# CSEP544
# Data Management

## Logical Database Design

# Quick Recap

- The Relational Data Model
- SQL

# Data Independence Principle

- Users see a *logical* view of the data

- System does *physical* implementation

# Example 1

Supplier

| sno | sname | scity | sstate |
|---|---|---|---|
| 11 | ACME | Seattle | WA |
| 12 | Walmart | Portland | OR |
| 13 | Walmart | Seattle | WA |

How would you store this in file(s)?

# Example 1

Supplier

| sno | sname | scity | sstate |
|---|---|---|---|
| 11 | ACME | Seattle | WA |
| 12 | Walmart | Portland | OR |
| 13 | Walmart | Seattle | WA |

How would you store this in file(s)?

Option 1: Row-major order (e.g. CSV file)

Supplier.csv

```
11|ACME|Seattle|WA\n12|Walmart|Portland|OR\n13…
```

# Example 1

Supplier

| sno | sname | scity | sstate |
|---|---|---|---|
| 11 | ACME | Seattle | WA |
| 12 | Walmart | Portland | OR |
| 13 | Walmart | Seattle | WA |

How would you store this in file(s)?

Option 2: Column-major order

`Supplier-sno.csv`

```
11|12|13|…
```

`Supplier-sname.csv`

```
ACME|Walmart|Walmart|…
```

…

# Example 1

Supplier

| sno | sname | scity | sstate |
|---|---|---|---|
| 11 | ACME | Seattle | WA |
| 12 | Walmart | Portland | OR |
| 13 | Walmart | Seattle | WA |

How would you store this in file(s)?

Option 3: Clustered by scity

```
Seattle,WA,(11|ACME;13|Walmart)\nPortland,OR,(…)…
```

…

# Example 2

```
SELECT *
FROM    Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = 'WA'
        and z.pcolor = 'red';
```

How would you evaluate this query?

Supplier

| sno | sname | sstate | … |
|---|---|---|---|
| | | | |
| | | | |
| | | WA | |
| | | WA | |
| | | | |
| | | | |
| | | WA | |

Supply

| sno | sname |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Part

| pno | pname | color |
|---|---|---|
| | | |
| | | red |
| | | |
| | | |
| | | |
| | | red |
| | | |

# Example 2

```
SELECT *
FROM    Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = 'WA'
        and z.pcolor = 'red';
```

How would you evaluate this query?

Supplier

| sno | sname | sstate | … |
|---|---|---|---|
| | | | |
| | | | |
| | | WA | |
| | | WA | |
| | | | |
| | | | |
| | | WA | |

Supply

| sno | pno |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Part

| pno | pname | color |
|---|---|---|
| | | |
| | | red |
| | | |
| | | |
| | | |
| | | red |
| | | |

**Option 1:**

for all Supplier
  if sstate= WA
    find all its Parts
      if color=red
        return *

# Example 2

```
SELECT *
FROM    Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = 'WA'
        and z.pcolor = 'red';
```

How would you evaluate this query?

Traverse Supply 3 times

Supplier

| sno | sname | sstate | … |
|---|---|---|---|
| | | | |
| | | | |
| | | WA | |
| | | WA | |
| | | | |
| | | | |
| | | WA | |

Supply

| sno | pno |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Part

| pno | pname | color |
|---|---|---|
| | | |
| | | red |
| | | |
| | | |
| | | |
| | | red |
| | | |

**Option 1:**

for all Supplier
  if sstate= WA
    find all its Parts
      if color=red
        return *

# Example 2

```
SELECT *
FROM    Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = 'WA'
        and z.pcolor = 'red';
```

How would you evaluate this query?

Supplier

| sno | sname | sstate | … |
|---|---|---|---|
| | | | |
| | | | |
| | | WA | |
| | | WA | |
| | | | |
| | | | |
| | | WA | |

Supply

| sno | pno |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Part

| pno | pname | color |
|---|---|---|
| | | |
| | | red |
| | | |
| | | |
| | | |
| | | red |
| | | |

**Option 2:**

```
for all Part
  if color = red
    find all its Supplier
      if sstate=WA
        return *
```

# Example 2

```
SELECT *
FROM    Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = 'WA'
        and z.pcolor = 'red';
```

How would you evaluate this query?

Traverse Supply 2 times

Supplier

| sno | sname | sstate | … |
|---|---|---|---|
| | | | |
| | | | |
| | | WA | |
| | | WA | |
| | | | |
| | | | |
| | | WA | |

Supply

| sno | pno |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Part

| pno | pname | color |
|---|---|---|
| | | |
| | | red |
| | | |
| | | |
| | | |
| | | red |
| | | |

**Option 2:**

for all Part
  if color = red
    find all its Supplier
      if sstate=WA
        return *

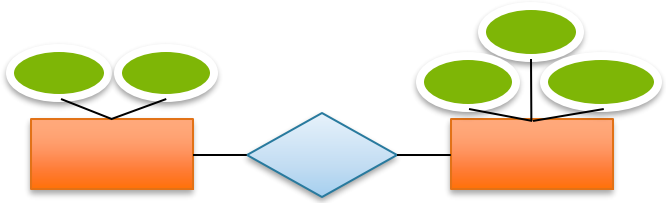
# Data Independence Principle

- Users see a *logical* view of the data

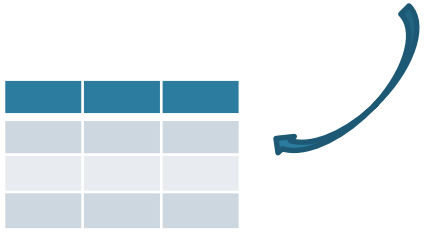
- System does *physical* implementation

# Logical Database Design

Conceptual Model

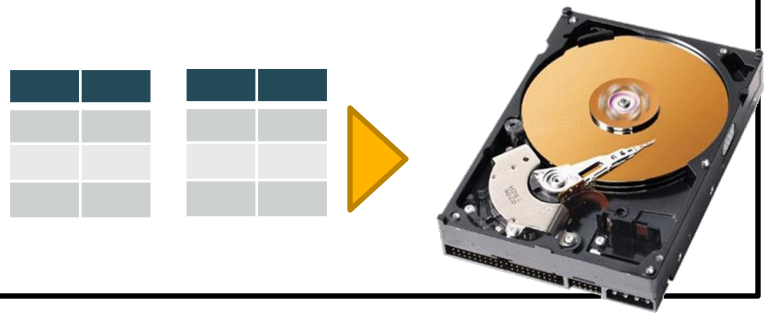


Relational Model
+ Schema
+ Constraints

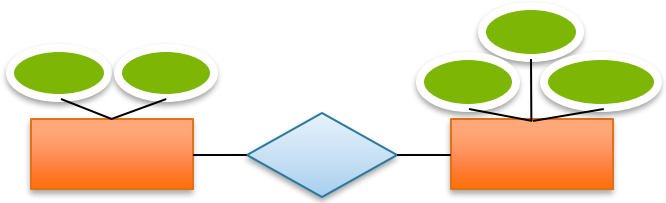


Conceptual Schema
+ Normalization

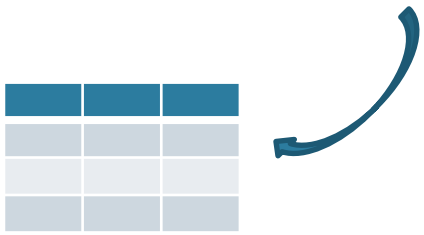
Physical Schema
+ Partitioning
+ Indexing



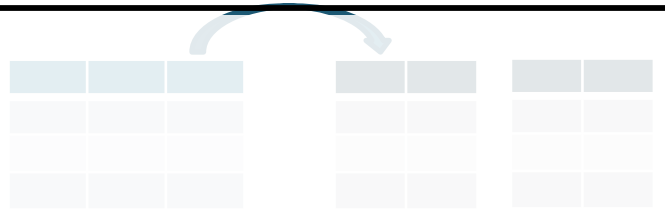
Conceptual Model



Relational Model
+ Schema
+ Constraints



Conceptual Schema
+ Normalization



Physical Schema
+ Partitioning
+ Indexing

# Conceptual Model

- A high-level description of the schema

- Usually done with Entity-Relationship diagrams (E/R)

# E/R Diagrams

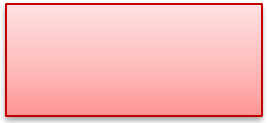
Design a schema for a database of doctors and their patients

# E/R Diagrams

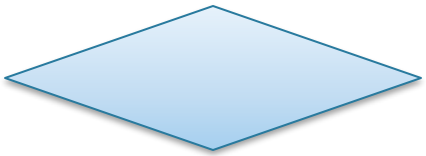
Design a schema for a database of doctors and their patients

Attributes

Entity sets



Relationship sets



# E/R Diagrams

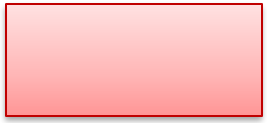
Design a schema for a database of doctors and their patients

