E$

$CF \longrightarrow B$

1. $AB \longrightarrow C$ (given)
2. $BC \longrightarrow AD$ (given)
3. $AB \longrightarrow BC$ (Augmentation on 1)

Armstrong's axioms

- Does $AB \rightarrow D$ follow from the FDs below?

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

1. $AB \rightarrow C$ (given)
2. $BC \rightarrow AD$ (given)
3. $AB \rightarrow BC$ (Augmentation on 1)
4. $AB \rightarrow AD$ (Transitivity on 2,3)

Armstrong's axioms

- Does $AB \rightarrow D$ follow from the FDs below?

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

1. $AB \rightarrow C$ (given)
2. $BC \rightarrow AD$ (given)
3. $AB \rightarrow BC$ (Augmentation on 1)
4. $AB \rightarrow AD$ (Transitivity on 2,3)
5. $AD \rightarrow D$ (Reflexivity)

Armstrong's axioms

- Does $AB \rightarrow D$ follow from the FDs below?

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

1. $AB \rightarrow C$ (given)
2. $BC \rightarrow AD$ (given)
3. $AB \rightarrow BC$ (Augmentation on 1)
4. $AB \rightarrow AD$ (Transitivity on 2,3)
5. $AD \rightarrow D$ (Reflexivity)
6. $AB \rightarrow D$ (Transitivity on 4,5)

Can we find an algorithmic way to do this?

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F,
the <u>closure</u>, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where
$\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

$\{A, B\}+$

A, B

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F, the closure, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where $\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

{A, B}+

A, B, C

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F,
the closure, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where
$\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

{A, B}+

A, B, C, D

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F,
the <u>closure</u>, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where
$\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

{A, B}+

A, B, C, D, E

Closure of attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F,
the <u>closure</u>, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B where
$\{A_1, \ldots, A_n\} \rightarrow B$ follows from the FDs in F

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

{A, B}+

A, B, C, D, E

Cannot be expanded further, so this is a closure

Closure algorithm

Start with $X = \{A_1, \ldots, A_n\}$ and set of FDs F.

Repeat until X doesn't change; do:

if $\{B_1, \ldots, B_n\} \longrightarrow C$ is entailed by F

and $\{B_1, \ldots, B_n\} \subseteq X$

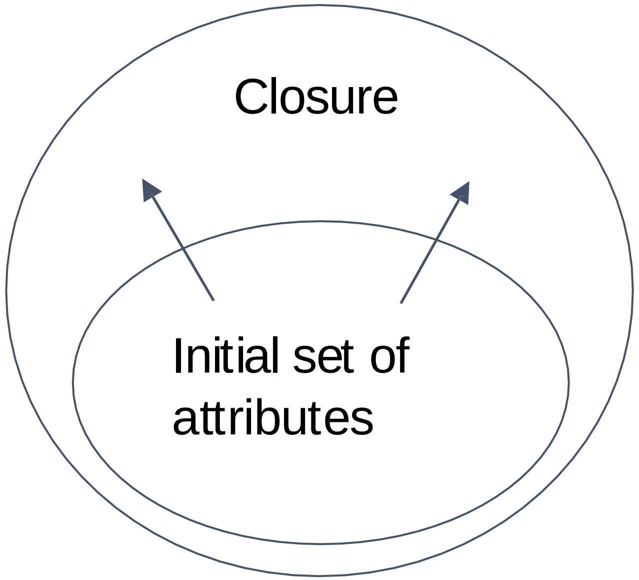
then add C to X.

Return X as X^+

Helps to split the FD's of F so each FD has a single attribute on the right

The algorithm

- only produces true FDs
- Discovers all true FDs



Why do we need the closure?

With closure we can find all FD's easily

To check if $X \rightarrow A$

1. Compute X^+
2. Check if $A \in X^+$

Note here that X is a set of attributes, but A is a single attribute. Why does considering FDs of this form suffice?

