CSEP544
Data Management

SQL, Database Design

Announcement

- HW1 is due on January 26

Recap

Relational data model

SQL

- SELECT-FROM-WHERE
- NULLs
- Joins, self-joins, outer-joins
- Aggregates, Group-by

Aggregates

Aggregate Operator

Aggregate op: set of values to single value

Aggregates in SQL:

- sum(1, 4, 3, 4) = 1+4+3+4 = 12
- max(1, 4, 3, 4) = 4
- min(1, 4, 3, 4) = 1
- count(1, 4, 3, 4) = 4
- avg(1, 4, 3, 4) = 3

Aggregate Operator

Aggregate op: set of values to single value

Aggregates in SQL:

- sum(1, 4, 3, 4) = 1+4+3+4 = 12
- max(1, 4, 3, 4) = 4
- min(1, 4, 3, 4) = 1
- count(1, **4**, 3, **4**) = 4
- avg(1, **4**, 3, **4**) = 3

May have duplicates

Count

Supplier

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

```
SELECT  count(*)
FROM    Part
```

4

```
SELECT  count(sstate)
FROM    Part
```

4

```
SELECT  count(DISTINCT sstate)
FROM    Part
```

2

GROUP-BY

```
SELECT …
FROM   …
WHERE  …
GROUP BY …
```

Supplier(sno,sname,**scity**,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

Aggregates and Group-By

Count # of Parts

```
SELECT count(*)
FROM   Part
```

Count # of Parts supplied by each city

```
SELECT x.scity, count(*)
FROM   Supplier x, Supply y, Part z
WHERE  x.sno = y.sno  and y.pno = z.pno
GROUP BY x.scity
```

Supplier(sno,sname,**scity**,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

Aggregates and Group-By

Count # of Parts

```
SELECT count(*) as C
FROM   Part
```

C
1540

Count # of Parts supplied by each city

```
SELECT x.scity, count(*)
FROM   Supplier x, Supply y, Part z
WHERE  x.sno = y.sno  and y.pno = z.pno
GROUP BY x.scity
```

Supplier(sno,sname,**scity**,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

Aggregates and Group-By

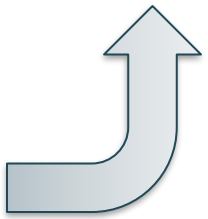
Count # of Parts

```
SELECT count(*) as C
FROM   Part
```

C
1540

Count # of Parts supplied by each city

```
SELECT x.scity, count(*) as C
FROM   Supplier x, Supply y, Part z
WHERE  x.sno = y.sno  and y.pno = z.pno
GROUP BY x.scity
```



City	C
Seattle	300
NYC	240
…	

Discussion

- GROUP-BY without an aggregate is equivalent to DISTINCT

- Every attribute in SELECT that is not aggregated must occur in GROUP-BY

See last lecture

The HAVING Clause

The HAVING Clause

```
SELECT …
FROM   …
WHERE  …
GROUP BY …
HAVING [condition w/ aggregates]
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

HAVING Clause

Compute the total quantity supplied by each supplier in ‘WA’

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

HAVING Clause

Compute the total quantity supplied by each supplier in 'WA'

```
SELECT x.sno, x.sname, sum(y.qty)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno and x.sstate='WA'
GROUP BY x.sno, x.sname
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

HAVING Clause

Compute the total quantity supplied by each supplier in 'WA'

```
SELECT x.sno, x.sname, sum(y.qty)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno and x.sstate='WA'
GROUP BY x.sno, x.sname
```

Compute the total quantity supplied by each supplier who supplied > 100 parts

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

HAVING Clause

Compute the total quantity supplied by each supplier in 'WA'

```
SELECT x.sno, x.sname, sum(y.qty)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno and x.sstate='WA'
GROUP BY x.sno, x.sname
```

Compute the total quantity supplied by each supplier who supplied > 100 parts

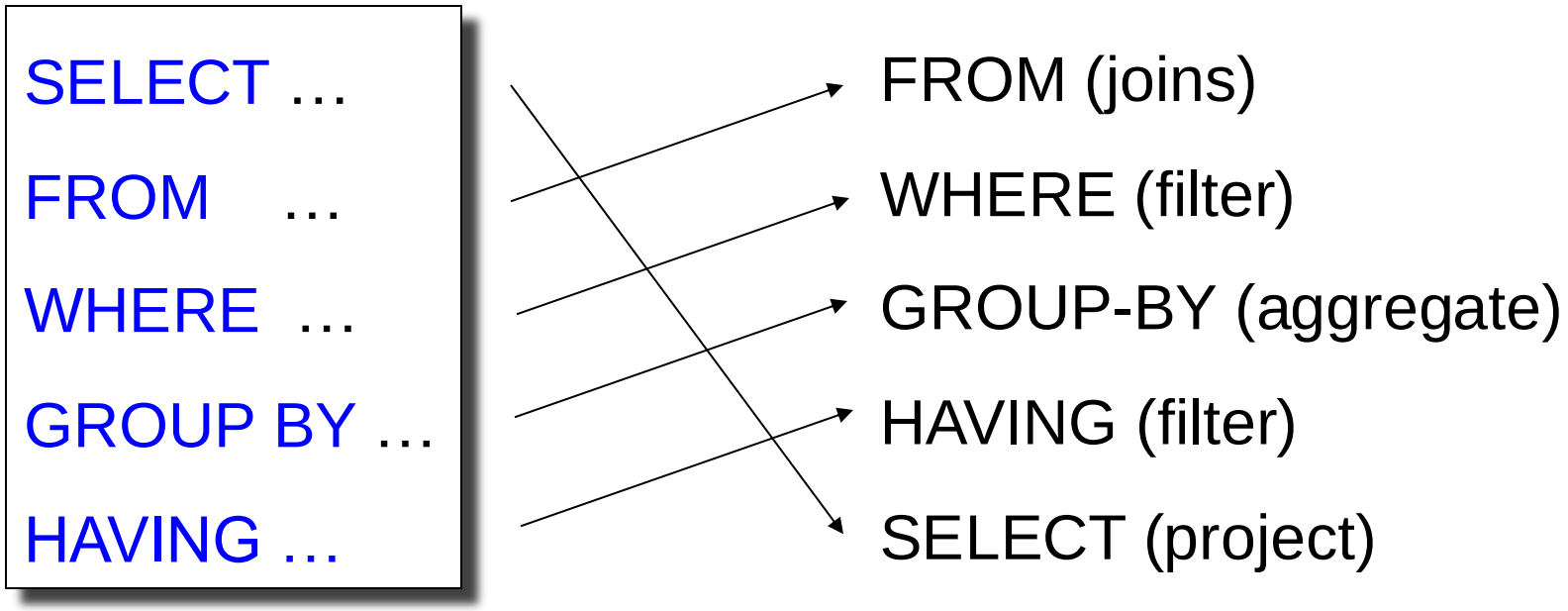
```
SELECT x.sno, x.sname, sum(y.qty)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno, x.sname
HAVING count(*) > 100
```

Semantics

```
SELECT …
FROM   …
WHERE  …
GROUP BY …
HAVING …
```

Paper *SQL Has Problems.* What is the logical order?

Semantics



Paper *SQL Has Problems.* What is the logical order?

Semantics

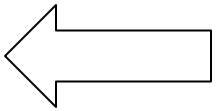
SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1
GROUP BY a_1, …, a_k
HAVING condition2 -- may have aggs

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1
GROUP BY a_1, …, a_k
HAVING condition2 -- may have aggs

Step 1: FROM-WHERE

a_1	…	a_k	b_1	…	b_1

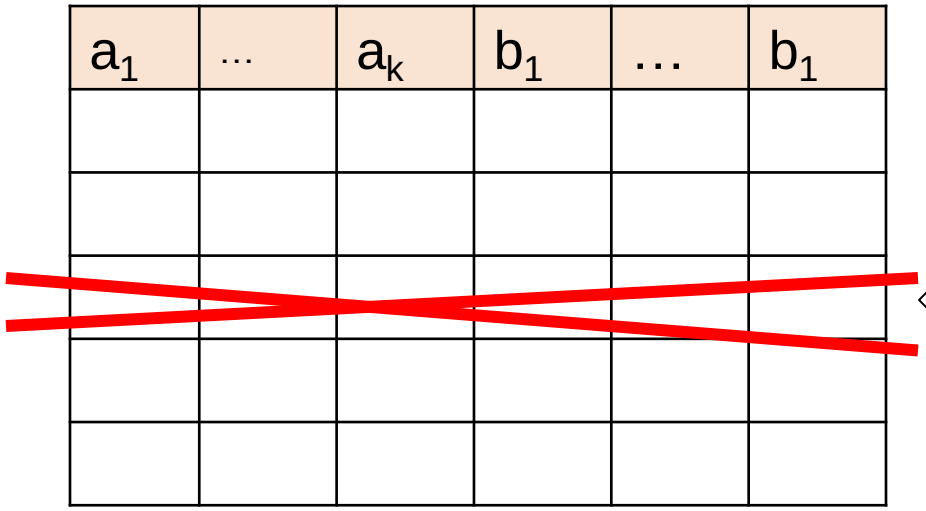


Check
WHERE condition1
in each row

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1
GROUP BY a_1, …, a_k
HAVING condition2 -- may have aggs

Step 1: FROM-WHERE



Check
WHERE condition1
in each row

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1
GROUP BY a_1, …, a_k
HAVING condition2 -- may have aggs

Step 1: FROM-WHERE

a_1	…	a_k	b_1	…	b_1

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1
GROUP BY a_1, …, a_k
HAVING condition2 -- may have aggs

Step 2: GROUP BY

a_1	…	a_k	b_1	…	b_1
u	…	v			
u		v			
p		q			
p		q			
p		q			

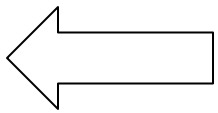
All attributes a_1, …, a_k, have the same value inside each group

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1
GROUP BY a_1, …, a_k
HAVING condition2 -- may have aggs

Step 3: HAVING

a_1	…	a_k	b_1	…	b_1
u	…	v			
u		v			
p		q			
p		q			
p		q			

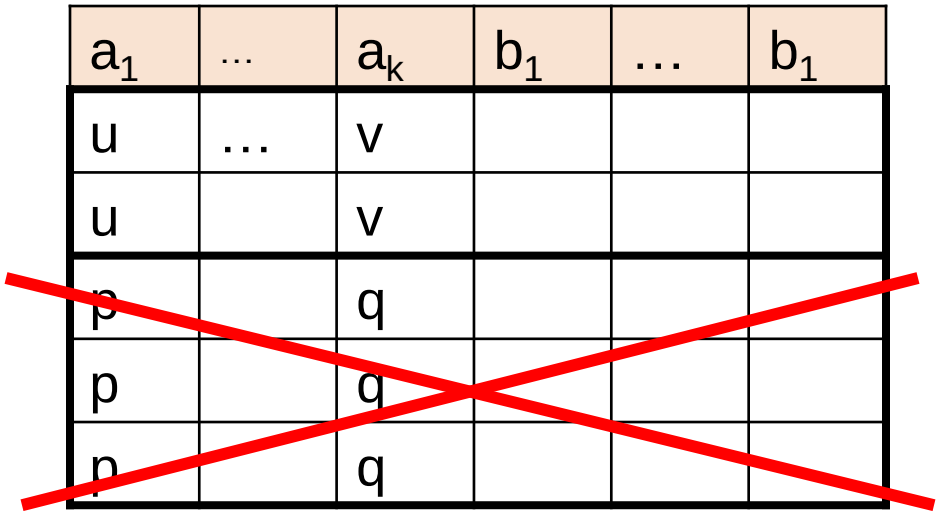


Check condition2 in each group

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1
GROUP BY a_1, …, a_k
HAVING condition2 -- may have aggs

Step 3: HAVING



Check condition2 in each group

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1
GROUP BY a_1, …, a_k
HAVING condition2 -- may have aggs