Patient

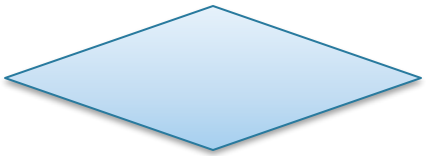
Doctor

Attributes

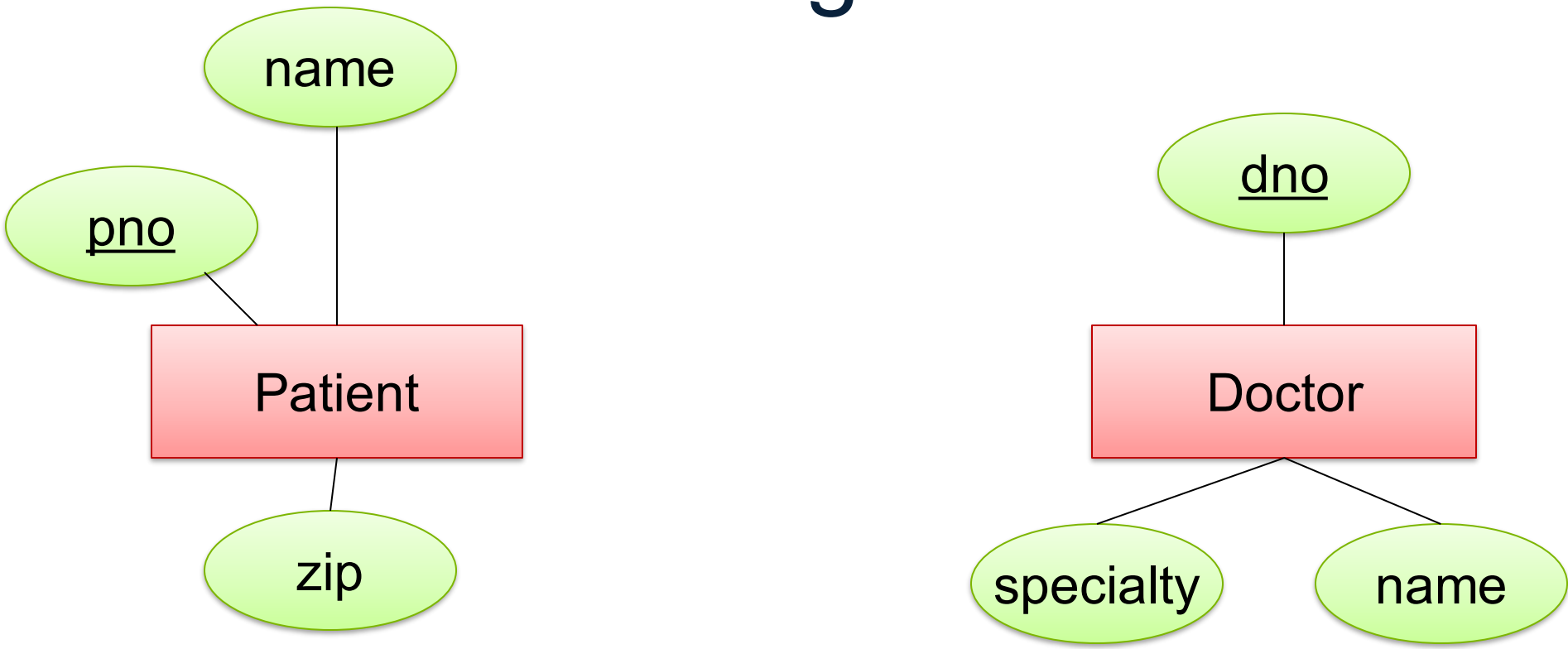
Entity sets



Relationship sets



# E/R Diagrams

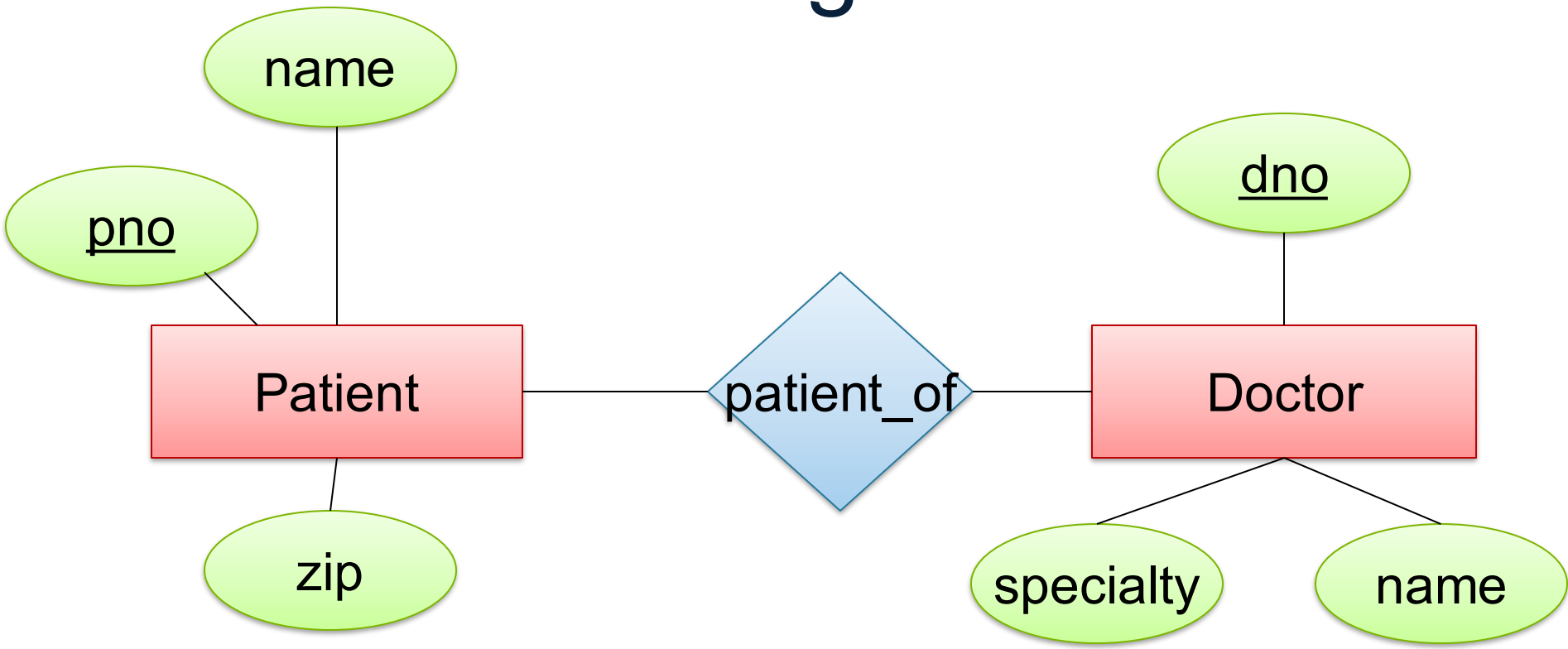



Attributes

Entity sets

Relationship sets

# E/R Diagrams

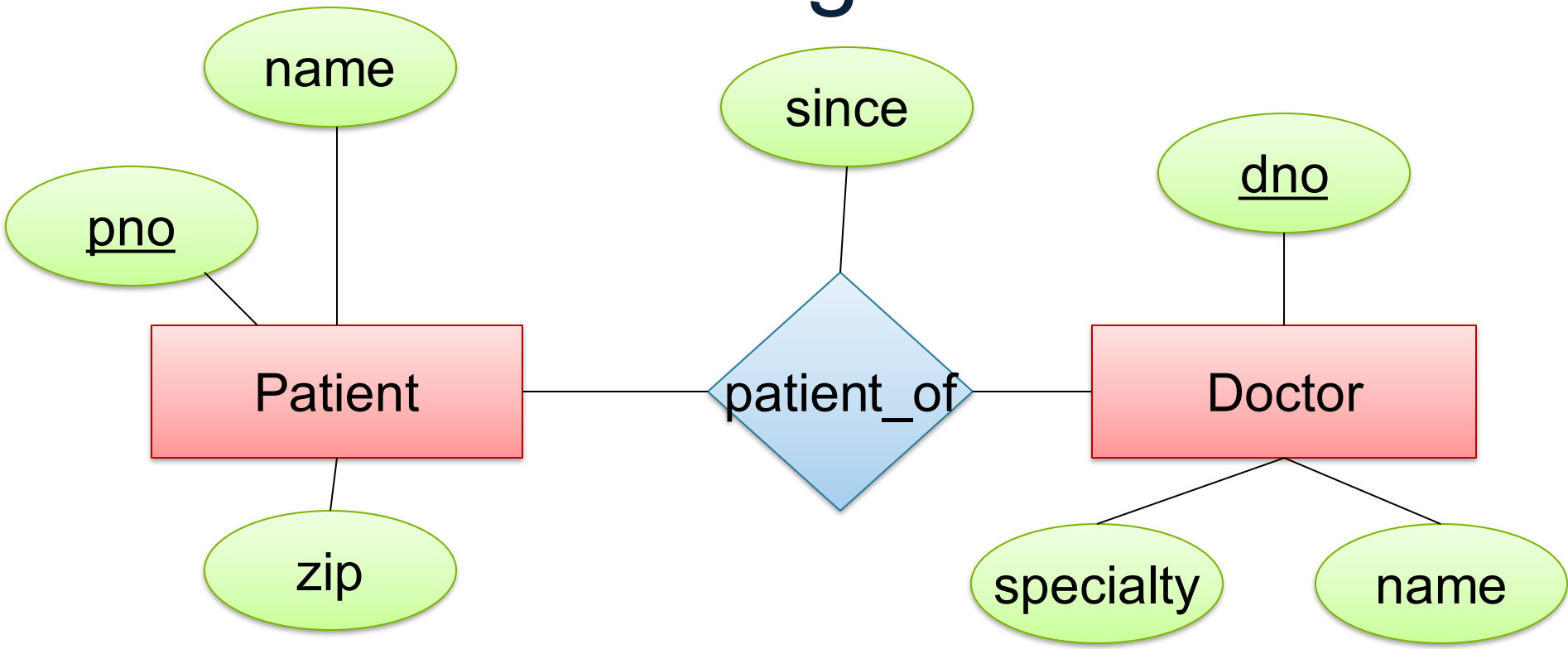



Attributes

Entity sets

Relationship sets

# E/R Diagrams

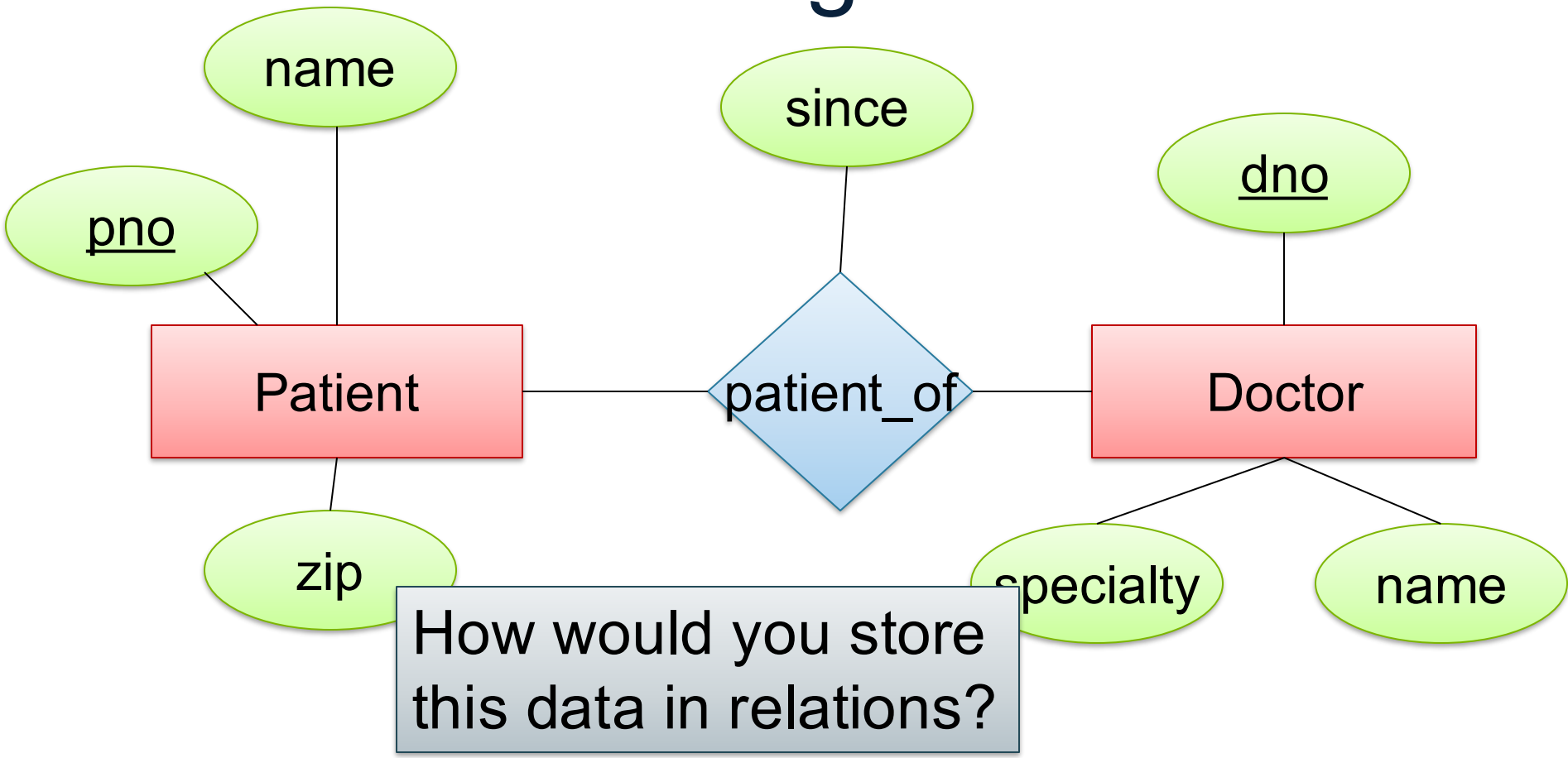



Attributes

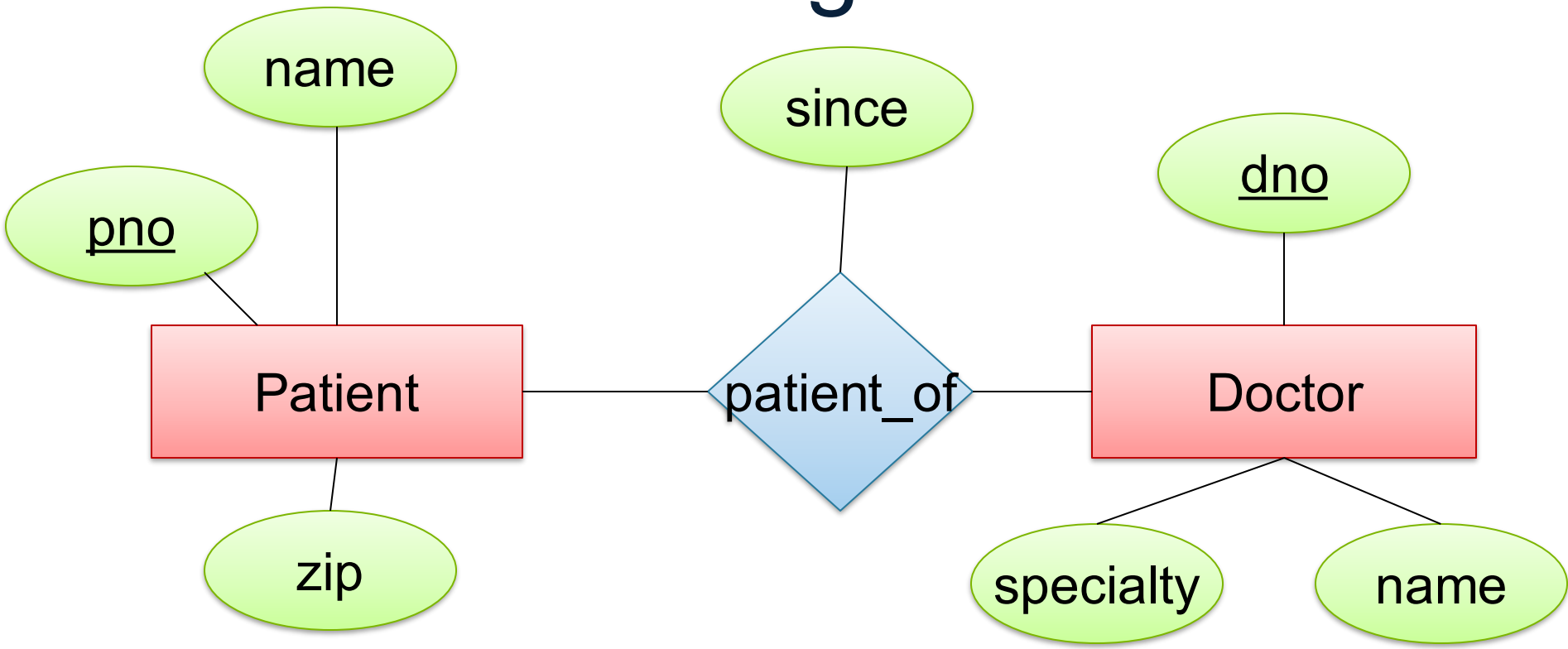
Entity sets

Relationship sets

# E/R Diagrams


name
since
dno
pno
Patient
patient_of
Doctor
zip
specialty
name
How would you store this data in relations?