Recall the <u>split/combine</u> rule:
$X \rightarrow A_1, \ldots, X \rightarrow A_n$
implies
$X \rightarrow \{A_1, \ldots, A_n\}$

Using closure to infer ALL FDs

<u>Example:</u>

Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

$\{A\}^+ = \{A\}$
$\{B\}^+ = \{B,D\}$
$\{C\}^+ = \{C\}$
$\{D\}^+ = \{D\}$
$\{A,B\}^+ = \{A,B,C,D\}$
$\{A,C\}^+ = \{A,C\}$
$\{A,D\}^+ = \{A,B,C,D\}$
$\{A,B,C\}^+ = \{A,B,D\}^+ = \{A,C,D\}^+ = \{A,B,C,D\}$ $\{B,C,D\}^+ = \{B,C,D\}$
$\{A,B,C,D\}^+ = \{A,B,C,D\}$

Using closure to infer ALL FDs

Example:

Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

$\{A\}^+ = \{A\}$, $\{B\}^+ = \{B,D\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$, $\{A,B\}^+ = \{A,B,C,D\}$, $\{A,C\}^+ = \{A,C\}$, $\{A,D\}^+ = \{A,B,C,D\}$, $\{A,B,C\}^+ = \{A,B,D\}^+ = \{A,C,D\}^+ = \{A,B,C,D\}$, $\{B,C,D\}^+ = \{B,C,D\}$, $\{A,B,C,D\}^+ = \{A,B,C,D\}$

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

Using closure to infer ALL FDs

Example:

Given F =
$\{A,B\} \to C$
$\{A,D\} \to B$
$\{B\} \to D$

Step 1: Compute X^+, for every set of attributes X:

$\{A\}^+ = \{A\}$, $\{B\}^+ = \{B,D\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$, $\{A,B\}^+ = \{A,B,C,D\}$,
$\{A,C\}^+ = \{A,C\}$, $\{A,D\}^+ = \{A,B,C,D\}$, $\{A,B,C\}^+ = \{A,B,D\}^+ = \{A,C,D\}^+ =$
$\{A,B,C,D\}$, $\{B,C,D\}^+ = \{B,C,D\}$, $\{A,B,C,D\}^+ = \{A,B,C,D\}$

Y is in the closure of X

Step 2: Enumerate all FDs $X \to Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$\{A,B\} \to \{C,D\}$, $\{A,D\} \to \{B,C\}$,
$\{A,B,C\} \to \{D\}$, $\{A,B,D\} \to \{C\}$,
$\{A,C,D\} \to \{B\}$

The FD $X \to Y$ is non-trivial

Keys and Superkeys

A superkey is a set of attributes $A_1, \ldots, A_n$ s.t.
for any other attribute B in R,
we have $\{A_1, \ldots, A_n\} \rightarrow B$

i.e. all attributes are functionally determined by a superkey

A key is a minimal superkey

This means that no subset of a key is also a superkey
(i.e., dropping any attribute from the key makes it no longer a superkey)

Keys and Superkeys

Q: What are superkeys and keys in the following relation?

{title, year, length, starName} is a superkey

{title, year, starName} is a key
{title, year} is not a key because title year → starName is not an FD
{year, starName} is not a key because year starName → title is not an FD
{title, starName} is not a key because title starName → year is not an FD

title	**year**	**length**	**genre**	**studioName**	**starName**
Ponyo	2008	103	anime	Ghibli	Yuria Nara
Ponyo	2008	103	anime	Ghibli	Hiroki Doi
Oldboy	2003	120	mystery	Show East	Choi Min-Sik

Keys and Superkeys

For each set of attributes X

1. Compute X^+
2. If X^+ = set of all attributes then X is a superkey
3. If X is minimal, then it is a key

Example

Product(name, price, category, color)

{name, category} $\longrightarrow$ price
{category} $\longrightarrow$ color

What is a key?

Example

Product(name, price, category, color)

{name, category} $\longrightarrow$ price
{category} $\longrightarrow$ color

$\{\text{name, category}\}^+$ = {name, price, category, color}
= the set of all attributes
$\Longrightarrow$ this is a **superkey**
$\Longrightarrow$ this is a **key**, since neither name nor category alone is a superkey

Today's agenda

- Refresher
  - SQL
  - Entity-Relationship model
- Design theory
  - Normal forms & functional dependencies
  - Boyce-Codd normal form
  - 3NF

Back to conceptual design

Now that we know how to find FDs, it's a straight-forward process:

1. Search for "bad" FDs

2. If there are any, then *keep decomposing the table into sub-tables* until no more bad FDs

3. When done, the database schema is *normalized*

Recall: there are several normal forms…

Boyce-Codd normal form (BCNF)

Main idea is that we define "good" and "bad" FDs as follows:

- $X \rightarrow A$ is a "*good FD" if X is a (super)key*
  - In other words, if A is the set of all attributes
- $X \rightarrow A$ is a "*bad FD*" otherwise

We will try to eliminate the "bad" FDs!