Step 3: HAVING

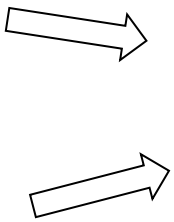
a_1	…	a_k	b_1	…	b_1
u	…	v			
u		v			
p		q			
p		q			
p		q			

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1
GROUP BY a_1, …, a_k
HAVING condition2 -- may have aggs

Step 4: SELECT

a_1	…	a_k	b_1	…	b_1
u	…	v			
u		v			
p		q			
p		q			
p		q			



a_1	…	a_k	agg_1	agg_2
u	…	v		
p		q		

Each group → one output

Discussion

- GROUP-BY is very versatile in SQL
- No analogous in programming languages: use nested loops instead

```
SELECT x.sno, count(*)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

Discussion

- GROUP-BY is very versatile in SQL
- No analogous in programming languages: use nested loops instead

```
SELECT x.sno, count(*)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

```
for x in Supplier:
  c = 0
  for y in Supply:
    if x.sno==y.sno:
      c = c+1
```

Discussion

- GROUP-BY is very versatile in SQL
- No analogous in programming languages: use nested loops instead

```
SELECT x.sno, count(*)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

```
for x in Supplier:
  c = 0
  for y in Supply:
    if x.sno==y.sno:
      c = c+1
```

- The empty group problem (next)

Empty Groups

Empty Groups Problem

- Every group is non-empty
- Consequences:
  - count(*) > 0
  - sum(…) > 0 (assuming numbers are >0)
- Sometimes we want to return 0 counts:
  - Parts that never sold
  - Suppliers that never supplied
- Use outer joins: count(…) skips NULLs

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Empty Groups Problem

Compute the number of parts supplied by each supplier

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Empty Groups Problem

Compute the number of parts supplied by each supplier

```
SELECT x.sno, count(*)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

Suppliers who never supplied any part will be missing: count(*) > 0

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Empty Groups Problem

Compute the number of parts supplied by each supplier

```
SELECT x.sno, count(*)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

Suppliers who never supplied any part will be missing: count(*) > 0

```
SELECT x.sno, count(y.sno)
FROM   Supplier x
  LEFT OUTER JOIN Supply y
ON  x.sno=y.sno
GROUP BY x.sno
```

Now we can get count(*)=0

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Empty Groups Problem

Compute the number of parts supplied by each supplier

```
SELECT x.sno, count(*)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

Suppliers who never supplied any part will be missing: count(*) > 0

```
SELECT x.sno, count(y.sno)
FROM   Supplier x
  LEFT OUTER JOIN Supply y
ON  x.sno=y.sno
GROUP BY x.sno
```

Now we can get count(*)=0

Can we write count(*)?

The WITH Clause

WITH Clause

```
WITH
        tbl1 AS (SELECT … FROM …),
        tbl2 AS (SELECT … FROM …),
        …
SELECT … FROM …[tbl1, tbl2,…] …
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Example

Warmup: find all parts supplied from Seattle

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Example

Warmup: find all parts supplied from Seattle

```
SELECT z.*
FROM Supplier x, Supply y, Part z
WHERE z.scity = 'Seattle';
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Example

Warmup: find all parts supplied from Seattle

```
SELECT z.*
FROM Supplier x, Supply y, Part z
WHERE z.scity = 'Seattle';
```

What is missing?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Example

Warmup: find all parts supplied from Seattle

```
SELECT DISTINCT z.*
FROM Supplier x, Supply y, Part z
WHERE z.scity = 'Seattle';
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Example

Find the average psize of all parts supplied from Seattle

```
SELECT avg(z.psize)
FROM Supplier x, Supply y, Part z
WHERE z.scity = 'Seattle';
```

What is wrong?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Example

Find the average psize of all parts supplied from Seattle

```
WITH Tmp AS (
        SELECT DISTINCT z.pno, z.psize
        FROM Supplier x, Supply y, Part z
        WHERE z.scity = 'Seattle')
SELECT avg(psize)
FROM Tmp;
```

Subqueries

Subqueries

- A subquery is a self-contained SQL query that occurs inside another query
- The subquery can be any of these clauses:
  - FROM
  - SELECT
  - WHERE
  - HAVING

Subqueries in FROM Clause

- Subquery in FROM: the same as in WITH
- Sometimes WITH is easier to read
- Some DBMS may not support both

Subqueries in FROM Clause

```
WITH Tmp AS (SELECT DISTINCT z.pno, z.psize
             FROM Supplier x, Supply y, Part z
             WHERE z.scity = 'Seattle')
SELECT avg(psize)
FROM Tmp;
```

Subqueries in FROM Clause

```
WITH Tmp AS (SELECT DISTINCT z.pno, z.psize
             FROM Supplier x, Supply y, Part z
             WHERE z.scity = 'Seattle')
SELECT avg(psize)
FROM Tmp;
```

same as:

```
SELECT avg(W.psize)
FROM (SELECT DISTINCT z.pno, z.psize
      FROM Supplier x, Supply y, Part z
      WHERE z.scity = 'Seattle') as W;
```

Subqueries in SELECT

- SELECT: only scalar expressions
- May use subquery in SELECT if it returns a single value

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subqueries in SELECT

Compute the total quantity supplied by each supplier in Seattle

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subqueries in SELECT

Compute the total quantity supplied by each supplier in Seattle

```
SELECT x.sno,
       ?

FROM   Supplier x
WHERE  x.scity = 'Seattle';
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subqueries in SELECT

Compute the total quantity supplied by each supplier in Seattle

```
SELECT x.sno, (SELECT sum(y.qty)
                      FROM Supply y
                      WHERE x.sno = y.sno) AS T
FROM    Supplier x
WHERE  x.scity = 'Seattle';
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subqueries in SELECT

Compute the total quantity supplied by each supplier in Seattle

```
SELECT x.sno, (SELECT sum(y.qty)
                FROM Supply y
                WHERE x.sno = y.sno) AS T
FROM   Supplier x
WHERE  x.scity = 'Seattle';
```

```
SELECT x.sno, sum(y.qty) as T
FROM   Supplier x, Supply y
WHERE x.sno = y.sno
      and x.scity = 'Seattle'
GROUP BY x.sno
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

Subqueries in SELECT

Compute the total quantity supplied by each supplier in Seattle

```
SELECT x.sno, (SELECT sum(y.qty)
                FROM Supply y
                WHERE x.sno = y.sno) AS T
FROM   Supplier x
WHERE  x.scity = 'Seattle';
```

```
SELECT x.sno, sum(y.qty) as T
FROM   Supplier x, Supply y
WHERE x.sno = y.sno
      and x.scity = 'Seattle'
GROUP BY x.sno
```

Not equivalent!
WHY?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subqueries in SELECT

Compute the total quantity supplied by each supplier in Seattle

```
SELECT x.sno, (SELECT sum(y.qty)
                      FROM Supply y
                      WHERE x.sno = y.sno) AS T
FROM    Supplier x
WHERE  x.scity = 'Seattle';
```

Now they are equivalent

```
SELECT x.sno, sum(y.qty) as T
FROM    Supplier x LEFT OUTER JOIN Supply y
ON x.sno = y.sno
       and x.scity = 'Seattle'
GROUP BY x.sno
```

Subqueries in WHERE

Three SQL constructs:

- [NOT] EXISTS (SELECT…)
- X [NOT] IN (SELECT …)
- X > ALL | ANY (SELECT …)

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

Simplified schema

Subqueries in WHERE

Find all parts that have some supplier offering them for < $100

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

Simplified schema

Subqueries in WHERE

Find all parts that have some supplier offering them for < $100

```
SELECT DISTINCT a.pno, a.pname
FROM    Part a, Supply b
WHERE b.price < 100 and b.pno = a.pno
```

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

Simplified schema

Subqueries in WHERE

Find all parts that have some supplier offering them for < $100

```
SELECT DISTINCT a.pno, a.pname
FROM    Part a, Supply b
WHERE b.price < 100 and b.pno = a.pno
```

```
SELECT a.pno, a.pname
FROM    Part a
WHERE EXISTS
            (SELECT *
             FROM Supply b
             WHERE b.price < 100
                   and b.pno = a.pno)
```

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

Natural language is ambiguous.
Question above is the same as:

Find all parts that are offered only for < $100

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

Find the other parts:

all parts that have some supplier offering them for >= $100

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

Find the other parts:

all parts that have some supplier offering them for >= $100

```
SELECT a.pno, a.pname
FROM   Part a
WHERE EXISTS
           (SELECT *
            FROM Supply b
            WHERE b.price >= 100
                  and b.pno = a.pno)
```

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

Find the other parts:
all parts that have some supplier offering them for >= $100

```
SELECT a.pno, a.pname
FROM   Part a
WHERE NOT EXISTS
          (SELECT *
           FROM Supply b
           WHERE b.price >= 100
                 and b.pno = a.pno)
```

Negate!

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

$$Answer(x, y) = Part(x, y) \wedge \exists z, w(Supply(z, x, w) \wedge w < 100)$$

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

$$Answer(x,y) = Part(x,y) \wedge \exists z,w(Supply(z,x,w) \wedge w < 100)$$

"**Exists**" quantifier

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

$$Answer(x,y) = Part(x,y) \wedge \exists z,w(Supply(z,x,w) \wedge w < 100)$$

Find all parts where all supplier offering them charge < $100

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

$$Answer(x,y) = Part(x,y) \wedge \exists z, w(Supply(z,x,w) \wedge w < 100)$$

Find all parts where all supplier offering them charge < $100

$$Answer(x,y) = Part(x,y) \wedge \forall z, w(Supply(z,x,w) \Rightarrow w < 100)$$

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

$$Answer(x,y) = Part(x,y) \wedge \exists z,w(Supply(z,x,w) \wedge w < 100)$$

Find all parts where all supplier offering them charge < $100

$$Answer(x,y) = Part(x,y) \wedge \forall z,w(Supply(z,x,w) \Rightarrow w < 100)$$

"**For all**" quantifier

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

$$Answer(x,y) = Part(x,y) \wedge \exists z,w(Supply(z,x,w) \wedge w < 100)$$

Find all parts where all supplier offering them charge < $100

$$Answer(x,y) = Part(x,y) \wedge \forall z,w(Supply(z,x,w) \Rightarrow w < 100)$$

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

$$Answer(x,y) = Part(x,y) \wedge \exists z,w(Supply(z,x,w) \wedge w < 100)$$

Find all parts where all supplier offering them charge < $100

$$Answer(x,y) = Part(x,y) \wedge \forall z,w(Supply(z,x,w) \Rightarrow w < 100)$$

$$= Part(x,y) \wedge \neg(\exists z,w\ \neg(Supply(z,x,w) \Rightarrow w < 100))$$

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

$$Answer(x,y) = Part(x,y) \wedge \exists z,w(Supply(z,x,w) \wedge w < 100)$$

Find all parts where all supplier offering them charge < $100

$$Answer(x,y) = Part(x,y) \wedge \forall z,w(Supply(z,x,w) \Rightarrow w < 100)$$

$$= Part(x,y) \wedge \neg(\exists z,w\ \neg(Supply(z,x,w) \Rightarrow w < 100))$$

$$= Part(x,y) \wedge \neg(\exists z,w\ (Supply(z,x,w) \wedge w \geq 100))$$

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

Simplified schema

Understanding Quantifiers

Find all parts that have some supplier offering them for < $100

$$Answer(x,y) = Part(x,y) \wedge \exists z,w(Supply(z,x,w) \wedge w < 100)$$

Find all parts where all supplier offering them charge < $100

$$Answer(x,y) = Part(x,y) \wedge \forall z,w(Supply(z,x,w) \Rightarrow w < 100)$$

$$= Part(x,y) \wedge \neg(\exists z,w\ \neg(\boldsymbol{Supply(z,x,w) \Rightarrow w < 100}))$$

$$= Part(x,y) \wedge \neg(\exists z,w\ (\boldsymbol{Supply(z,x,w) \wedge w \geq 100}))$$

$$\neg(A \Rightarrow B) = A \wedge \neg B$$

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

Simplified schema

Understanding Quantifiers

Find all parts supplied by all suppliers

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

Simplified schema

Understanding Quantifiers

Find all parts supplied by all suppliers

$$Answer(x, y) =$$

$$= Part(x, y) \wedge \forall u, v(Supplier(u, v) \Rightarrow \exists p\, Supply(u, x, p))$$