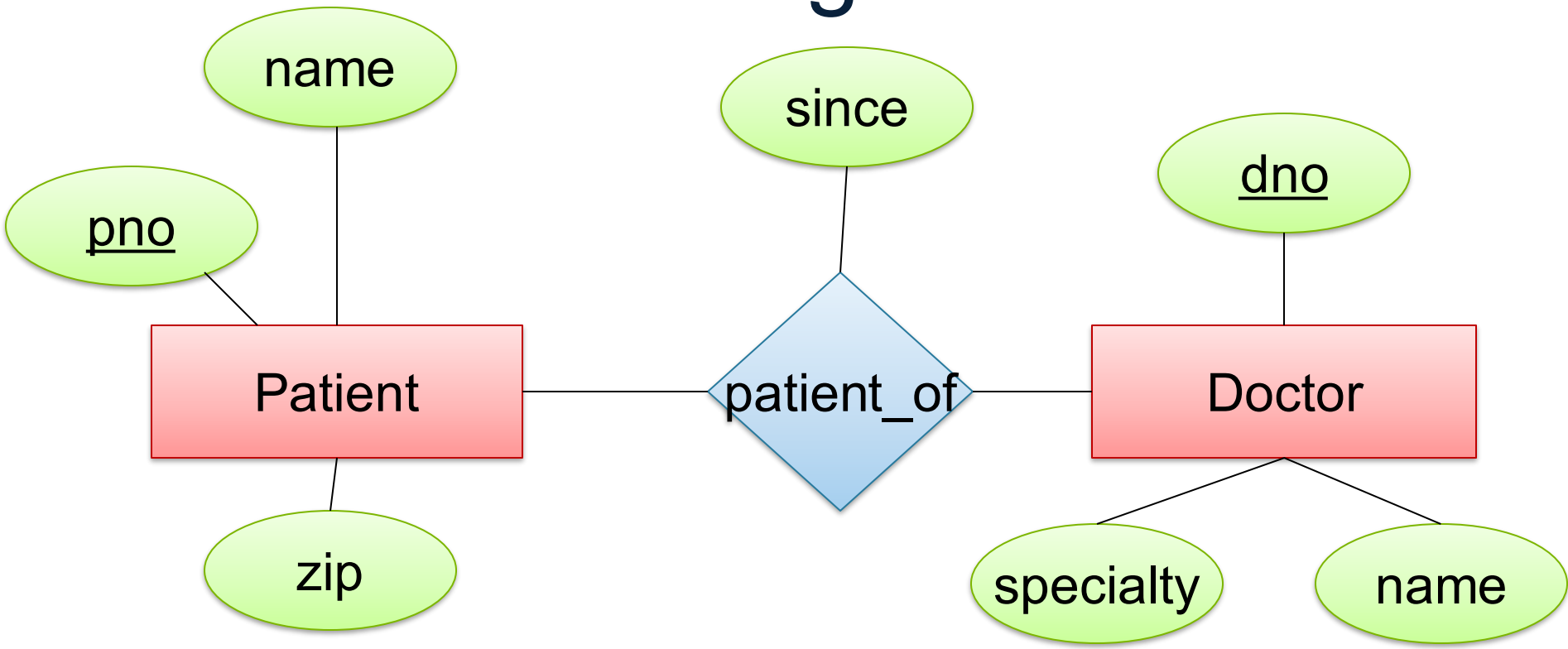
# E/R Diagrams




Patient

| pno | name | zip |
|---|---|---|
| P311 | Alice | 98765 |
| … | | |

# E/R Diagrams



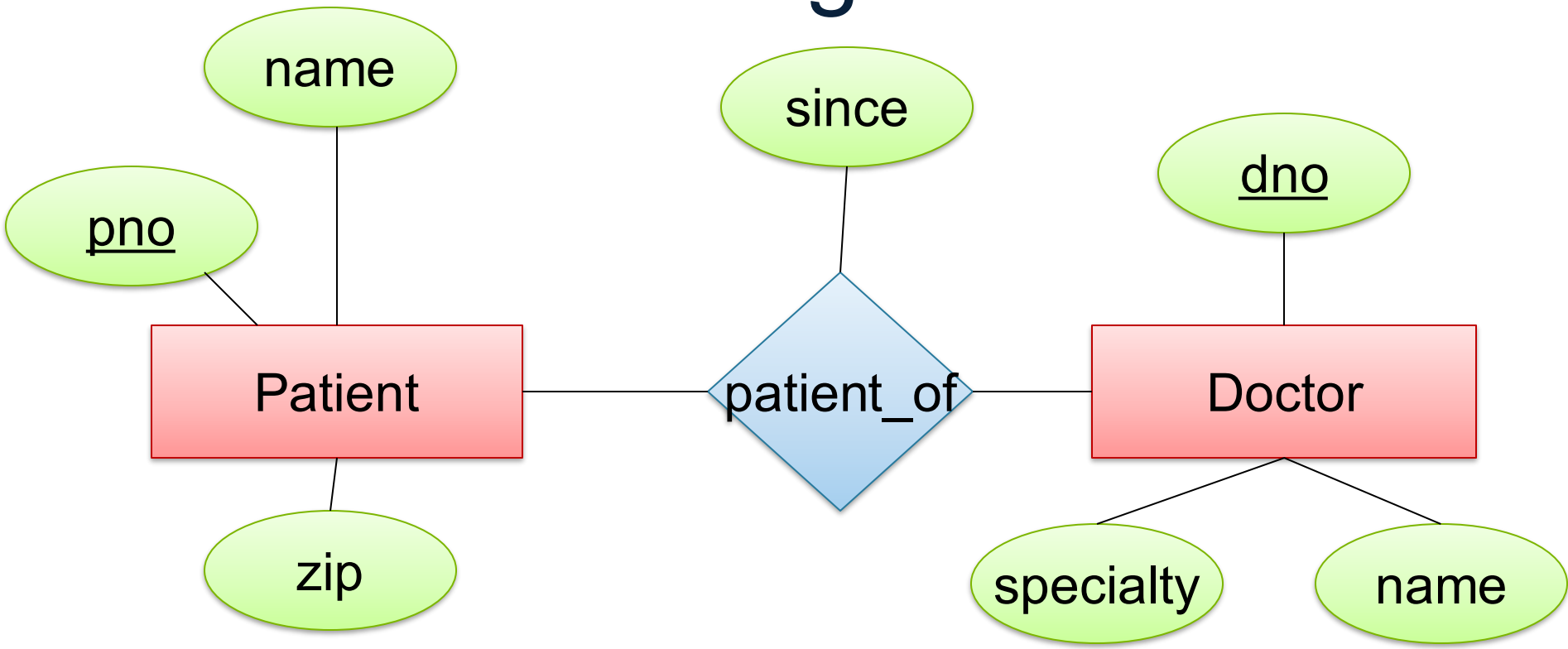

Patient

| pno | name | zip |
|---|---|---|
| P311 | Alice | 98765 |
| … | | |

Doctor

| dno | name | spec |
|---|---|---|
| D007 | Bob | cardio |
| … | | |

# E/R Diagrams




Patient

| pno | name | zip |
|---|---|---|
| P311 | Alice | 98765 |
| … | | |

Patient_of

| pno | dno | since |
|---|---|---|
| P311 | D007 | 2001 |
| … | | |

Doctor

| dno | name | spec |
|---|---|---|
| D007 | Bob | cardio |
| … | | |

# Summary so far

- Old terminology:
  entity=object, entity set=class
- An entity set → a relation
- A relationship → a relation
  (but there are exceptions: next)