Boyce-Codd normal form (BCNF)

Why does this definition of “good” and “bad” FDs make sense?

If X is *not* a (super)key, it functionally determines *some* of the attributes; therefore, those other attributes can be duplicated

- Recall: this means there is redundancy
- And redundancy like this can lead to data anomalies!

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Boyce-Codd normal form (BCNF)

BCNF is a simple condition for removing anomalies from relations:

A relation R is <u>in BCNF</u> if:

if $\{A_1, \ldots, A_n\} \rightarrow B$ is a non-trivial FD in R

then $\{A_1, \ldots, A_n\}$ is a superkey for R

Equivalently: $\forall$ sets of attributes X, either ($X^+ = X$) or (X^+ = all attributes)

In other words: there are no "bad" FDs

Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

$\boxed{\text{SSN} \rightarrow \text{Name,City}}$

This FD is bad because it is <u>not</u> a superkey

$\Longrightarrow$ <u>Not</u> in BCNF

What is the key?
{SSN, PhoneNumber}

Example

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Madison

SSN → Name,City

This FD is now good because it is the key

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

Now in BCNF!

Boyce-Codd normal form (BCNF)

Any two-attribute relation is in BCNF

- If there are no nontrivial FDs, BCNF holds
- If A → B holds, but not B → A, the only nontrivial FD has A (i.e., the key) on the left
- Symmetric case when B → A holds, but not A → B
- If both A → B and B → A hold, any nontrivial FD has A or B (both are keys) on the left

```
Employee(empId, ssn)
```

empId → ssn
ssn → empId

BCNF decomposition algorithm

BCNFDecomp(R):

- Find an FD $X \rightarrow Y$ that violates BCNF (X and Y are sets of attributes)
- Compute the closure X^+
- let $Y = X^+ - X$, $Z = (X^+)^C$

Let Y be the attributes that X functionally determines (+ that are not in X)

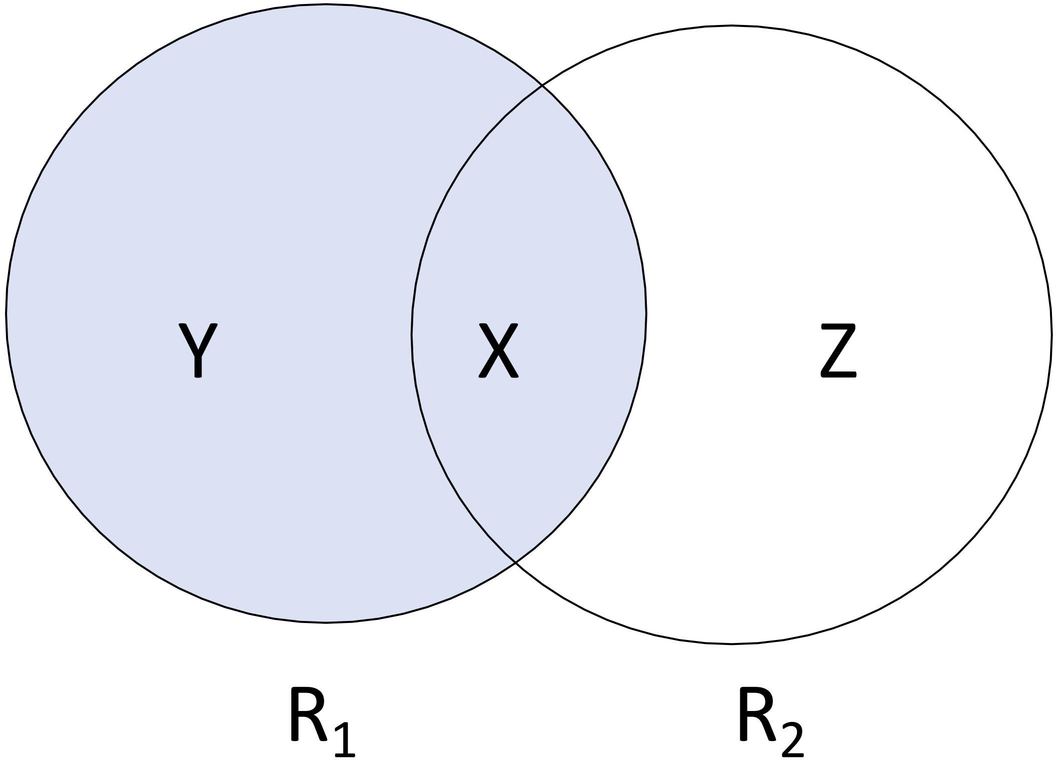
And let Z be the complement, the other attributes that it doesn't