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

Simplified schema

Understanding Quantifiers

Find all parts supplied by all suppliers

$$Answer(x,y) =$$

$$= Part(x,y) \wedge \forall u,v(Supplier(u,v) \Rightarrow \exists p\ Supply(u,x,p))$$

$$= Part(x,y) \wedge \neg\exists u,v(Supplier(u,v) \wedge \neg\exists p\ Supply(u,x,p))$$

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

Simplified schema

Understanding Quantifiers

Find all parts supplied by all suppliers

$$Answer(x,y) =$$

$$= Part(x,y) \wedge \forall u,v(Supplier(u,v) \Rightarrow \exists p\ Supply(u,x,p))$$

$$= \boldsymbol{Part(x,y)} \wedge \neg\exists u,v(Supplier(u,v) \wedge \neg\exists p\ Supply(u,x,p))$$

```
SELECT a.pno, a.pname
FROM   Part a
WHERE
```

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

Simplified schema

Understanding Quantifiers

Find all parts supplied by all suppliers

$$Answer(x,y) =$$

$$= Part(x,y) \wedge \forall u,v(Supplier(u,v) \Rightarrow \exists p\ Supply(u,x,p))$$

$$= Part(x,y) \wedge \neg\exists \boldsymbol{u,v}(Supplier(u,v) \wedge \neg\exists p\ Supply(u,x,p))$$

```
SELECT a.pno, a.pname
FROM   Part a
WHERE NOT EXISTS
       (




                                    )
```

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

Simplified schema

Understanding Quantifiers

Find all parts supplied by all suppliers

$$Answer(x,y) =$$

$$= Part(x,y) \wedge \forall u,v(Supplier(u,v) \Rightarrow \exists p\ Supply(u,x,p))$$

$$= Part(x,y) \wedge \neg\exists u,v(\boldsymbol{Supplier(u,v)} \wedge \neg\exists \boldsymbol{p}\ Supply(u,x,p))$$

```
SELECT a.pno, a.pname
FROM    Part a
WHERE NOT EXISTS
        (SELECT *
          FROM Supplier c
          WHERE NOT EXISTS
             (
                                    ))
```

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

Simplified schema

Understanding Quantifiers

Find all parts supplied by all suppliers

$$Answer(x,y) =$$

$$= Part(x,y) \wedge \forall u,v(Supplier(u,v) \Rightarrow \exists p\ Supply(u,x,p))$$

$$= Part(x,y) \wedge \neg\exists u,v(Supplier(u,v) \wedge \neg\exists \boldsymbol{p\ Supply(u,x,p)})$$

```
SELECT a.pno, a.pname
FROM   Part a
WHERE NOT EXISTS
       (SELECT *
        FROM Supplier c
        WHERE NOT EXISTS
           (SELECT * FROM Supply b
            WHERE a.pno=b.pno and b.sno=c.sno))
```

Subqueries in WHERE

- EXISTS(….)
  check if empty
- NOT EXISTS(…)
  check if not empty

Subqueries in WHERE

- EXISTS(….)
  check if empty
- NOT EXISTS(…)
  check if not empty
- X IN (…)
  check if X in the set
- X NOT IN (…)
  check if X not in set

Subqueries in WHERE

- EXISTS(….)
  check if empty
- NOT EXISTS(…)
  check if not empty
- X IN (…)
  check if X in the set
- X NOT IN (…)
  check if X not in set
- X > SOME (…)
  ∃Y in (…) and X>Y
- X > ALL (…)
  ∀Y in (…): X > Y

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

```
SELECT a.pno, a.pname
FROM    Part a
WHERE NOT EXISTS
        (SELECT *
         FROM Supply b
         WHERE b.price >= 100
               and b.pno = a.pno)
```

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

```
SELECT a.pno, a.pname
FROM   Part a
WHERE NOT EXISTS
        (SELECT *
         FROM Supply b
         WHERE b.price >= 100
               and b.pno = a.pno)
```

```
SELECT a.pno, a.pname
FROM   Part a
WHERE a.pno NOT IN
        (SELECT b.pno
         FROM Supply b
         WHERE b.price >= 100)
```

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

```
SELECT a.pno, a.pname
FROM   Part a
WHERE NOT EXISTS
        (SELECT *
         FROM Supply b
         WHERE b.price >= 100
               and b.pno = a.pno)
```

```
SELECT a.pno, a.pname
FROM   Part a
WHERE a.pno NOT IN
        (SELECT b.pno
         FROM Supply b
         WHERE b.price >= 100)
```

If evaluated naively, which query is more efficient?

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

```
SELECT a.pno, a.pname
FROM   Part a
WHERE NOT EXISTS
        (SELECT *
         FROM Supply b
         WHERE b.price >= 100
               and b.pno = a.pno)
```

Correlated subquery

```
SELECT a.pno, a.pname
FROM   Part a
WHERE a.pno NOT IN
        (SELECT b.pno
         FROM Supply b
         WHERE b.price >= 100)
```

If evaluated naively, which query is more efficient?

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

```
SELECT a.pno, a.pname
FROM   Part a
WHERE NOT EXISTS
        (SELECT *
         FROM Supply b
         WHERE b.price >= 100
               and b.pno = a.pno)
```

Correlated subquery

```
SELECT a.pno, a.pname
FROM   Part a
WHERE a.pno NOT IN
        (SELECT b.pno
         FROM Supply b
         WHERE b.price >= 100)
```

Uncorrelated

If evaluated naively, which query is more efficient?

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

```
SELECT a.pno, a.pname
FROM   Part a
WHERE NOT EXISTS
        (SELECT *
         FROM Supply b
         WHERE b.price >= 100
               and b.pno = a.pno)
```

```
SELECT a.pno, a.pname
FROM   Part a
WHERE a.pno NOT IN
        (SELECT b.pno
         FROM Supply b
         WHERE b.price >= 100)
```

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

Simplified schema

Subqueries in WHERE

Find all parts where all supplier offering them charge < $100

```
SELECT a.pno, a.pname
FROM   Part a
WHERE NOT EXISTS
        (SELECT *
         FROM Supply b
         WHERE b.price >= 100
               and b.pno = a.pno)
```

```
SELECT a.pno, a.pname
FROM   Part a
WHERE a.pno NOT IN
        (SELECT b.pno
         FROM Supply b
         WHERE b.price >= 100)
```

```
SELECT a.pno, a.pname
FROM   Part a
WHERE 100 < ALL
        (SELECT b.price
         FROM Supply b
         WHERE b.pno = a.pno)
```

Discussion

- Queries w/ existential quantifiers can be unnested into SELECT-FROM-WHERE

- Queries w/ universal quantifier cannot

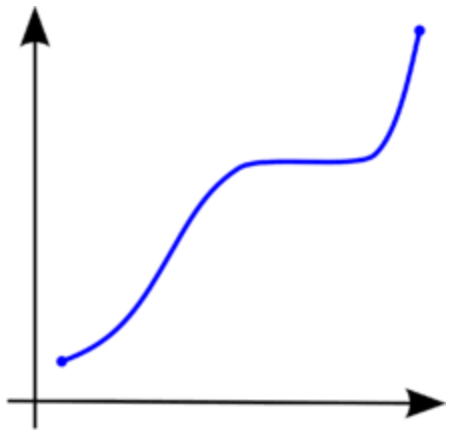
We will prove this next

Monotone Queries

Montone Functions

A function $f: R \rightarrow R$ is **monotone** if $x \leq y$ implies $f(x) \leq f(y)$

Monotone:
$x^3 + x^2$,
e^x,
$\log(x)$,
…



Non-Monotone:
$x^3 - x^2$,
e^{-x},
$\frac{1}{x}$,
…

Monotone Queries

A query Q is monotone if $I \subseteq J$ implies $q(I) \subseteq q(J)$

Monotone Queries

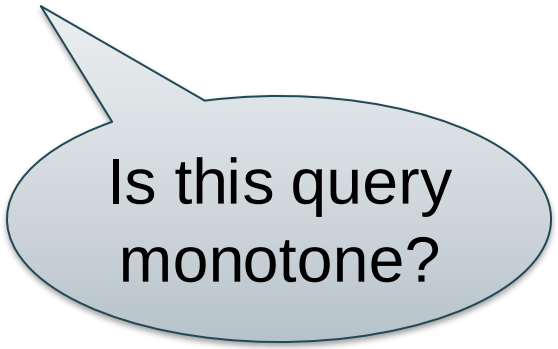
A query Q is monotone if $I \subseteq J$ implies $q(I) \subseteq q(J)$

Adding tuples to the input does
not remove tuples from the output

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

Monotone Queries

Find all parts that have some supplier offering them for < $100



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

Monotone Queries

Find all parts that have some supplier offering them for < $100

Supply

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

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

Monotone Queries

Find all parts that have some supplier offering them for < $100

Supply

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone

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

Monotone Queries

Find all parts that have some supplier offering them for < $100

Supply

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300
s03	p200	99

pno	pname
p100	phone
p200	mouse
p300	lamp

J

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

Monotone Queries

Find all parts that have some supplier offering them for < $100

Supply

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300
s03	p200	99

pno	pname
p100	phone
p200	mouse
p300	lamp

J

pno	pname
p100	phone
p200	mouse

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

Monotone

Monotone Queries

Find all parts that have some supplier offering them for < $100

Supply

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300
s03	p200	99

pno	pname
p100	phone
p200	mouse
p300	lamp

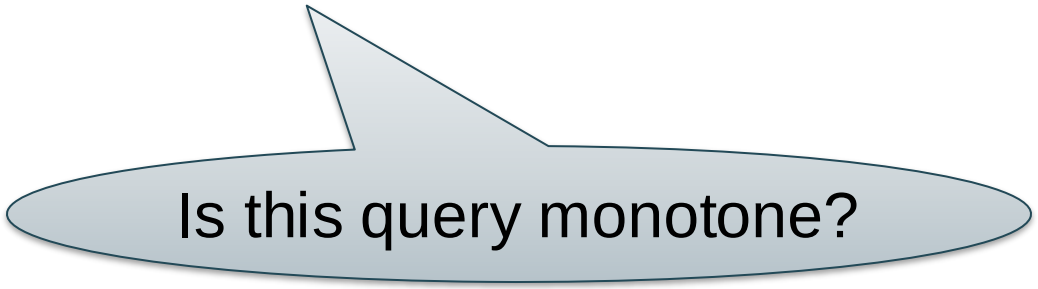
J

pno	pname
p100	phone
p200	mouse

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

Monotone Queries

Find all parts where all supplier offering them charge < $100



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

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

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

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone
p200	mouse
p300	lamp

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

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone
p200	mouse
p300	lamp

Why?