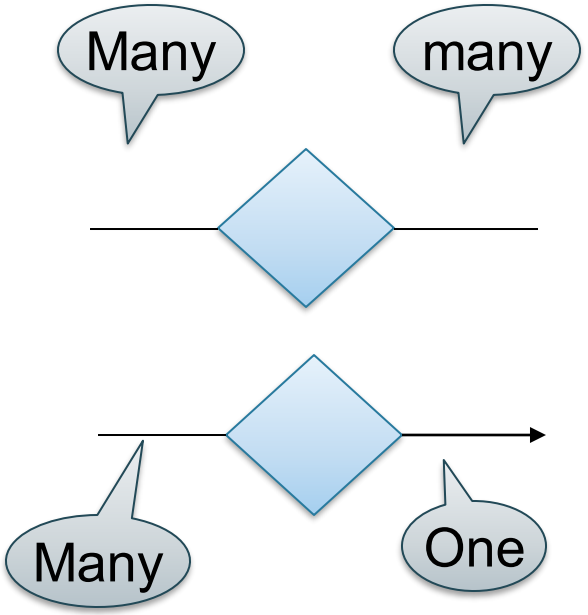
# Relationships

# Types of Relationships

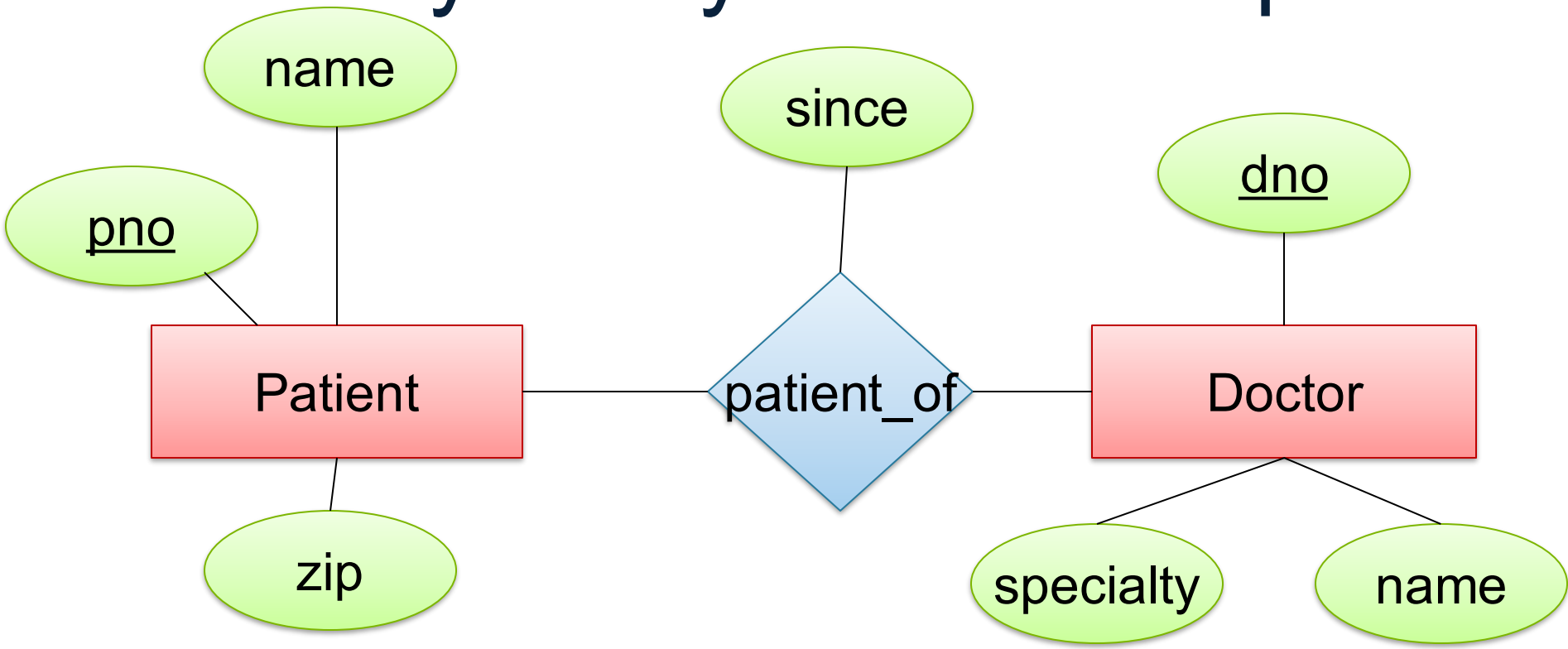
Many-to-many

Many-to-one

One-to-one



# Many-many Relationship




Patient

| pno | name | zip |
|---|---|---|
| P311 | Alice | 98765 |
| … | | |

Patient_of

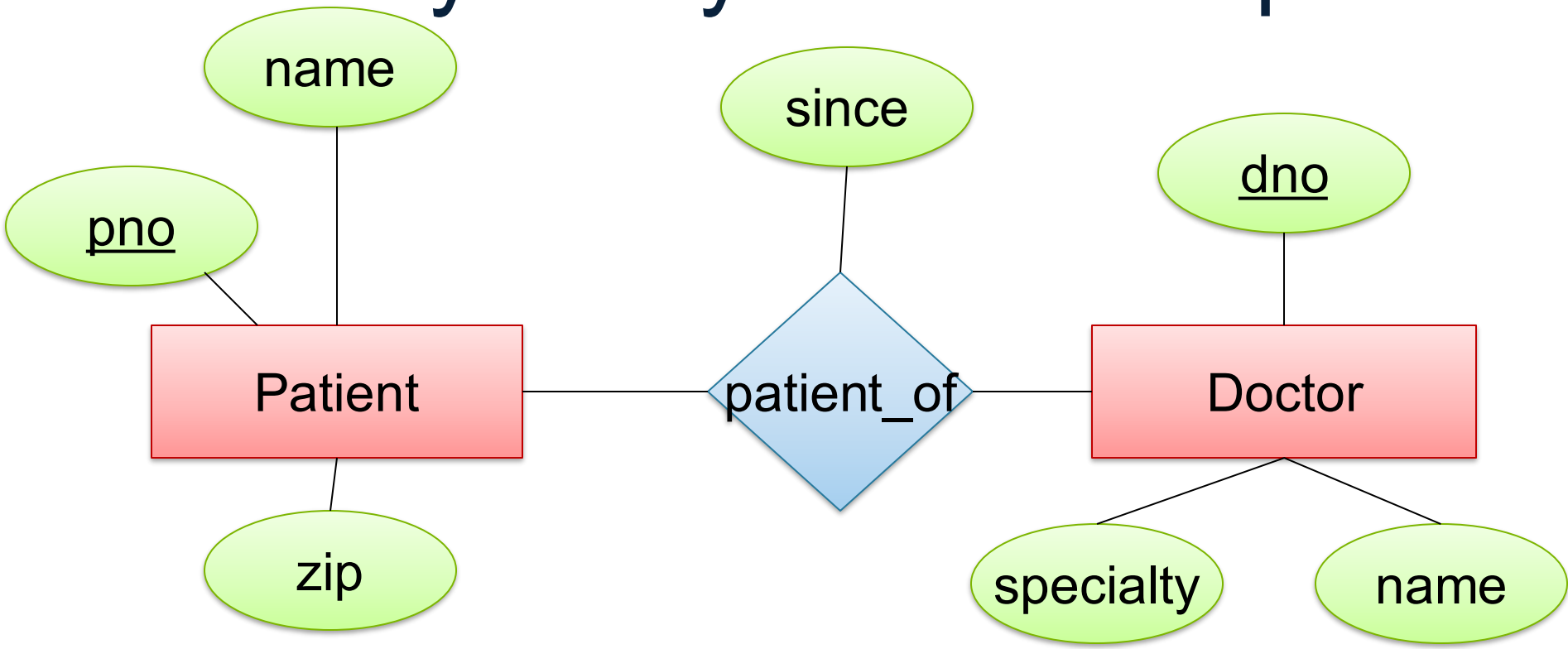
| pno | dno | since |
|---|---|---|
| P311 | D007 | 2001 |
| … | | |

Doctor

| dno | name | spec |
|---|---|---|
| D007 | Bob | cardio |
| … | | |

# Many-many Relationship

Every patient sees only one doctor




Patient

| pno | name | zip |
|---|---|---|
| P311 | Alice | 98765 |
| … | | |

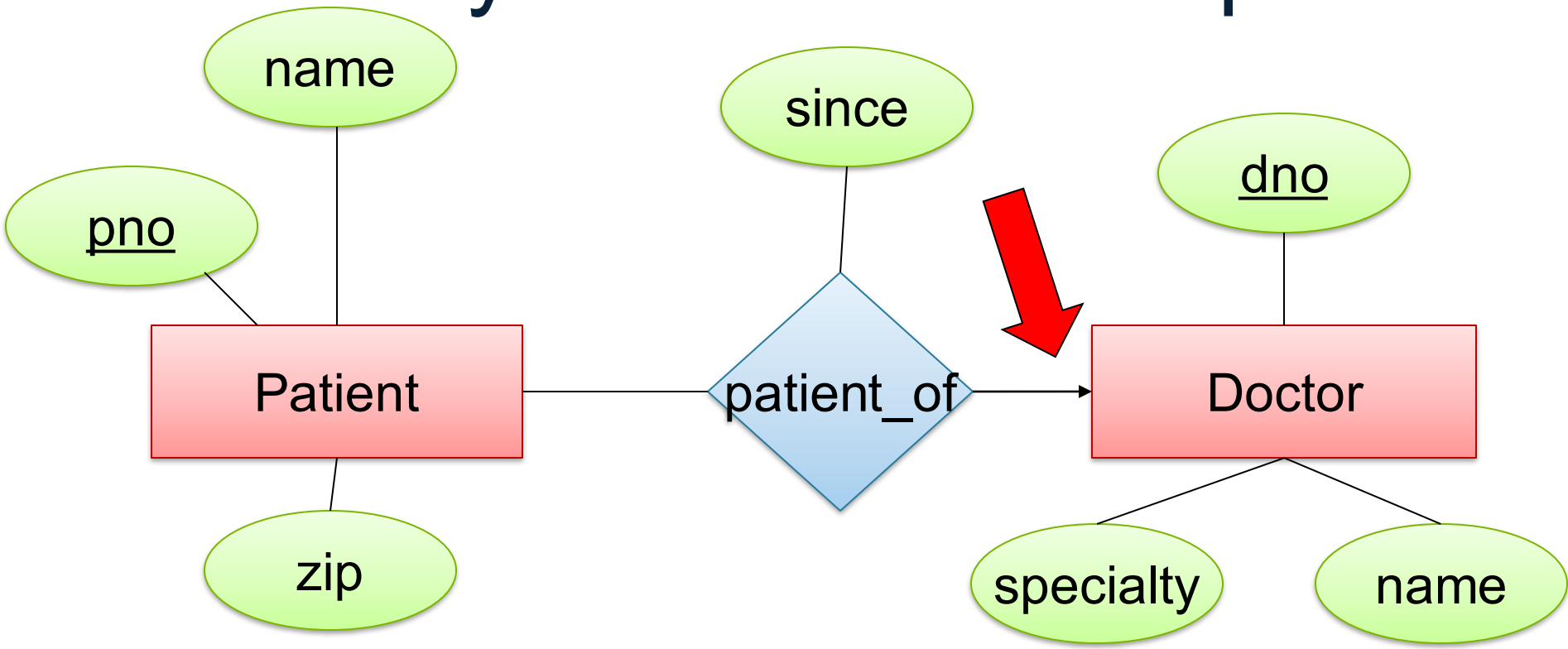
Patient_of

| pno | dno | since |
|---|---|---|
| P311 | D007 | 2001 |
| … | | |

Doctor

| dno | name | spec |
|---|---|---|
| D007 | Bob | cardio |
| … | | |

# Many-one Relationship




Patient

| pno | name | zip |
|---|---|---|
| P311 | Alice | 98765 |
| … | | |

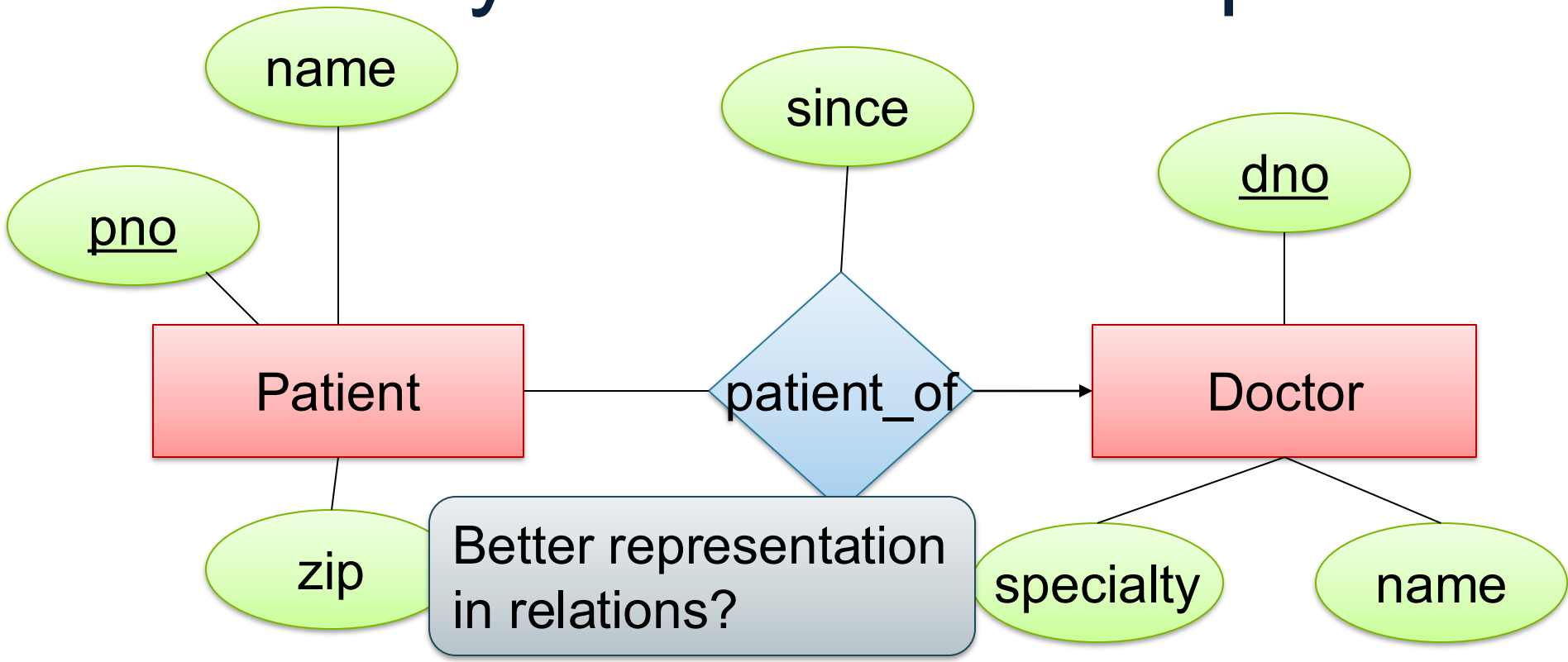
Patient_of

| pno | dno | since |
|---|---|---|
| P311 | D007 | 2001 |
| … | | |

Doctor

| dno | name | spec |
|---|---|---|
| D007 | Bob | cardio |
| … | | |

# Many-one Relationship




Patient

| pno | name | zip |
|---|---|---|
| P311 | Alice | 98765 |
| … | | |

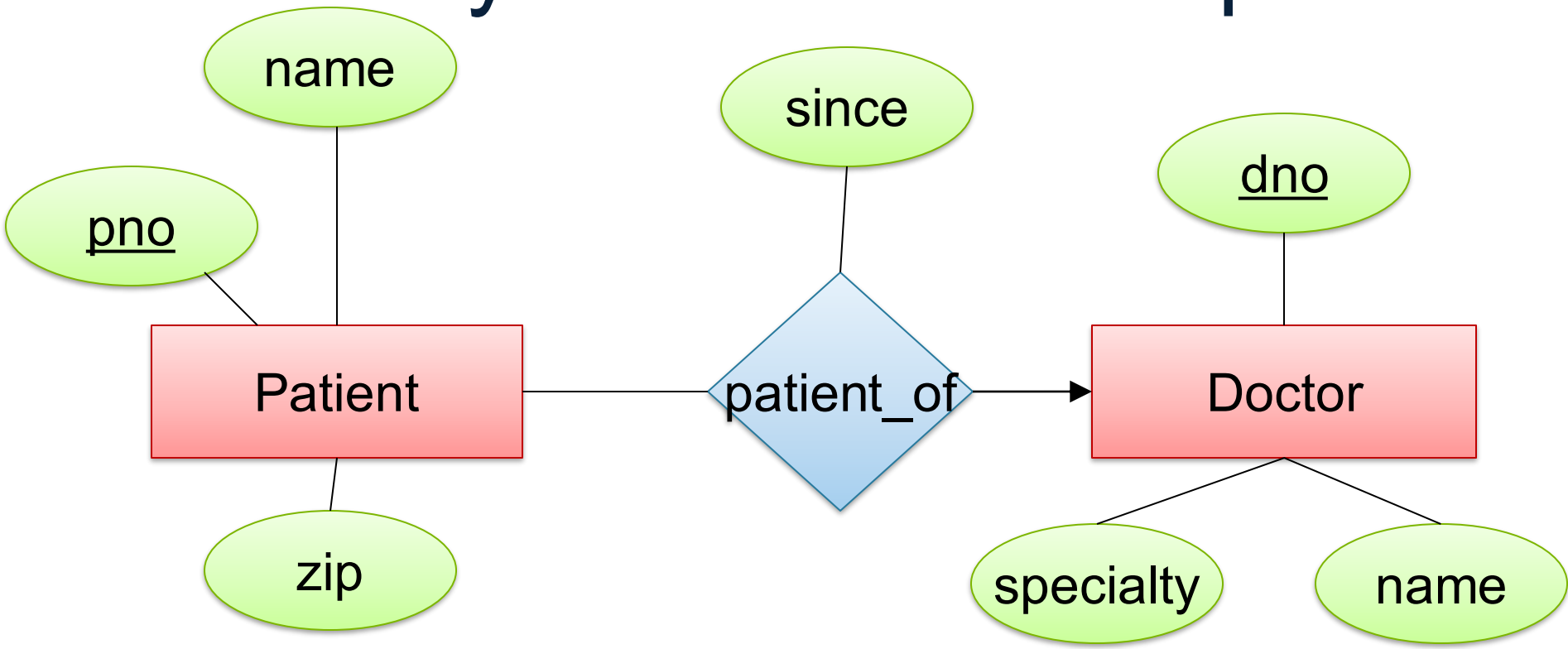
Patient_of

| pno | dno | since |
|---|---|---|
| P311 | D007 | 2001 |
| … | | |

Doctor

| dno | name | spec |
|---|---|---|
| D007 | Bob | cardio |
| … | | |

# Many-one Relationship




Patient

| pno | name | zip | dno | since |
|---|---|---|---|---|
| P311 | Alice | 98765 | D007 | 2001 |
| … | | | | |