BCNF decomposition algorithm

BCNFDecomp(R):

- Find an FD X → Y that violates BCNF (X and Y are sets of attributes)
- Compute the closure X+
- let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$

Split into one relation (table) with X plus the attributes that X determines (Y)…

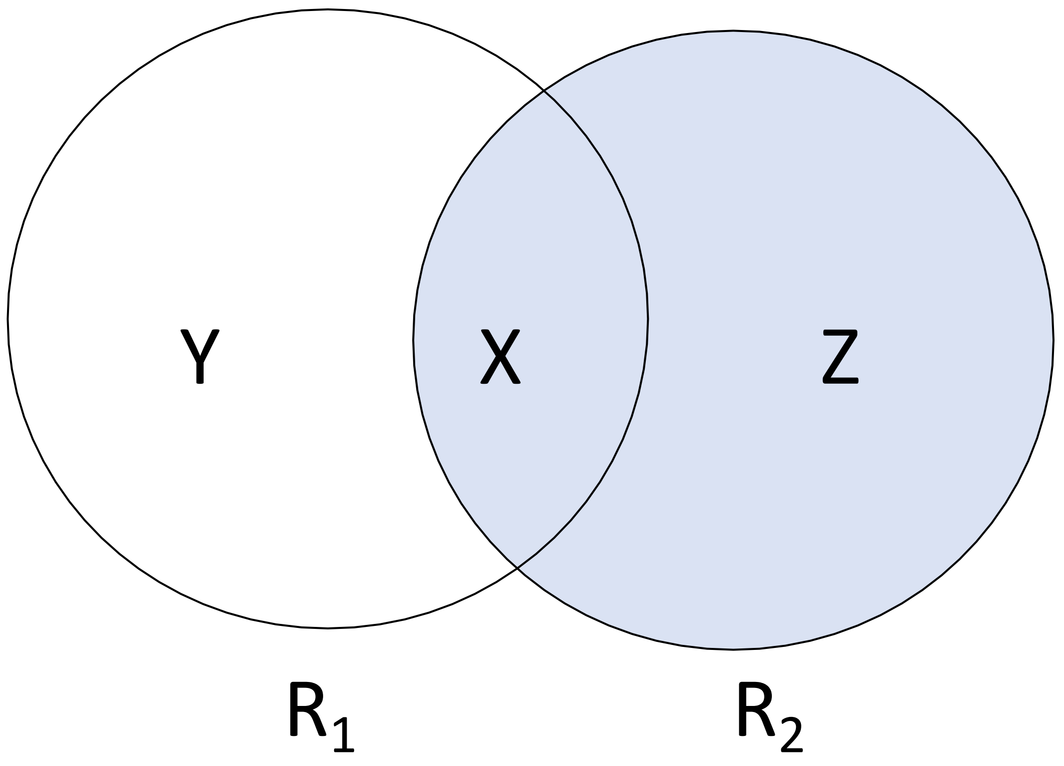


BCNF decomposition algorithm

BCNFDecomp(R):

- Find an FD $X \rightarrow Y$ that violates BCNF (X and Y are sets of attributes)
- Compute the closure X+
- let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$
- Recursively decompose R_1 and R_2