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

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone
p200	mouse
p300	lamp

Why?

$$Part(x, y) \wedge \forall z, p(Supply(z, x, p) \Rightarrow p < 100)$$

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

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone
p200	mouse
p300	lamp

Why?

$$Part(x, y) \wedge \forall z, p(Supply(z, x, p) \Rightarrow p < 100)$$

If $Supply(z, x, p)$ is FALSE, then $Supply(z, x, p) \Rightarrow p < 100$ is TRUE

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

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone
p200	mouse
p300	lamp

Why?

$$Part(x, y) \wedge \forall z, p(Supply(z, x, p) \Rightarrow p < 100)$$

If $Supply(z, x, p)$ is FALSE, then $Supply(z, x, p) \Rightarrow p < 100$ is TRUE

Hence (p200,mouse) is in the output

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

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone
p200	mouse
p300	lamp

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

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone
p200	mouse
p300	lamp

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300
s03	p100	199

pno	pname
p100	phone
p200	mouse
p300	lamp

J

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

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone
p200	mouse
p300	lamp

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300
s03	p100	199

pno	pname
p100	phone
p200	mouse
p300	lamp

J

pno	pname
~~p100~~	~~phone~~
p200	mouse
p300	lamp

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

Non-Monotone

Monotone Queries

Find all parts where all supplier offering them charge < $100

Supply

sno	pno	price
s01	p100	20
s02	p100	50
s01	p300	30

Part

pno	pname
p100	phone
p200	mouse
p300	lamp

I

pno	pname
p100	phone
p200	mouse
p300	lamp

sno	pno	price
s01	p100	200
s02	p100	50
s01	p200	300
s03	p100	199

pno	pname
p100	phone
p200	mouse
p300	lamp

J

pno	pname
~~p100~~	~~phone~~
p200	mouse
p300	lamp

A Theorem

Every SELECT-FROM-WHERE query without subqueries and without aggregates is monotone

A Theorem

Every SELECT-FROM-WHERE query without subqueries and without aggregates is monotone

Proof. Consider a SQL query:

```
SELECT attrs
FROM T1, T2, ...
WHERE condition
```

A Theorem

Every SELECT-FROM-WHERE query without subqueries and without aggregates is monotone

Proof. Consider a SQL query:

```
SELECT attrs
FROM T1, T2, ...
WHERE condition
```

Its nested loop semantics is:

```
for each r1 in T1:
 for each t2 in T2:
   for each t3 in T3:
       …
       if (condition):
         output (a1,a2,…)
```

A Theorem

Every SELECT-FROM-WHERE query without subqueries and without aggregates is monotone

Proof. Consider a SQL query:

SELECT attrs

FROM T1, T2, ...

WHERE condition

Its nested loop semantics is:

```
for each r1 in T1:
 for each t2 in T2:
   for each t3 in T3:
       …
       if (condition):
          output (a1,a2,…)
```

If we insert a tuple into one of the input relations T_i, we will not remove any tuples from the output.

An Application

The query:

Find all parts where all supplier offering them charge < $100

Cannot be unnested without using aggregates

Finding Witnesses a.k.a. ARGMAX

The Witness aka ARGMAX

- Find the city with the largest population
- Find product/products with largest price
- …
- SQL does not have ARGMAX
- Two solutions:
  - Use intermediate relation in WITH
  - Self-join and HAVING

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

Using WITH

For each supplier, find the most expensive product they supply

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

Using WITH

For each supplier, find the most expensive product they supply

Finding the max price is easy:

```
SELECT x.sno, x.name, max(y.price)
FROM Supplier x, Supply y
WHERE x.sno = y.sno
GROUP BY x.sno, x.name
```

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

Using WITH

For each supplier, find the most expensive product they supply

Finding the max price is easy:

But we also want pno, pname:

Why does this not work?

```
SELECT x.sno, y.pno, x.name, max(y.price)
FROM Supplier x, Supply y
WHERE x.sno = y.sno
GROUP BY x.sno, x.name
```

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

Using WITH

For each supplier, find the most expensive product they supply

Compute max price in temporary table

```
WITH Temp AS
  (SELECT x.sno, x.sname, max(y.price) as m
   FROM Supplier x, Supply y
   WHERE x.sno = y.sno
   GROUP BY x.sno, x.name)
. . . . . .
```

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

Using WITH

For each supplier, find the most expensive product they supply

Compute max price in temporary table

```
WITH Temp AS
  (SELECT x.sno, x.sname, max(y.price) as m
   FROM Supplier x, Supply y
   WHERE x.sno = y.sno
   GROUP BY x.sno, x.name)

SELECT t.sno, t.sname, v.pno, v.pname
FROM Temp t, Supply u, Part v
WHERE t.sno = u.sno
      and u.pno = v.pno
      and t.m = u.price;
```

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

Using Self-joins and HAVING

For each supplier, find the most expensive product they supply

Join directly the temp table with Supply and Part

```
SELECT x.sno, x.sname, v.pno, v.pname
FROM Supplier x, Supply y, Supply u, Part v
WHERE x.sno = y.sno
      and x.sno = u.sno
      and u.pno = v.pno
HAVING max(y.price) = v.price
GROUP BY x.sno, x.sname, v.pno, v.pname
```

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

Using Self-joins and HAVING

For each supplier, find the most expensive product they supply

Join directly the temp table with Supply and Part

```
SELECT x.sno, x.sname, v.pno, v.pname
FROM Supplier x, Supply y, Supply u, Part v
WHERE x.sno = y.sno
      and x.sno = u.sno
      and u.pno = v.pno
HAVING max(y.price) = v.price
GROUP BY x.sno, x.sname, v.pno, v.pname
```

```
WITH Temp AS
  (SELECT x.sno, x.sname,
                max(y.price) as m
   FROM Supplier x, Supply y
   WHERE x.sno = y.sno
   GROUP BY x.sno, x.name)

SELECT t.sno, t.sname, v.pno, v.pname
FROM Temp t, Supply u, Part v
WHERE t.sno = u.sno
      and u.pno = v.pno
      and t.m = u.price;
```

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

Using Self-joins and HAVING

For each supplier, find the most expensive product they supply

Join directly the temp table with Supply and Part

```
SELECT x.sno, x.sname, v.pno, v.pname
FROM Supplier x, Supply y, Supply u, Part v
WHERE x.sno = y.sno
      and x.sno = u.sno
      and u.pno = v.pno
HAVING max(y.price) = v.price
GROUP BY x.sno, x.sname, v.pno, v.pname
```

```
WITH Temp AS
  (SELECT x.sno, x.sname,
                 max(y.price) as m
   FROM Supplier x, Supply y
   WHERE x.sno = y.sno
   GROUP BY x.sno, x.name)

SELECT t.sno, t.sname, v.pno, v.pname
FROM Temp t, Supply u, Part v
WHERE t.sno = u.sno
      and u.pno = v.pno
      and t.m = u.price;
```

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

Using Self-joins and HAVING

For each supplier, find the most expensive product they supply

Join directly the temp table with Supply and Part

```
SELECT x.sno, x.sname, v.pno, v.pname
FROM Supplier x, Supply y, Supply u, Part v
WHERE x.sno = y.sno
      and x.sno = u.sno
      and u.pno = v.pno
HAVING max(y.price) = v.price
GROUP BY x.sno, x.sname, v.pno, v.pname
```

```
WITH Temp AS
  (SELECT x.sno, x.sname,
                max(y.price) as m
   FROM Supplier x, Supply y
   WHERE x.sno = y.sno
   GROUP BY x.sno, x.name)

SELECT t.sno, t.sname, v.pno, v.pname
FROM Temp t, Supply u, Part v
WHERE t.sno = u.sno
      and u.pno = v.pno
      and t.m = u.price;
```

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

Using Self-joins and HAVING

For each supplier, find the most expensive product they supply

Join directly the temp table with Supply and Part

```
SELECT x.sno, x.sname, v.pno, v.pname
FROM Supplier x, Supply y, Supply u, Part v
WHERE x.sno = y.sno
      and x.sno = u.sno
      and u.pno = v.pno
HAVING max(y.price) = v.price
GROUP BY x.sno, x.sname, v.pno, v.pname
```

```
WITH Temp AS
  (SELECT x.sno, x.sname,
          max(y.price) as m
   FROM Supplier x, Supply y
   WHERE x.sno = y.sno
   GROUP BY x.sno, x.name)

SELECT t.sno, t.sname, v.pno, v.pname
FROM Temp t, Supply u, Part v
WHERE t.sno = u.sno
      and u.pno = v.pno
      and t.m = u.price;
```

Implicit DISTINCT

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

Using Self-joins and HAVING

For each supplier, find the most expensive product they supply

Join directly the temp table with Supply and Part

```
SELECT x.sno, x.sname, v.pno, v.pname
FROM Supplier x, Supply y, Supply u, Part v
WHERE x.sno = y.sno
      and x.sno = u.sno
      and u.pno = v.pno
HAVING max(y.price) = v.price
GROUP BY x.sno, x.sname, v.pno, v.pname
```

```
WITH Temp AS
  (SELECT x.sno, x.sname,
                max(y.price) as m
   FROM Supplier x, Supply y
   WHERE x.sno = y.sno
   GROUP BY x.sno, x.name)

SELECT t.sno, t.sname, v.pno, v.pname
FROM Temp t, Supply u, Part v
WHERE t.sno = u.sno
      and u.pno = v.pno
      and t.m = u.price;
```

Discussion

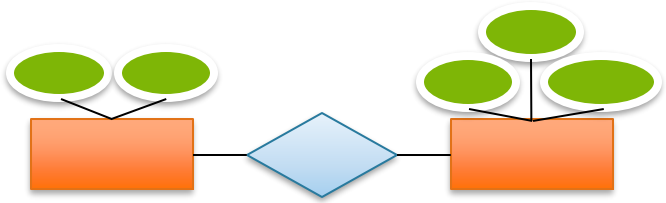
- Why doesn't SQL have ARGMAX?
  [in class]
- Solution 1: use temp table
- Solution 2: self-join + HAVING
- Solution 3: NOT EXISTS
- … [perhaps more]

Final Words on SQL

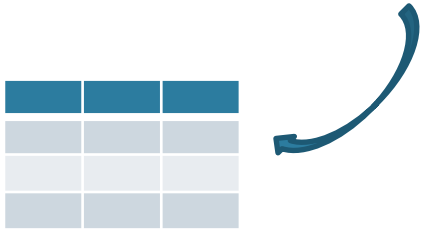
- In this class we only use the fragment discussed so far
  - We will add WITH RECURSIVE next week
- Look up scalar function as needed:
  - Substring operations, math functions, etc
- We don't discuss, and don't use in HW: WINDOW function, GROUP SETs, etc

Database Design

Conceptual Model

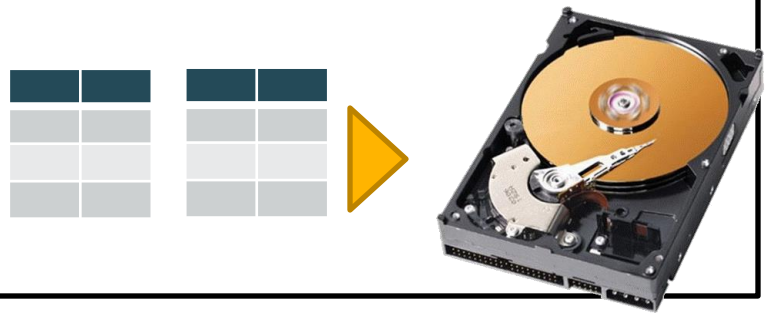


Relational Model
+ Schema
+ Constraints

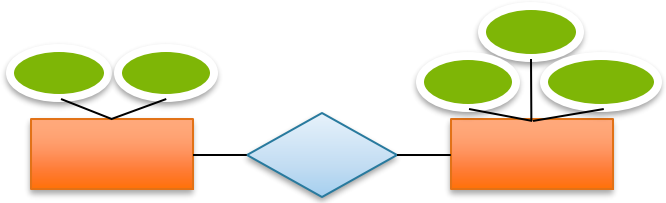


Conceptual Schema
+ Normalization

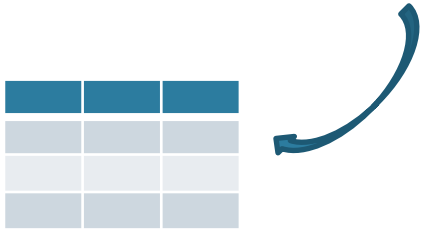
Physical Schema
+ Partitioning
+ Indexing