Doctor

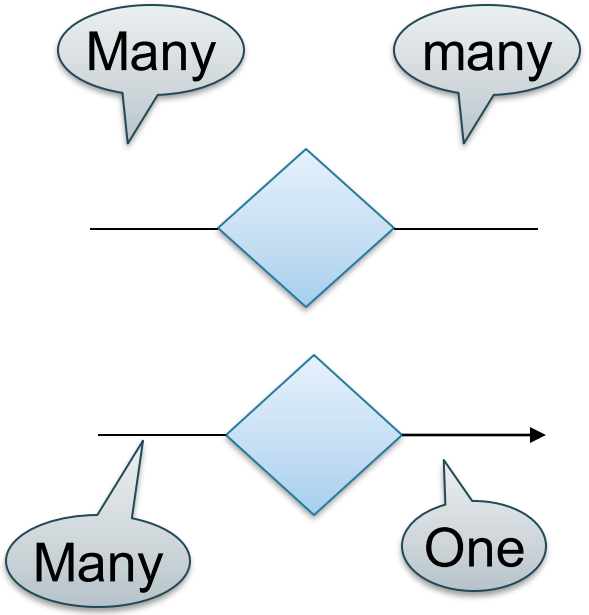
| dno | name | spec |
|---|---|---|
| D007 | Bob | cardio |
| … | | |

# Summary

Many-to-many
- Separate relation

Many-to-one
- No separate relation

One-to-one
- Merge the two entity sets

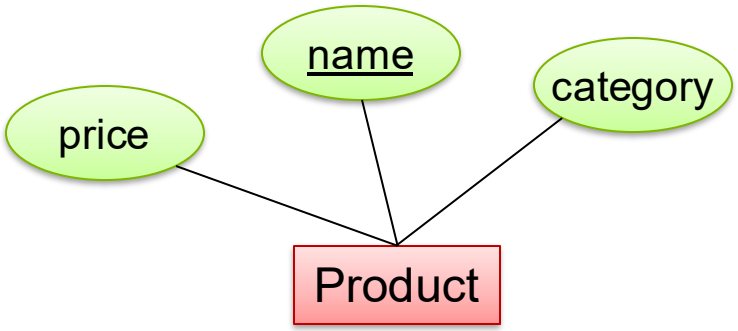


# Subclasses

# Subclasses

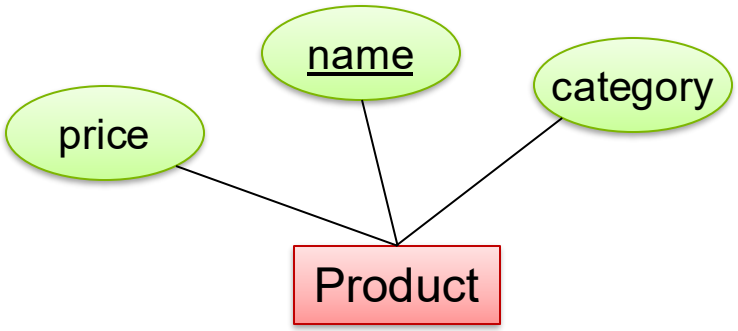
Entity set of Products

# Subclasses




Entity set of Products

# Subclasses

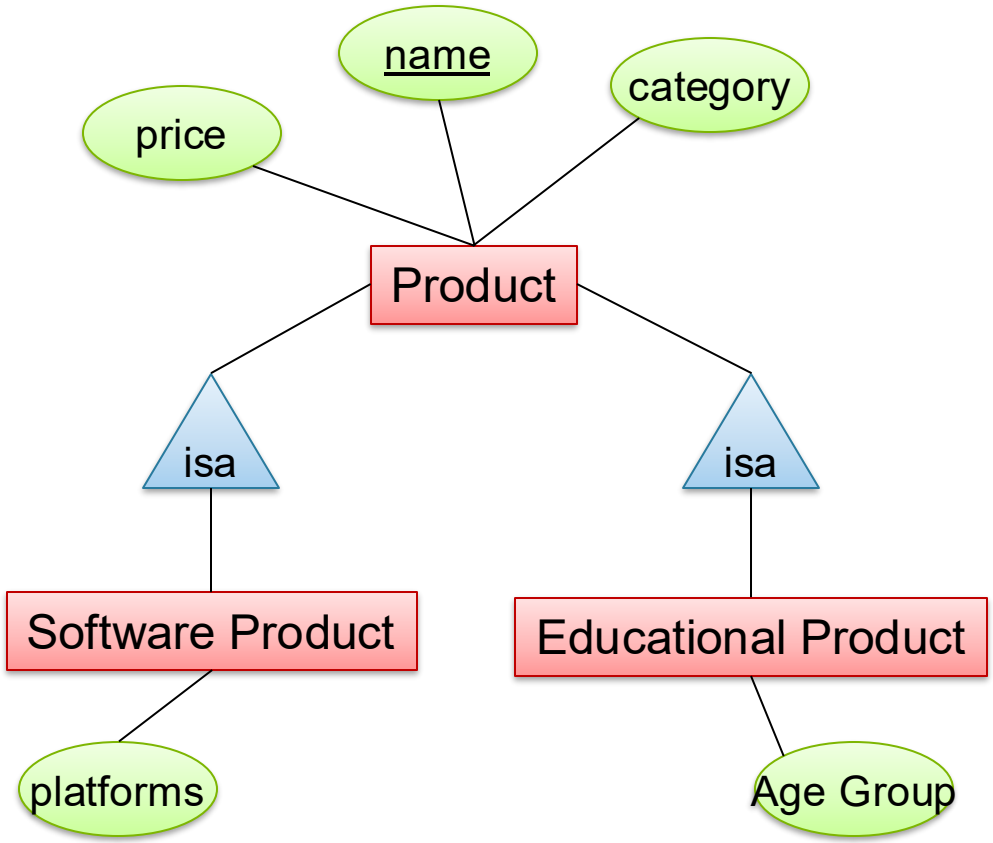



Entity set of Products

Some special products:

- Software products
- Educational products

# Subclasses

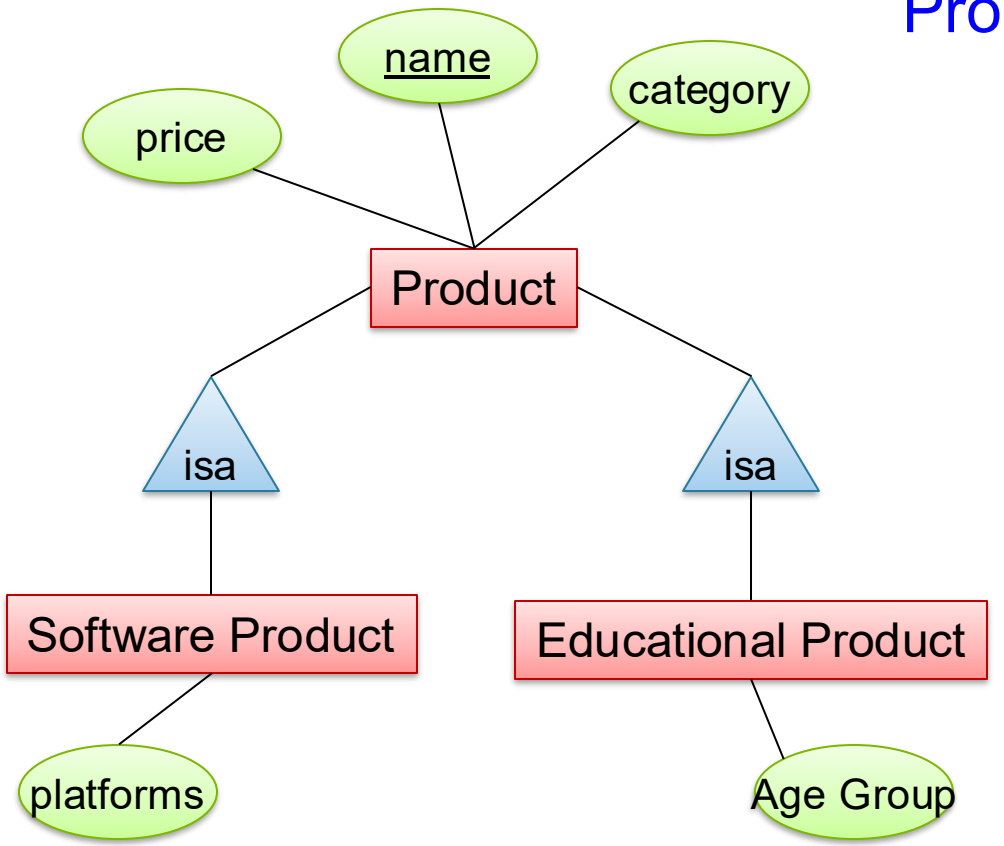



Entity set of Products

Some special products:

- Software products
- Educational products

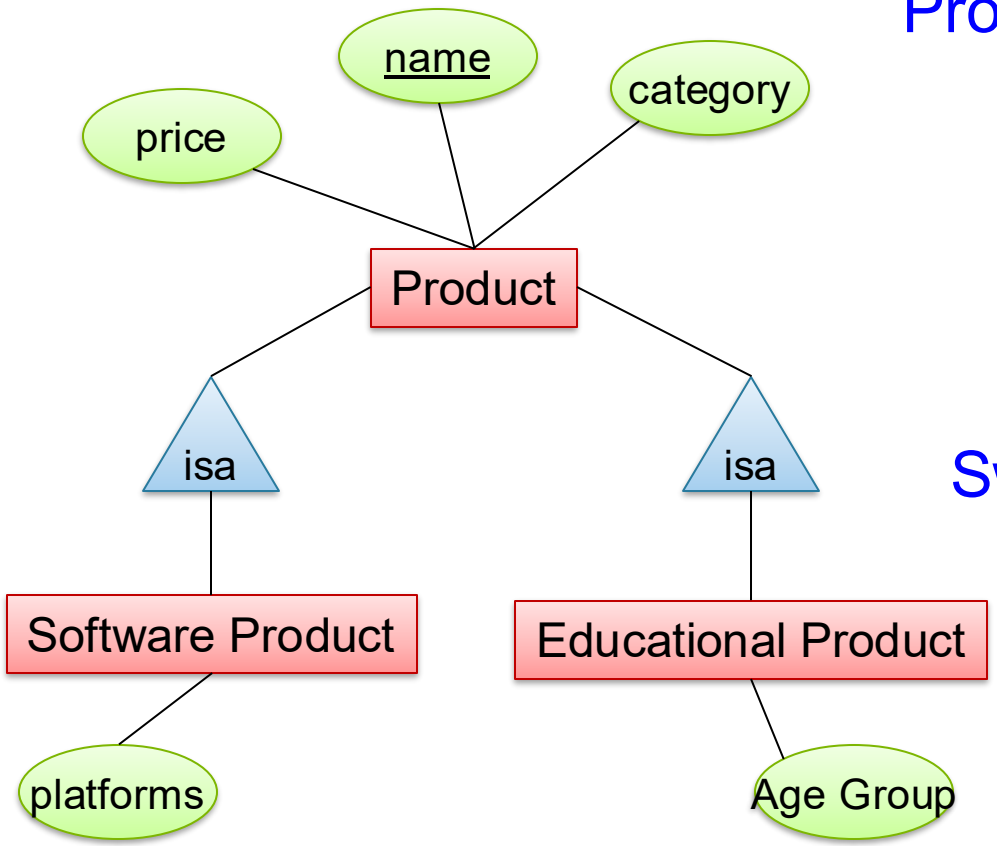
# Subclasses




Product

| Name | Price | Category |
|---|---|---|
| Gizmo | 99 | gadget |
| Camera | 49 | photo |
| Toy | 39 | gadget |