And one relation with X plus the attributes it does not determine (Z)



Example: BCNF decomposition

- In general, there can be multiple decompositions

```
R(title,year,studioName,president,presAddr)
```

R's FDs

title year → studioName
studioName → president
president → presAddr

Example: BCNF decomposition

- In general, there can be multiple decompositions

```
R(title,year,studioName,president,presAddr)
```

Key

R's FDs

BCNF violations

title year → studioName
studioName → president
president → presAddr

Example: BCNF decomposition

- In general, there can be multiple decompositions

`R(title,year,studioName,president,presAddr)`

`R1(studioName,president,presAddr)`

`R2(title,year,studioName)`

Key

R's FDs

title year → studioName

studioName → president

president → presAddr

BCNF violations

Example: BCNF decomposition

- In general, there can be multiple decompositions

`R(title,year,studioName,president,presAddr)`

`R1(studioName,president,presAddr)` `R2(title,year,studioName)`

Key

R1's FDs

studioName → president
president → presAddr

BCNF violation

Example: BCNF decomposition

- In general, there can be multiple decompositions

`R(title,year,studioName,president,presAddr)`

`R1(studioName,president,presAddr)`   `R2(title,year,studioName)`

`R3(president,presAddr)`   `R4(president,studioName)`

R1's FDs

Key

studioName → president
president → presAddr

BCNF violation

Q: Is this algorithm guaranteed to terminate successfully?

Recap: decompose to remove redundancies

1. We saw that redundancies in the data (“bad FDs”) can lead to data anomalies

2. We developed mechanisms to detect and remove redundancies by decomposing tables into BCNF

    1. BCNF decomposition is *standard practice*- very powerful & widely used!

3. However, sometimes decompositions can lead to more subtle unwanted effects…

When does this happen?

Recovering information from a decomposition

$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$ $R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

R_1 = the *projection* of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

R_2 = the *projection* of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

Recovering information from a decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Price
Gizmo	19.99
OneClick	24.99
~~Gizmo~~	~~19.99~~

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Sometimes a decomposition is "correct"

i.e. it is a <u>Lossless decomposition</u>

Recovering information from a decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

However sometimes it isn't

What's wrong here?

Lossless decompositions

$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$

$R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

A decomposition R to (R1, R2) is lossless if R = R1 Join R2

Lossless decompositions

$$\begin{tikzcd}[column sep=tiny] & R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p) \arrow[dl] \arrow[dr] & \\ R_1(A_1,\ldots,A_n,B_1,\ldots,B_m) & & R_2(A_1,\ldots,A_n,C_1,\ldots,C_p) \end{tikzcd}$$

If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$
Then the decomposition is lossless

Note: don't need
$A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p$

BCNF decomposition is always lossless. Why?

A Problem with BCNF

Unit	Company	Product
…	…	…

Unit ⟶ Company
Company,Product ⟶ Unit

Unit	Company
…	…

Unit	Product
…	…

Unit ⟶ Company

We do a BCNF decomposition on a "bad" FD:
$\{\text{Unit}\}^+ = \{\text{Unit, Company}\}$

We lose the FD Company,Product ⟶ Unit!!

Why is that a problem?

Unit	Company
Galaga99	UW
Bingo	UW

Unit	Product
Galaga99	Databases
Bingo	Databases

No problem so far. All local FD's are satisfied.

Unit $\rightarrow$ Company

Unit	Company	Product
Galaga99	UW	Databases
Bingo	UW	Databases

Let's put all the data back into a single table again:

Violates the FD Company,Product $\rightarrow$ Unit!!

The problem with BCNF

- We started with a table R and FDs F

- We decomposed R into BCNF tables R_1, R_2, …
  with their own FDs F_1, F_2, …

- We insert some tuples into each of the relations—which satisfy their local FDs but when reconstruct it violates some FD across tables!

Practical Problem: To enforce FD, must reconstruct R—on each insert!

Desirable properties of decomposition

In general, we want the decomposition to have the following properties

(1) Elimination of anomalies

(2) Recoverability of information: can we recover the original relation by joining?

(3) Preservation of dependencies: if we check the projected FD's in the decomposed relations, does the reconstructed original relation satisfy the original FD's?