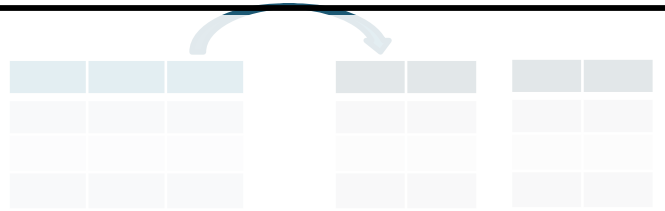
Conceptual Model



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 Diagrams

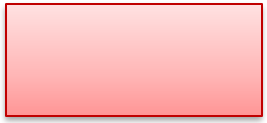
Design a schema for a database of doctors and their patients

E/R Diagrams

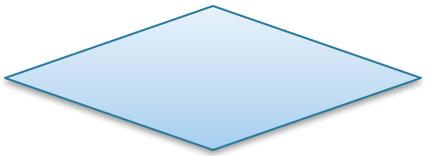
Design a schema for a database of doctors and their patients

Attributes

Entity sets



Relationship sets



E/R Diagrams

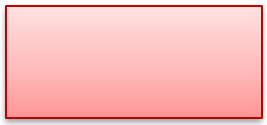
Design a schema for a database of doctors and their patients

Patient

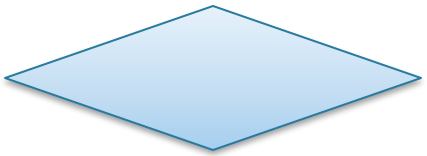
Doctor

Attributes

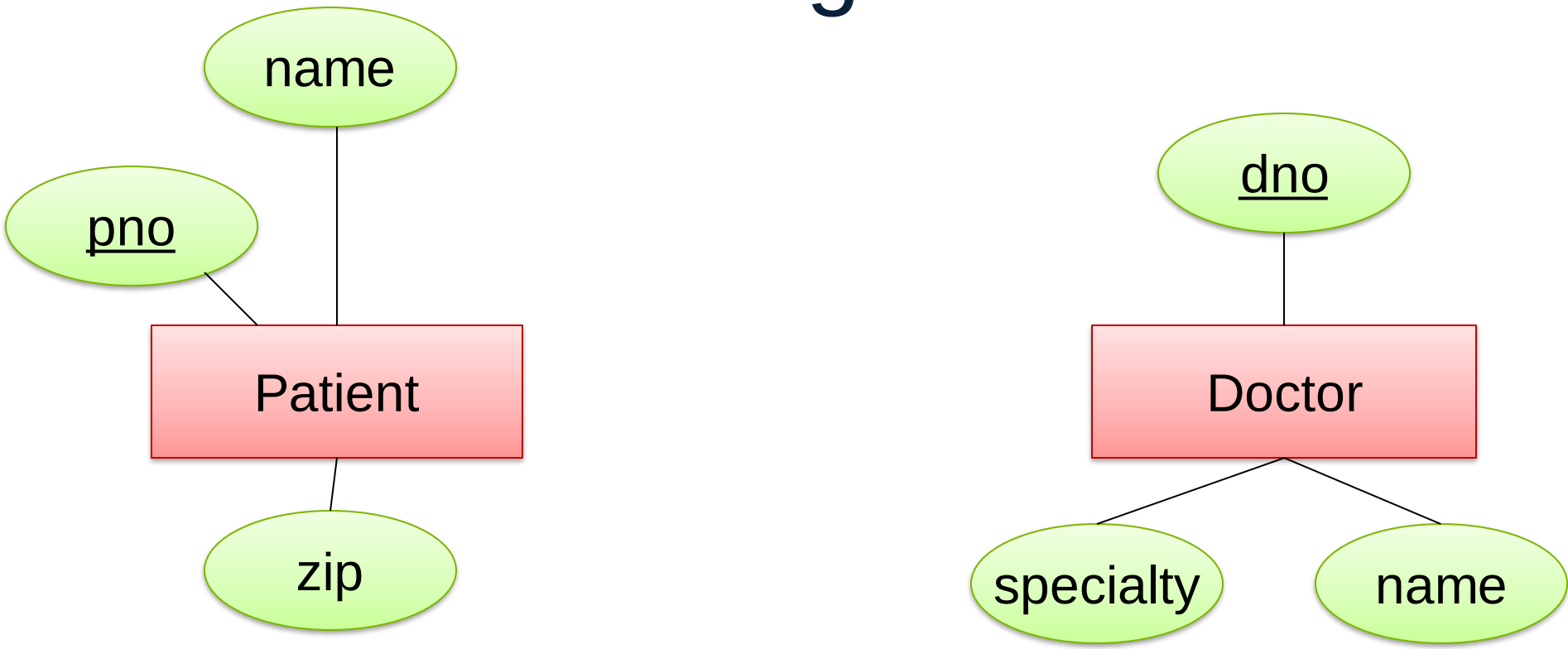
Entity sets



Relationship sets



E/R Diagrams

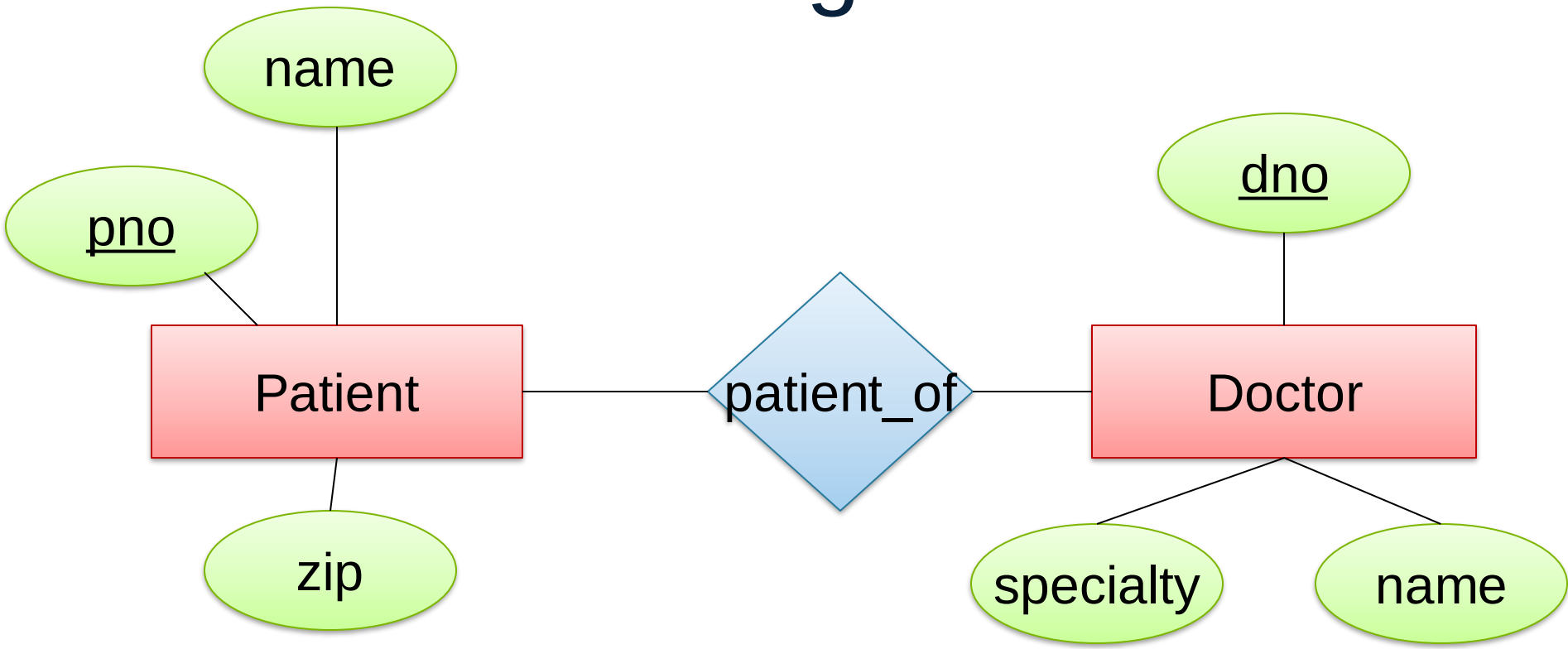


Attributes

Entity sets

Relationship sets

E/R Diagrams

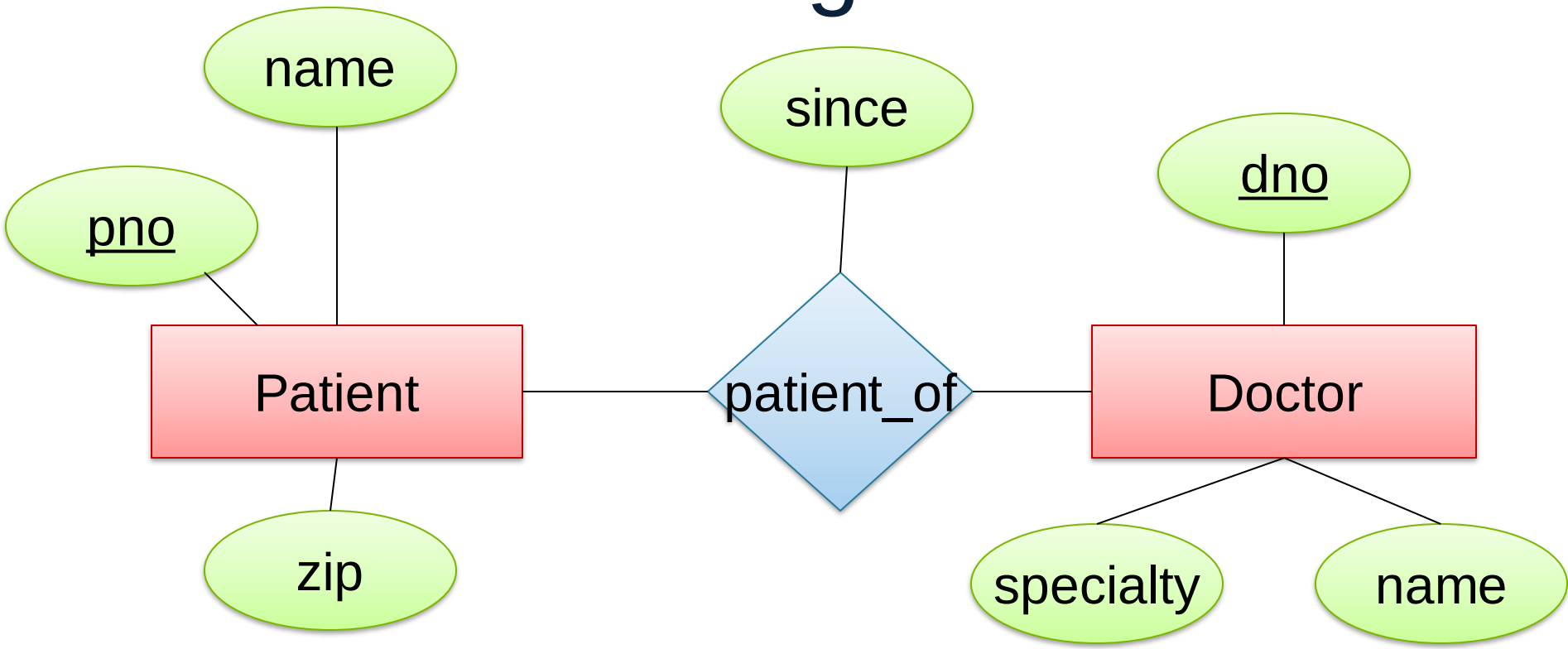


Attributes

Entity sets

Relationship sets

E/R Diagrams