# Subclasses



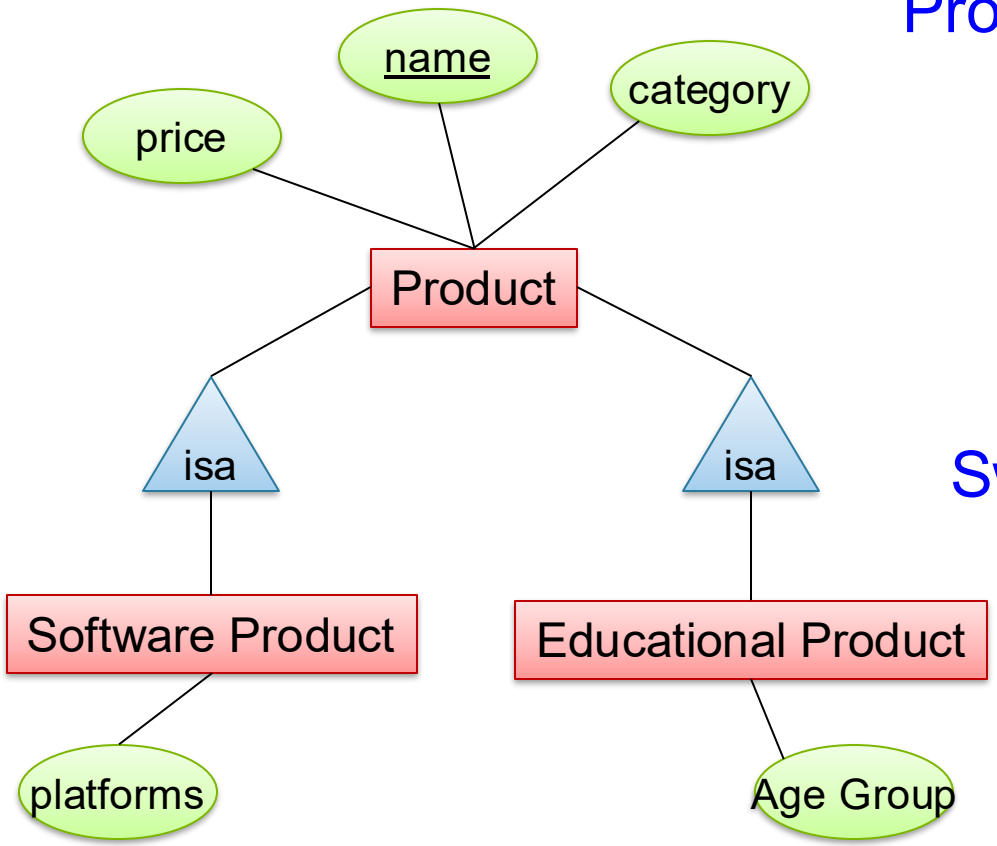

Product

| Name | Price | Category |
|---|---|---|
| Gizmo | 99 | gadget |
| Camera | 49 | photo |
| Toy | 39 | gadget |

Sw.Product

| Name | platforms |
|---|---|
| Gizmo | unix |

# Subclasses




Product

| Name | Price | Category |
|---|---|---|
| Gizmo | 99 | gadget |
| Camera | 49 | photo |
| Toy | 39 | gadget |

Sw.Product

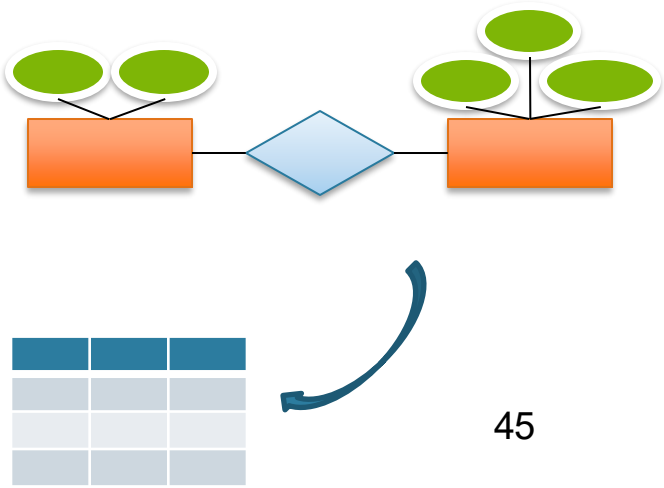
| Name | platforms |
|---|---|
| Gizmo | unix |

Ed.Product

| Name | Age Group |
|---|---|
| Gizmo | toddler |
| Toy | senior |

# ER→SQL: Summary

- Entity set → CREATE TABLE
- m-n Relationship → CREATE TABLE w/ FK
- m-1 Relationship → add FK
- isA → attribute is both Key and FK



# Note on HW1

- You need to create an E/R diagram
- Use entities, relationships, inheritance

- Convert them to SQL Tables correctly:
  - Declare keys/foreign keys
  - Don't create separate tables for N-1 rels
  - Don't use postgres' subclasses

Conceptual Model



Relational Model
+ Schema
+ Constraints



**Conceptual Schema**
**+ Normalization**

Physical Schema
+ Partitioning
+ Indexing

# FDs and Normal Forms

- A functional dependency
  is an expression A→B
- It means that the values in column A uniquely determine those in column B

# FDs and Normal Forms

- A functional dependency
  is an expression A→B
- It means that the values in column A uniquely determine those in column B

| A | B | C |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c3 |
| a3 | b2 | c4 |
| a3 | b1 | c2 |

# FDs and Normal Forms

- A functional dependency
  is an expression A→B
- It means that the values in column A uniquely determine those in column B

| A | B | C |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c3 |
| a3 | b2 | c4 |
| a3 | b1 | c2 |

A→B
C→B

A ↛C
C↛A

# Boyce Codd Normal Form

- A super-key is a set of attributes X such that, for every attribute A: $X \rightarrow A$

# Boyce Codd Normal Form

- A super-key is a set of attributes X such that, for every attribute A: $X \rightarrow A$
- A key is a minimal super-key

# Boyce Codd Normal Form

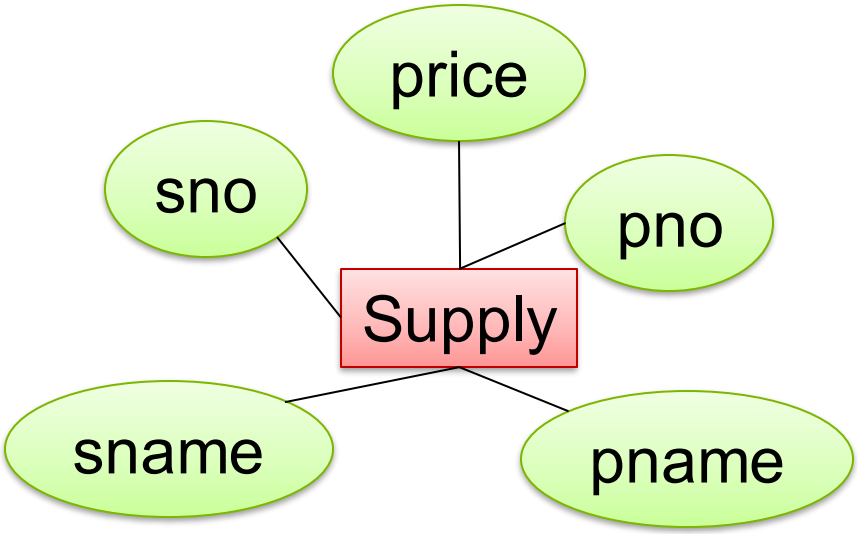
- A super-key is a set of attributes X such that, for every attribute A: $X \rightarrow A$
- A key is a minimal super-key
- A relation is in BCNF if, for every non-trivial FD $X \rightarrow A$, X is a super-key

# Boyce Codd Normal Form

- A super-key is a set of attributes X such that, for every attribute A: X→A
- A key is a minimal super-key
- A relation is in BCNF if, for every non-trivial FD X→A, X is a super-key
- When the relation is not in BCNF then choose a violation X→A and split R into R(XA) and R(X[rest])

# Example

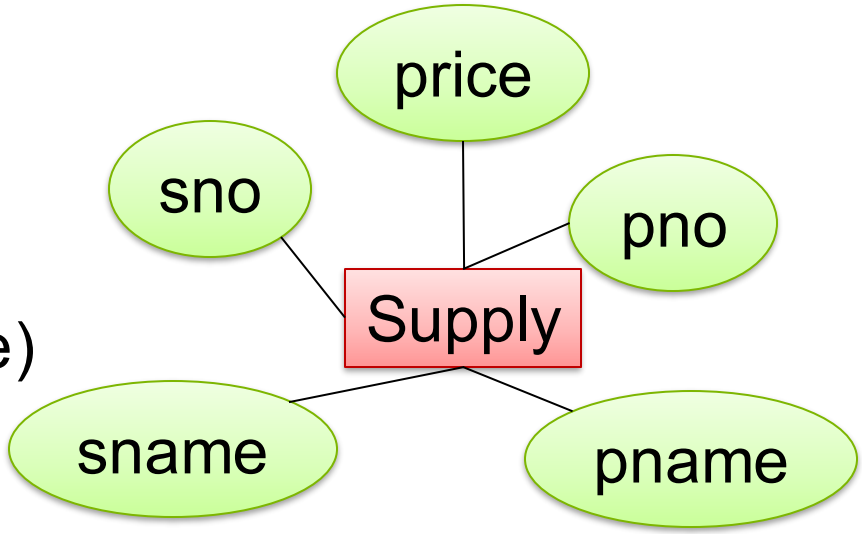
sno→sname
pno→pname



# Example

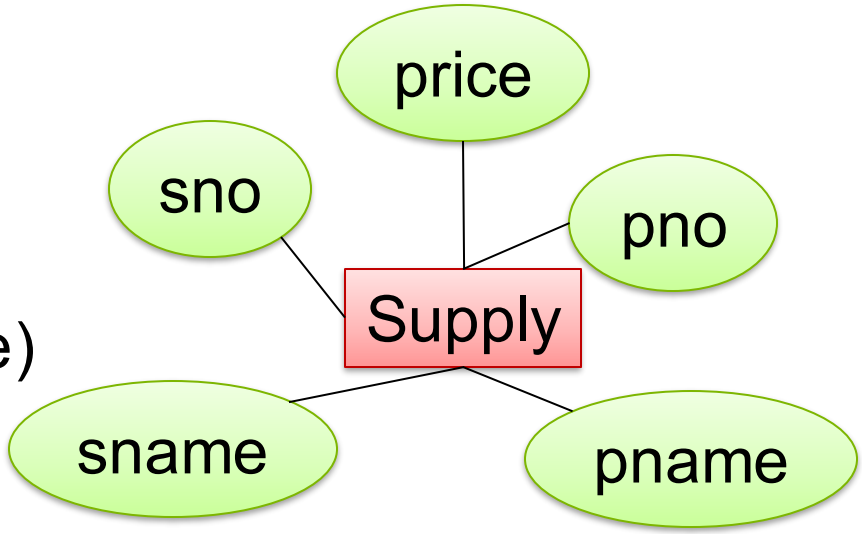
sno→sname
pno→pname

Supply(sno,sname,price,pno,pname)



# Example

sno→sname
pno→pname

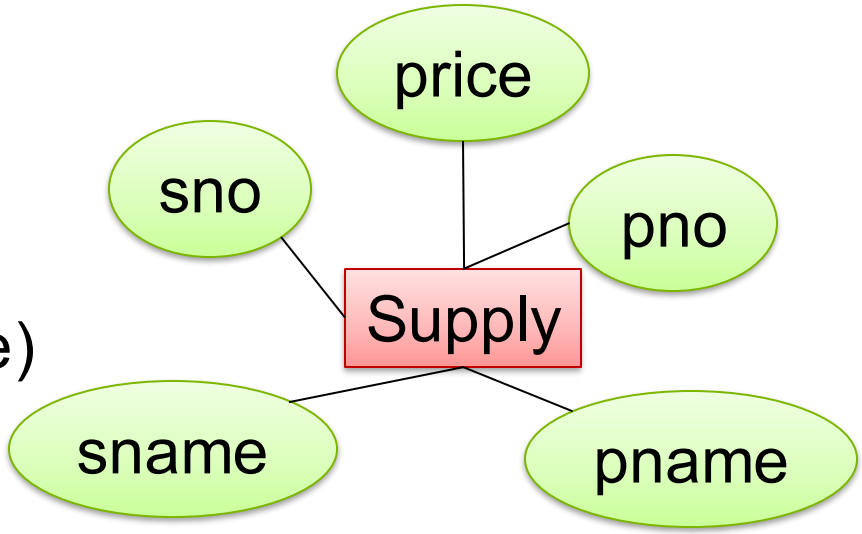



Supply(sno,sname,price,pno,pname)