- BCNF gives (1) and (2), but not necessarily (3)
- 3NF gives (2) and (3), but not necessarily (1)
- In fact, there is no way to get all three at once!

Third normal form (3NF)

A relation R is <u>in 3NF</u> if:

For every non-trivial FD $A_1, \ldots, A_n \longrightarrow B$, either

- $\{A_1, \ldots, A_n\}$ is a superkey for R
- B is a prime attribute (i.e., B is part of some candidate key of R)

Example:

- The keys are AB and AC
- $B \rightarrow C$ is a BCNF violation, but not a 3NF violation because C is prime (part of the key AC)

R(A,B,C)

$AC \longrightarrow B$
$B \longrightarrow C$

3NF decomposition algorithm

3NFDecomp(R, F):

- Find minimal basis for F, say G
- For each FD $X \to A$ in G, if there is no relation that contains XA, create a new relation (X, A)
- Eliminate any relation that is a proper subset of another relation.
- If none of the resulting schemas are superkeys, add one more relation whose schema is a key for R

Minimal basis:

$AB \to C$
$C \to B$
$A \to D$

Keys:

ABE,ACE

R(A,B,C,D,E)

R_1(A,B,C)

R_3(A,D)

R_4(A,B,E)

Minimal basis generation

Given a set of FD's F, any set of FD's equivalent to F is a basis for F

Input: $F = \{A \to AB, AB \to C\}$

1. Split FD's so that they have singleton right sides
   $G = \{A \to B, A \to A, AB \to C\}$
2. Remove trivial FDs
   $G = \{A \to B, AB \to C\}$
3. Minimize the left sides of each FD
   $G = \{A \to B, A \to C\}$
4. Remove redundant FDs
   $G = \{A \to B, A \to C\}$

For each FD $X \to A$ in F:
For each attribute B in X:
If $(X - \{B\})^+$ contains A,
remove B from X.

BCNF vs 3NF

- Given a non-trivial FD X → B (X is a set of attributes)
  - BCNF: X must be a superkey
  - 3NF: X must be a superkey or B is prime
- Use 3NF over BCNF if you need dependency preservation
- However, 3NF may not remove all redundancies and anomalies

3NF

BCNF

3NF relation:

A	B	C
1	2	3
3	2	3
2	3	1

F: B → C, AC → B

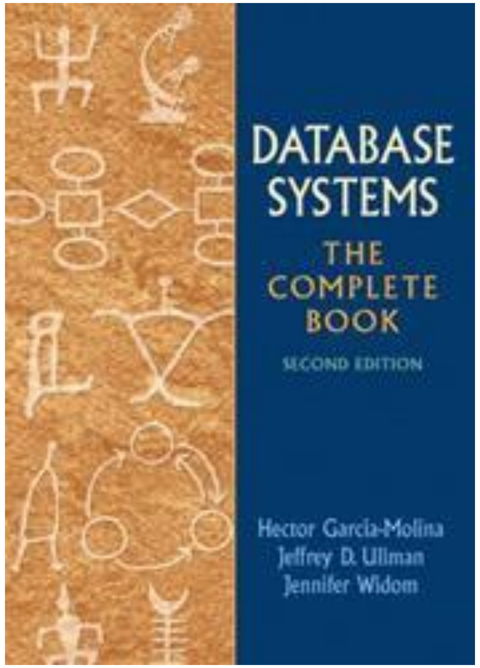
Can have redundancy and update anomalies

Can have deletion anomalies

Further readings

4NF: Remove Multi-value dependency redundancies

Property	3NF	BCNF	4NF
Lossless join	Yes	Yes	Yes
Eliminates FD redundancies	No	Yes	Yes
Eliminates MVD redundancies	No	No	Yes
Preserves FD's	Yes	No	No
Preserves MVD's	No	No	No



3NF

BCNF

4NF

Summary

Good schema design is important

- Avoid redundancy and anomalies
- Functional dependencies

Normal forms describe how to remove this redundancy by decomposing relations

- BCNF gives elimination of anomalies and lossless join
- 3NF gives lossless join and dependency preservation

BCNF is intuitive and most widely used in practice