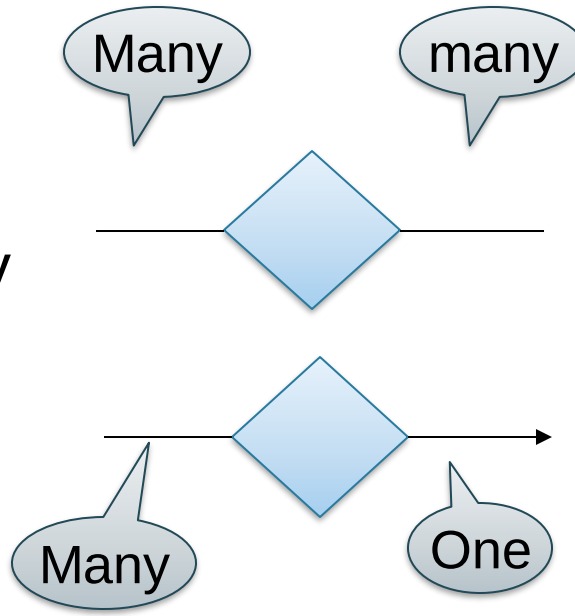
Attributes

Entity sets

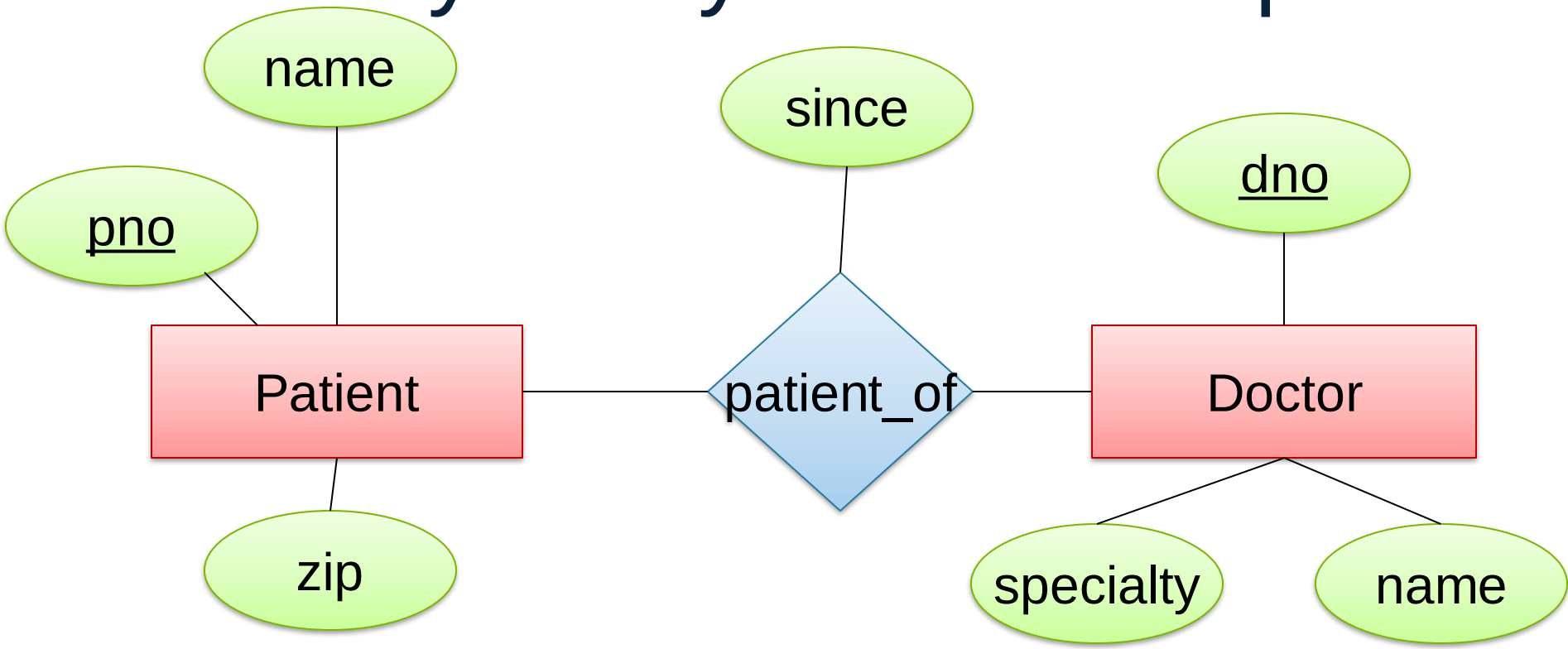
Relationship sets

E/R Diagrams

- Each entity set has a key
- ER relationships can include multiplicity
  - One-to-one, one-to-many, etc.
  - Indicated with arrows
- Can model multi-way relationships
- Can model subclasses
- And more...



Many-many Relationship



Patient

pno	name	zip
P311	Alice	98765
…		

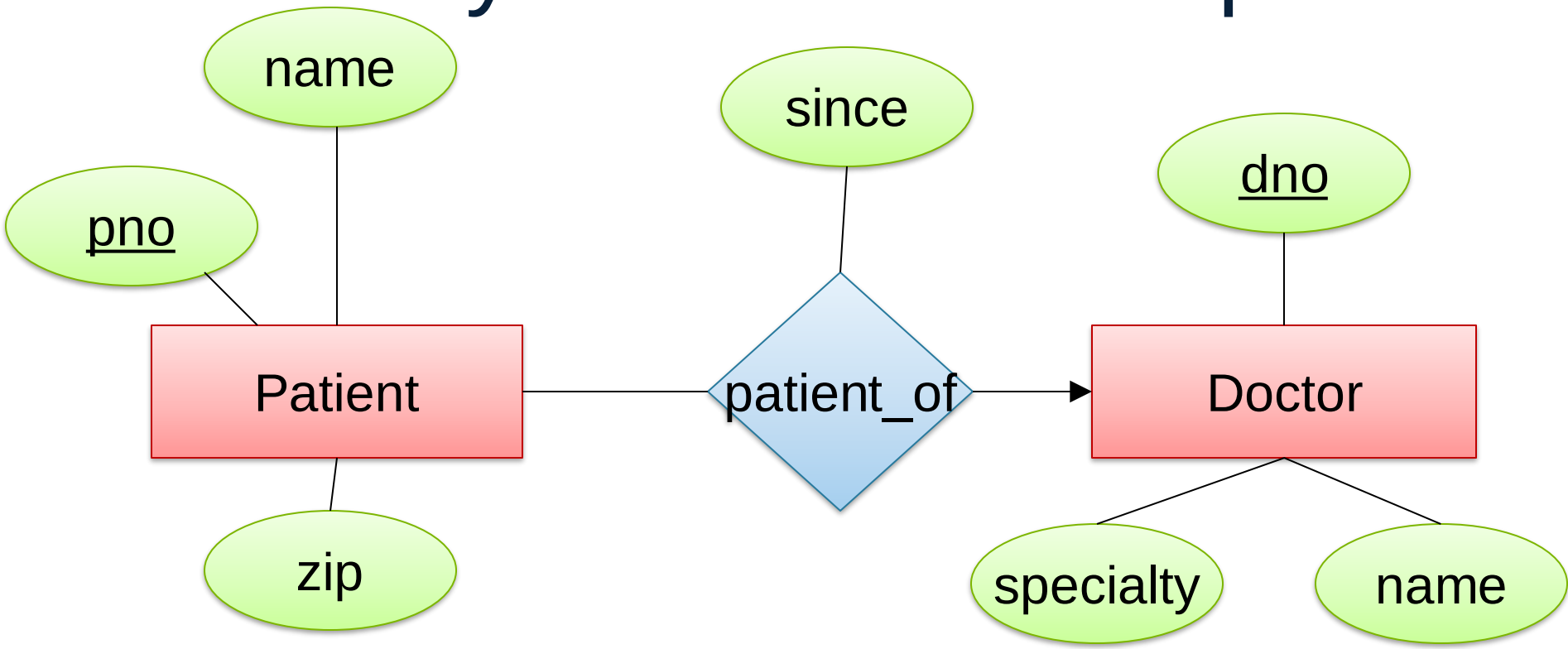
Patient_of

pno	dno	since
P311	D007	2001
…		

Doctor

dno	name	spec
D007	Bob	cardio
…		

Many-one Relationship



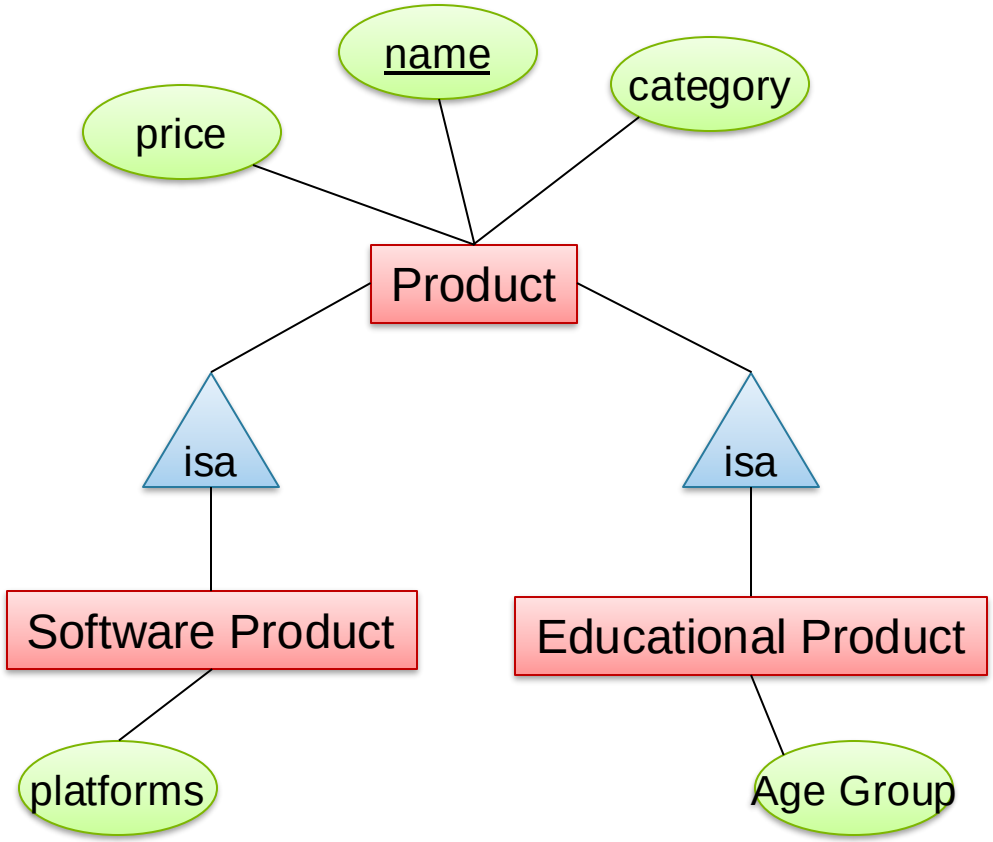
Patient

pno	name	zip	dno	since
P311	Alice	98765	D007	2001
…				

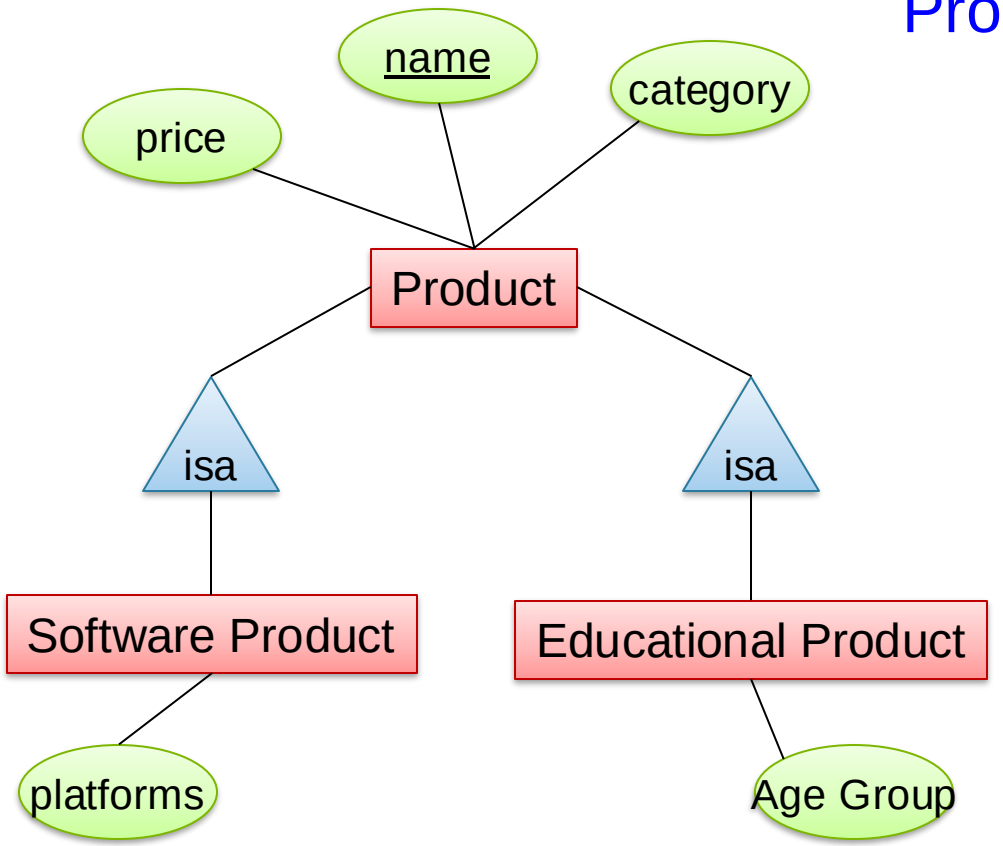
Doctor

dno	name	spec
D007	Bob	cardio
…		

Subclasses


name
price
category
Product
isa
isa
Software Product
Educational Product
platforms
Age Group