T1(sno,sname)  T2(sno,price,pno,pname)

# Example

sno→sname
pno→pname




Supply(sno,sname,price,pno,pname)

T1(sno,sname)

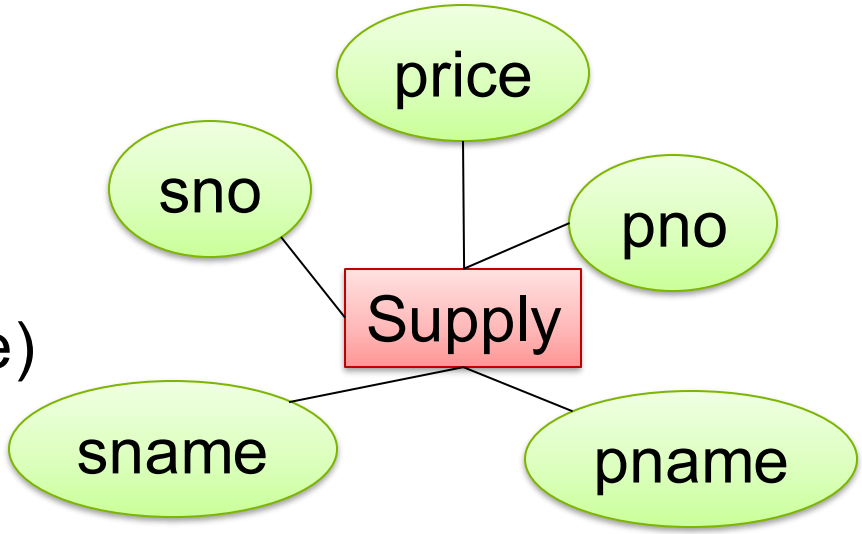
T2(sno,price,pno,pname)

T3(pno,pname)

T4(sno,price,pno)

# Example

sno→sname
pno→pname




Supply(sno,sname,price,pno,pname)

T1(sno,sname)

T2(sno,price,pno,pname)

T3(pno,pname)

T4(sno,price,pno)

```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# Discussion

- BCNF avoids “data anomalies”
- Check details on data anomalies, FDs, Armstrong Axioms, and the BCNF here
- You shouldn’t need this for HW1

Conceptual Model



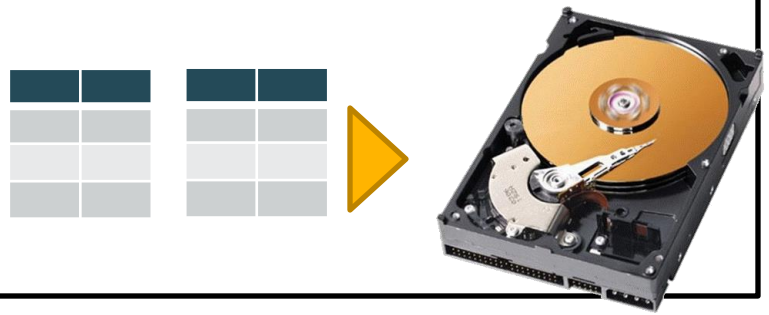
Relational Model
+ Schema
+ Constraints



Conceptual Schema
+ Normalization



Physical Schema
+ Partitioning
+ Indexing



# Indexes

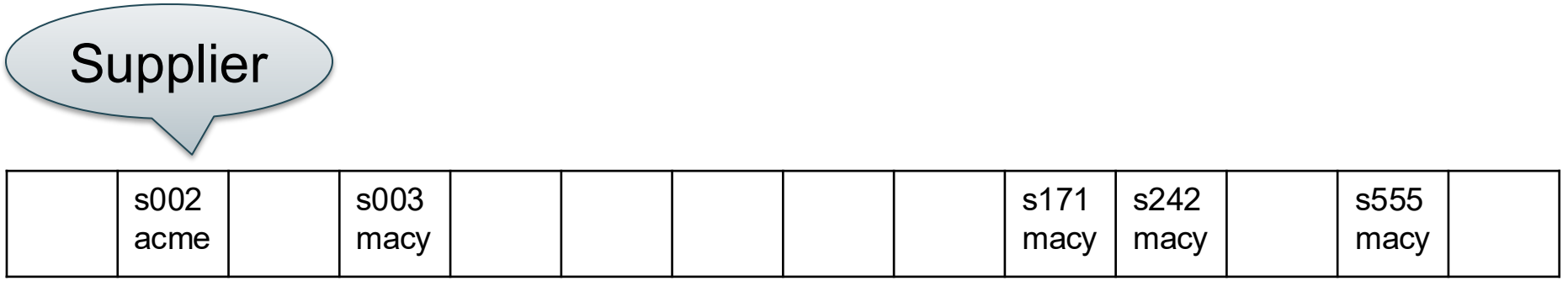
- An index is an auxiliary file that allows direct access to the data file based on the value of an attribute

- It is usually a B+ tree or a hash-table

```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# Indexes

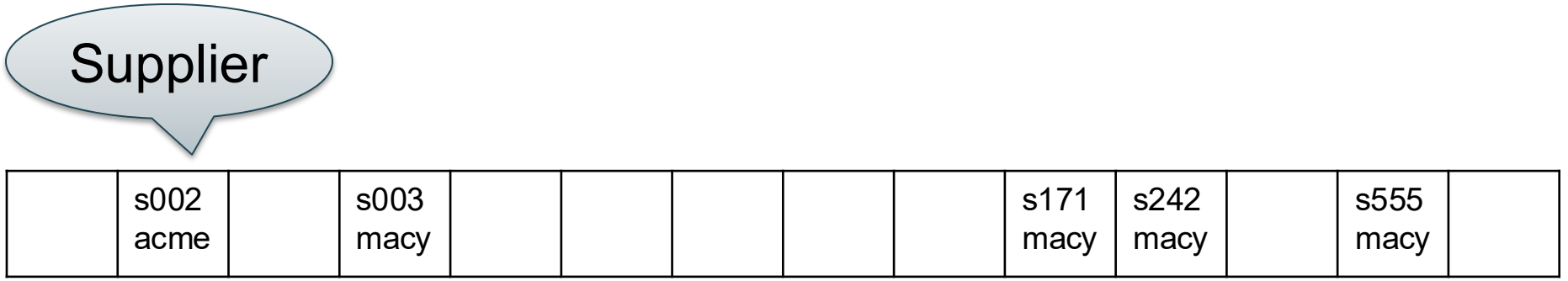
```
CREATE TABLE Supplier(sno int primary key, sname text);
```



```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# Indexes

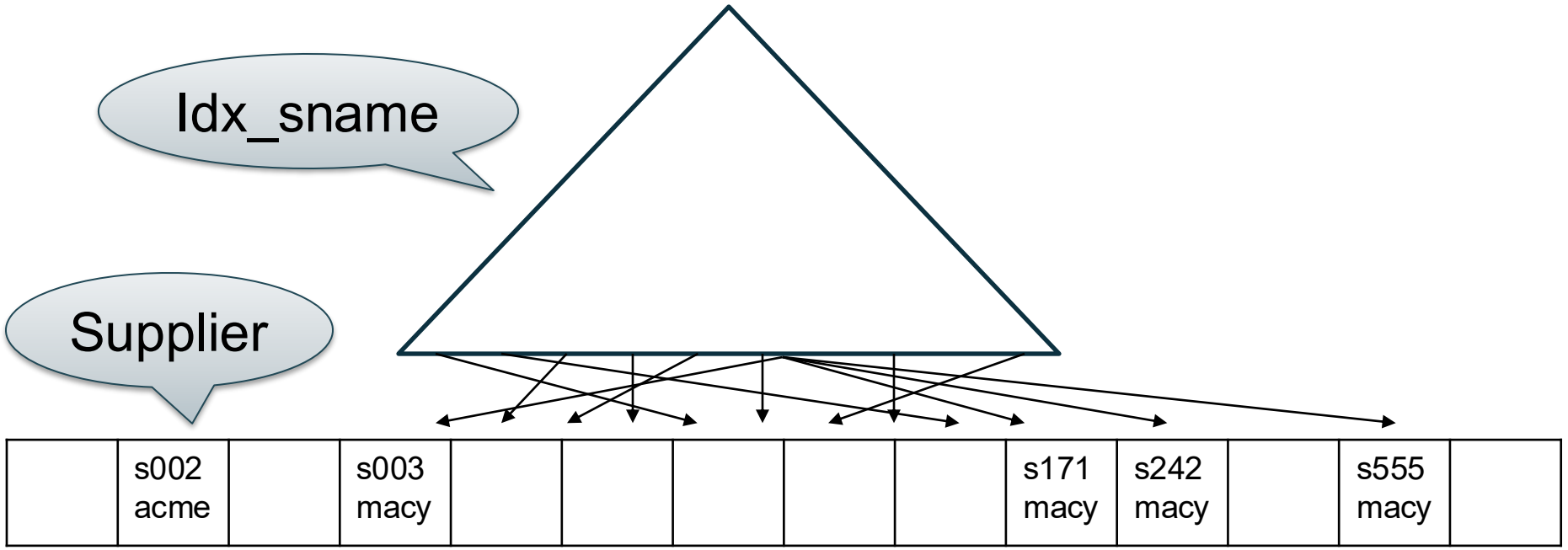
```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname)
```



```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# Indexes

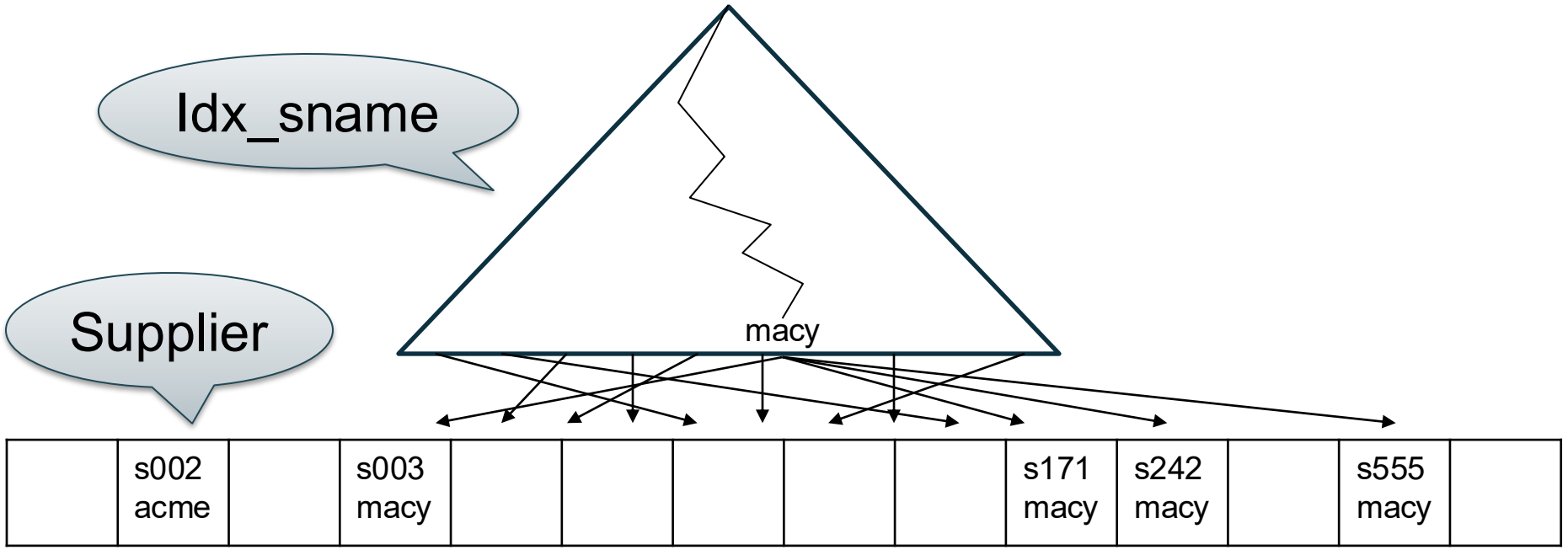
```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname)
```



```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# Indexes

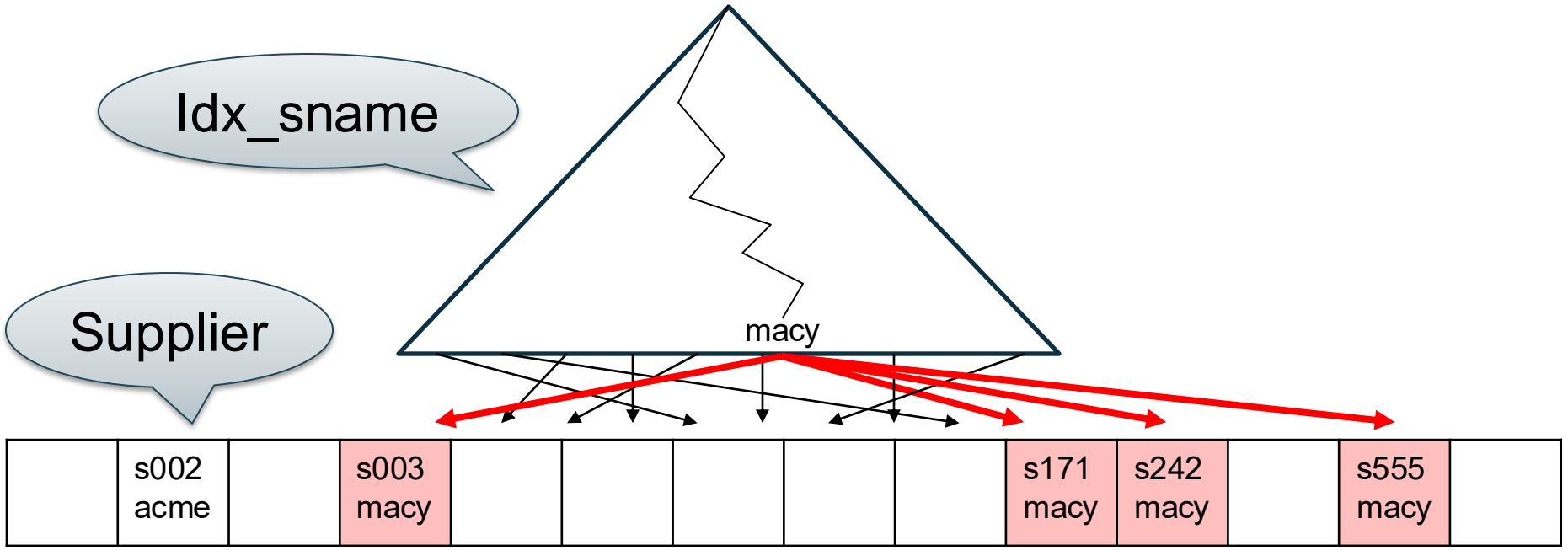
```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname)
```