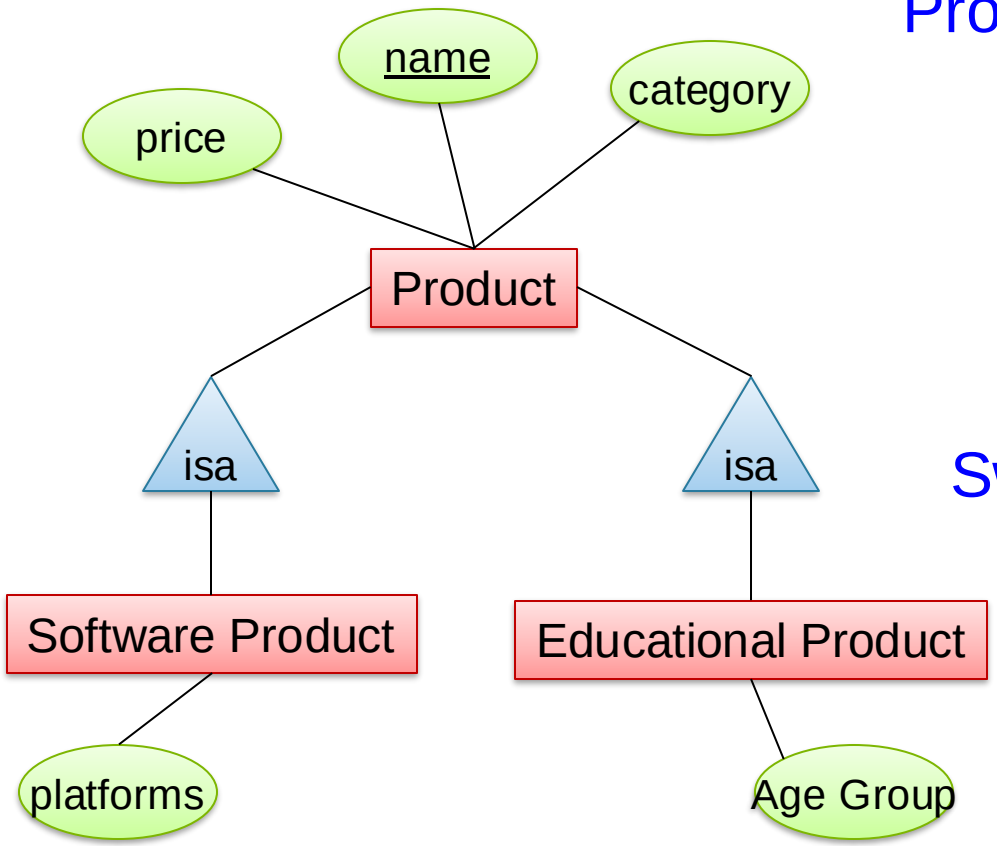
Subclasses



Product

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

Subclasses



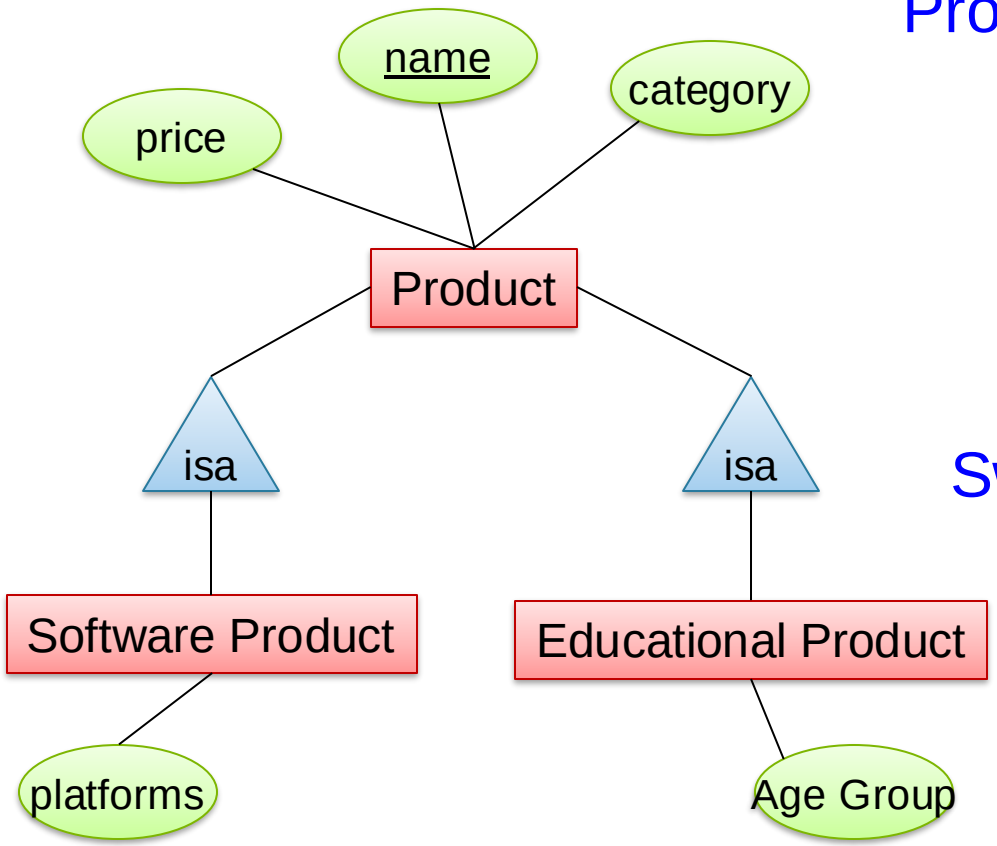
Product

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

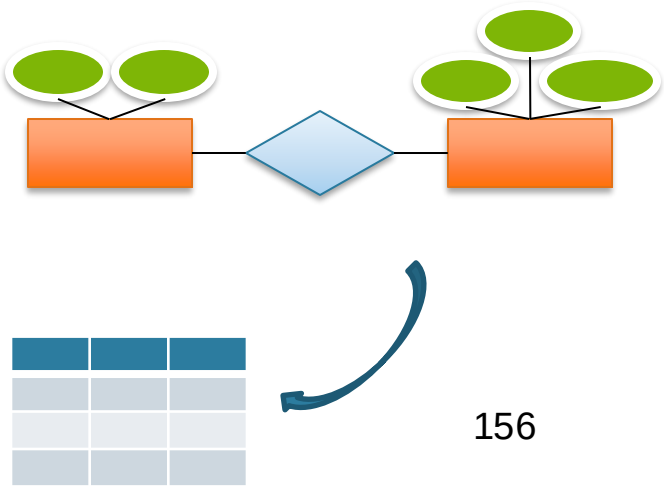
Name	platforms
Gizmo	unix

Ed.Product

Name	Age Group
Gizmo	toddler
Toy	senior

Converting E/R to SQL

- 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 and expression A→B, which means that the values in the A column uniquely determine those in the B column

FDs and Normal Forms

- A functional dependency is and expression A→B, which means that the values in the A column uniquely determine those in the B column

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 and expression A→B, which means that the values in the A column uniquely determine those in the B column

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

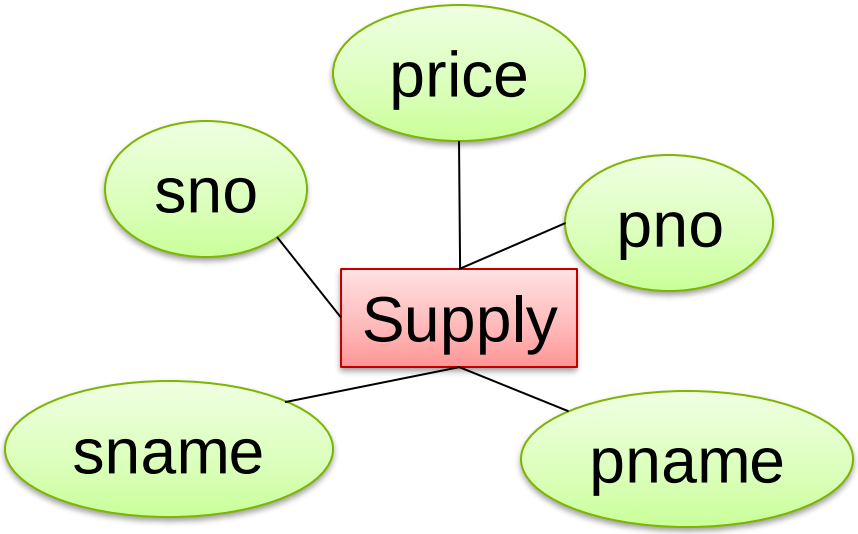
- A super-key is a set of attributes X such that, for every attribute A: $X \rightarrow A$
- A key is a minimal super-key
- A relation is in BCNF if, for every non-trivial FD $X \rightarrow A$, X is a super-key