```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# Indexes

```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname)
```



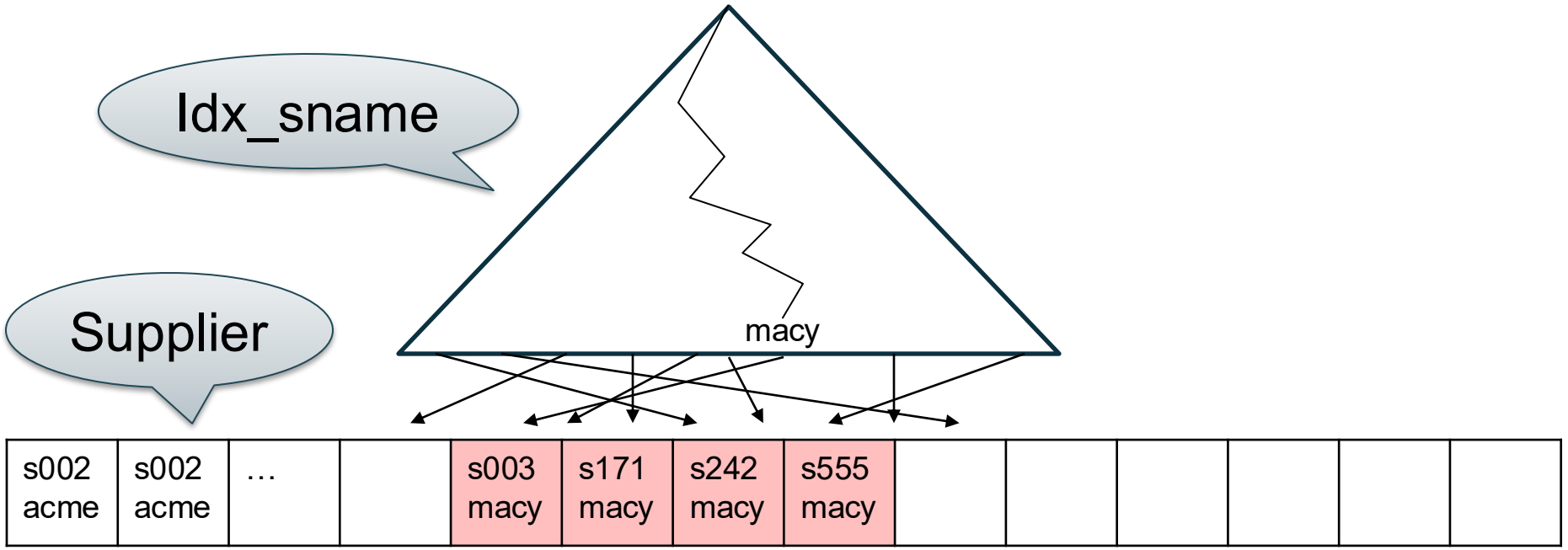
# Indexes

- We say that an index is clustered if the data file is sorted in the order of the index attribute
- Most systems:
  CREATE CLUSTERED INDEX …
- Postgres: first create index, then:
  CLUSTER index_name

```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# Indexes

```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname);
CLUSTER Idx_sname;
```



Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)

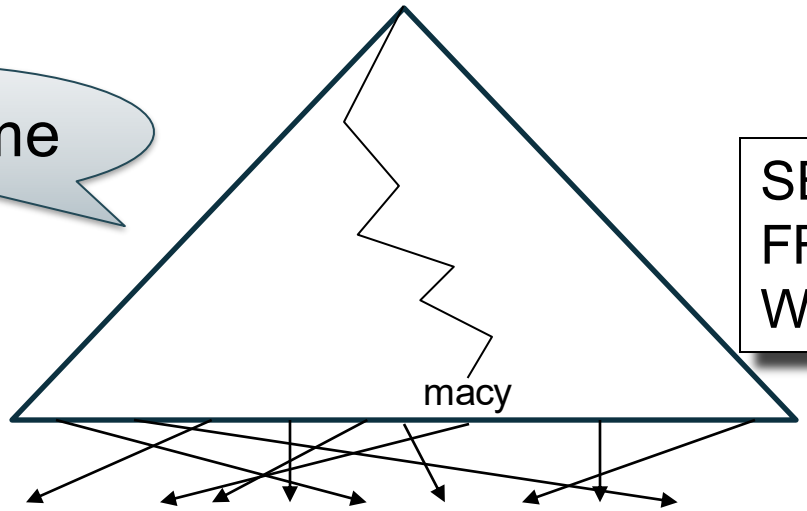
# Indexes

```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname);
CLUSTER Idx_sname;
```

This runs faster now

Idx_sname

```
SELECT *
FROM Supplier
WHERE sname = 'macy';
```




Supplier

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| s002 acme | s002 acme | … | | s003 macy | s171 macy | s242 macy | s555 macy | | | | | | |

```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# When Does an Index Help?

```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname);
```

```
SELECT *
FROM Supplier
WHERE sid = ‘s007’;
```

```
SELECT *
FROM Supplier
WHERE sname = ‘macy’;
```

Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)

# When Does an Index Help?

```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname);
```

```
SELECT *
FROM Supplier
WHERE sid = 's007';
```

```
SELECT *
FROM Supplier
WHERE sname = 'macy';
```

```
SELECT *
FROM Supplier x,
        Supply y
WHERE x.sid=y.sid;
```

```
SELECT *
FROM Supplier x,
        Supply y
WHERE x.sid=y.sid
   and x.name = 'macy';
```

```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# When Does an Index Help?

```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname);
```

```
SELECT *
FROM Supplier
WHERE sid = 's007';
```

NO

```
SELECT *
FROM Supplier
WHERE sname = 'macy';
```

YES

NO

```
SELECT *
FROM Supplier x,
        Supply y
WHERE x.sid=y.sid;
```

YES

```
SELECT *
FROM Supplier x,
        Supply y
WHERE x.sid=y.sid
   and x.name = 'macy';
```

```
Supplier(sno,sname)
Supply(sno,pno,price)
Part(pno,pname)
```

# Where Indexes Help

- Selection based on attribute value
- Join on the index attribute IF few tuples

```
SELECT *
FROM Supplier x, Supply y
WHERE x.sid=y.sid and x.name = 'macy';
```

Index on Supply(sid) useful

- Join on two relations if both clustered

```
SELECT *
FROM Supplier x, Supply y
WHERE x.sid=y.sid;
```

Clustered indexes on both Supplier(sid) and Supply(sid) useful

# Where Indexes Hurt

- Each new index significantly slows down inserts, updates, deletes
- Advice for HW1:
  - CREATE Tables w/o keys, fk's, indexes
  - Insert data
  - Create key/fk's using ALTER TABLE
  - Create indexes
  - Cluster if you want

# Data Transformation

# Data Science Pipeline

- Get the data from somewhere
- We want our own logical schema
- First load data in a triple store…
- …next transform it into logical schema

# Example

A Json file of Parts:

```
{ “Part”: [
    { “pno” : 303,
      ”name”: ”gizmo”,
      “price”: 399,
      “weight”: ”5lb”
     },
     { “pno”: 8778,
      ”name”: “iPhone”,
      ”price”: 999,
      ”storage”:256
     }
…
   ]
}
```

# Example

A Json file of Parts:

```
{ “Part”: [
    { “pno” : 303,
      ”name”: ”gizmo”,
      “price”: 399,
      “weight”: ”5lb”
     },
     { “pno”: 8778,
      ”name”: “iPhone”,
      ”price”: 999,
      ”storage”:256
     }
…
   ]
}
```

- We know that pno is a key
- We don’t know what other attributes might be in the file

# Example

A Json file of Parts:

```
{ “Part”: [
    { “pno” : 303,
      ”name”: ”gizmo”,
      “price”: 399,
      “weight”: ”5lb”
    },
    { “pno”: 8778,
      ”name”: “iPhone”,
      ”price”: 999,
      ”storage”:256
    }
…
   ]
}
```

- We know that pno is a key
- We don’t know what other attributes might be in the file

TripleStore

| pno | pred | val |
|---|---|---|
| 303 | name | gizmo |
| 303 | price | 399 |
| 303 | weight | 5lb |
| 8778 | name | iPhone |
| 8778 | price | 999 |
| 8778 | storage | 256 |
| … | … | … |

# Example

Suppose our logical schema is: `Part(pno,name,price)`

TripleStore

| pno | pred | val |
|---|---|---|
| 303 | name | gizmo |
| 303 | price | 399 |
| 303 | weight | 5lb |
| 8778 | name | iPhone |
| 8778 | price | 999 |
| 8778 | storage | 256 |
| … | … | … |

# Example

Suppose our logical schema is: `Part(pno,name,price)`

```
CREATE TABLE Part AS
  SELECT x.pno as pno,
         x.val as name,
         y.val as price
  FROM    TripleStore x, TripleStore y
  WHERE   x.pno = y.pno
        and x.pred = 'name'
        and y.pred = 'price';
```

TripleStore

| pno | pred | val |
|---|---|---|
| 303 | name | gizmo |
| 303 | price | 399 |
| 303 | weight | 5lb |
| 8778 | name | iPhone |
| 8778 | price | 999 |
| 8778 | storage | 256 |
| … | … | … |

# Example

Suppose our logical schema is: `Part(pno,name,price)`

```
CREATE TABLE Part AS
  SELECT x.pno as pno,
         x.val as name,
         y.val as price
  FROM    TripleStore x, TripleStore y
  WHERE   x.pno = y.pno
        and x.pred = 'name'
        and y.pred = 'price';
```

TripleStore

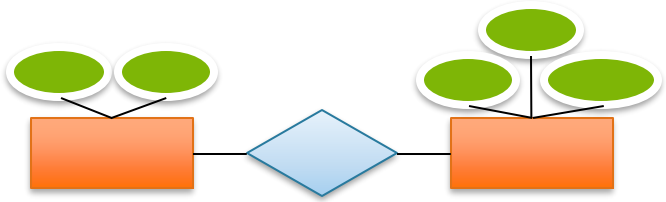
| pno | pred | val |
|---|---|---|
| 303 | name | gizmo |
| 303 | price | 399 |
| 303 | weight | 5lb |
| 8778 | name | iPhone |
| 8778 | price | 999 |
| 8778 | storage | 256 |
| … | … | … |

Part

| pno | name | price |
|---|---|---|
| 303 | gizmo | 399 |
| 8778 | iPhone | 999 |
| … | … | … |

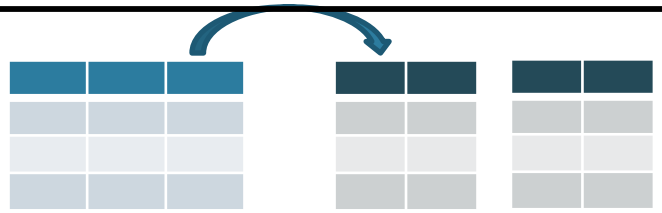
# Summary

Conceptual Model



Relational Model
+ Schema
+ Constraints

Conceptual Schema
+ Normalization



Physical Schema
+ Partitioning
+ Indexing