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

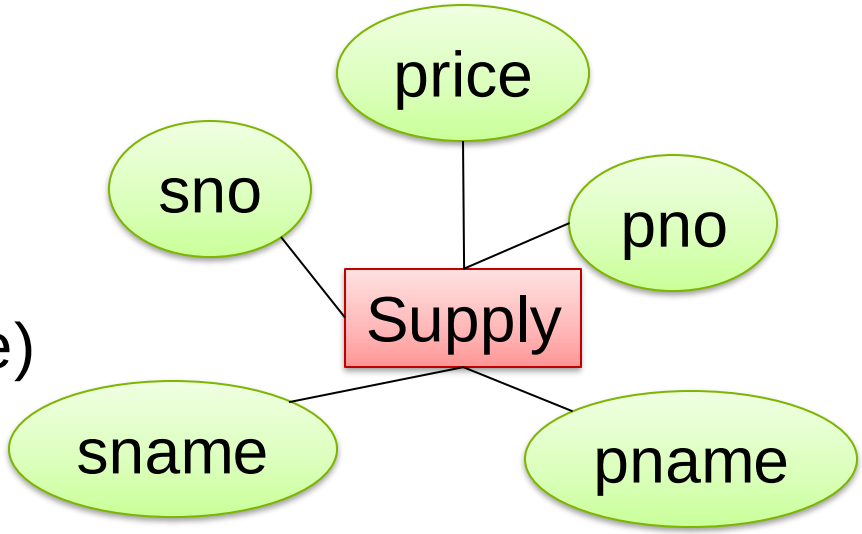
sno→sname
pno→pname



Example

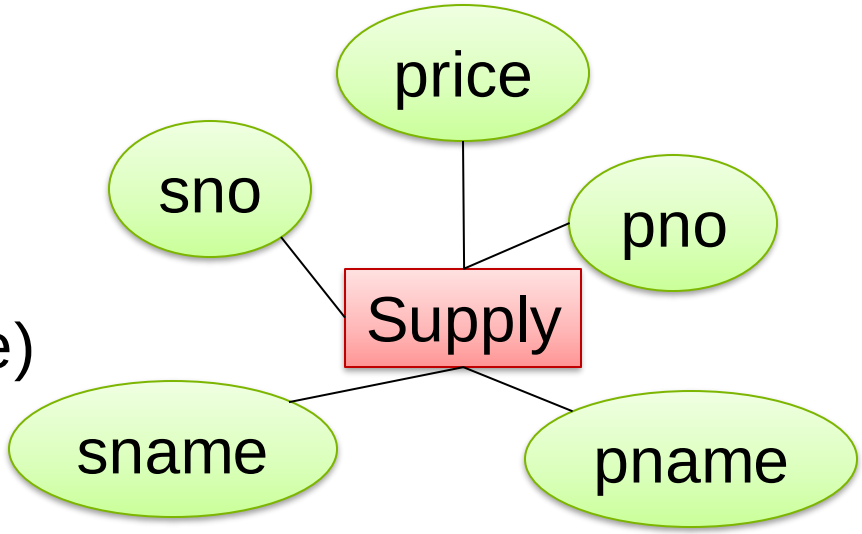
sno→sname
pno→pname

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



Example

sno→sname
pno→pname

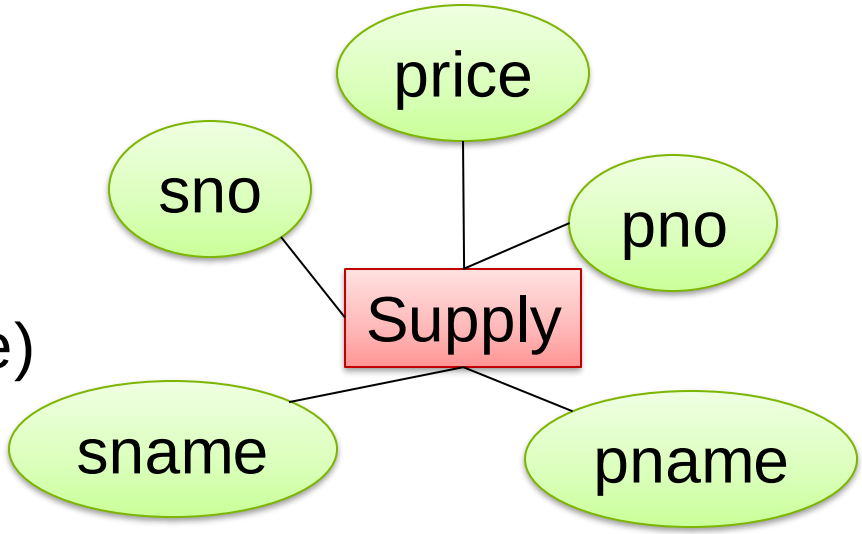


Supply(sno,sname,price,pno,pname)

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

Example

sno→sname
pno→pname



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

T1(<u>sno</u>,sname)

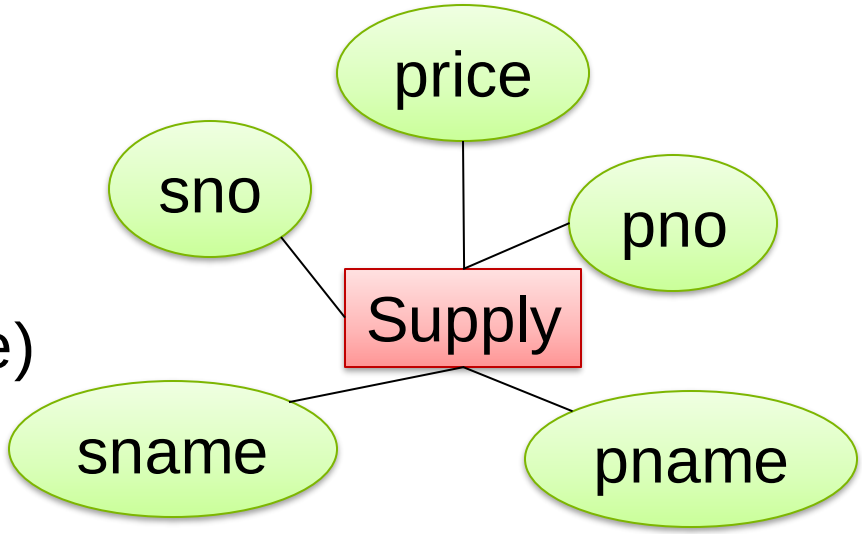
T2(sno,price,pno,pname)

T3(<u>pno</u>,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

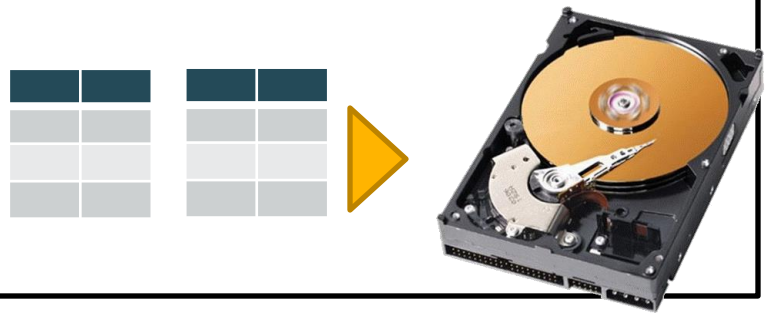
- BCNF avoids “data anomalies”
- Check details on data anomalies, FDs, Armstrong Axioms, and the BCNF here
- You shouldn’t need this for HW1

Conceptual Model

Relational Model
+ Schema
+ Constraints

Conceptual Schema
+ Normalization

Physical Schema
+ Partitioning
+ Indexing



Indexes

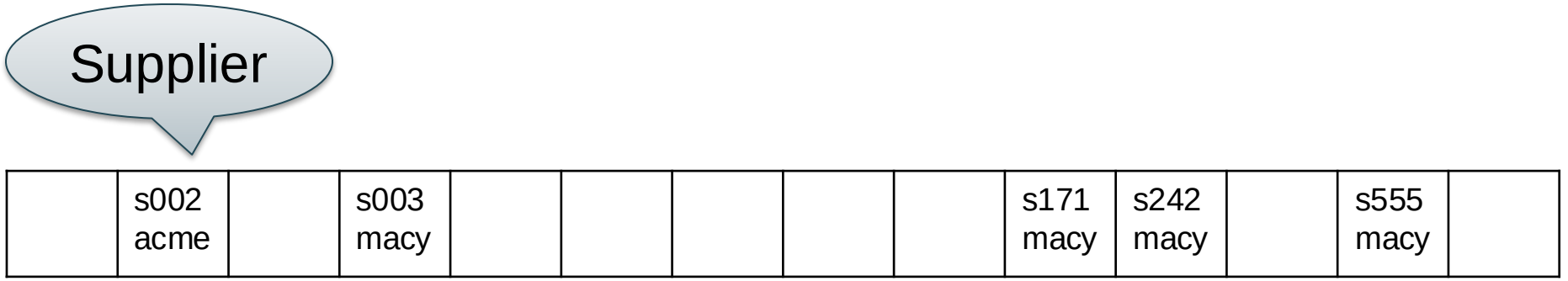
- An index is an auxiliary file that allows direct access to the data file based on the value of an attribute

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

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

Indexes

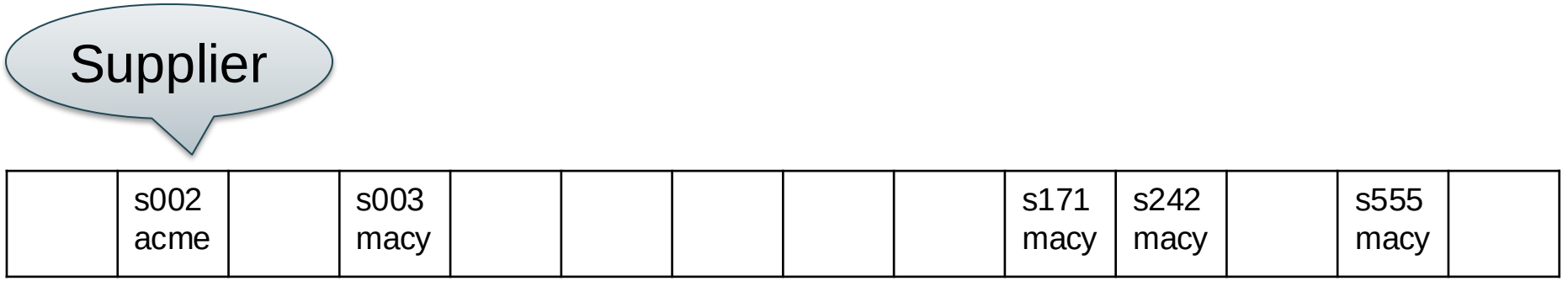
```
CREATE TABLE Supplier(sno int primary key, sname text);
```



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

Indexes

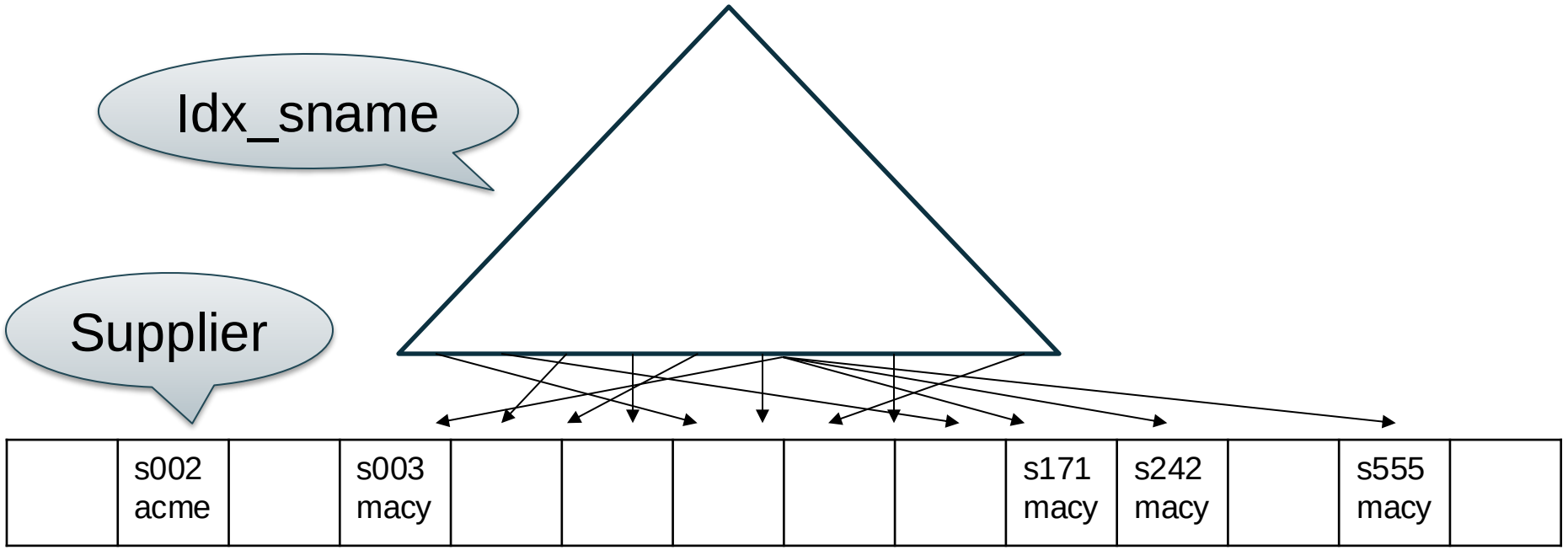
```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname)
```



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

Indexes

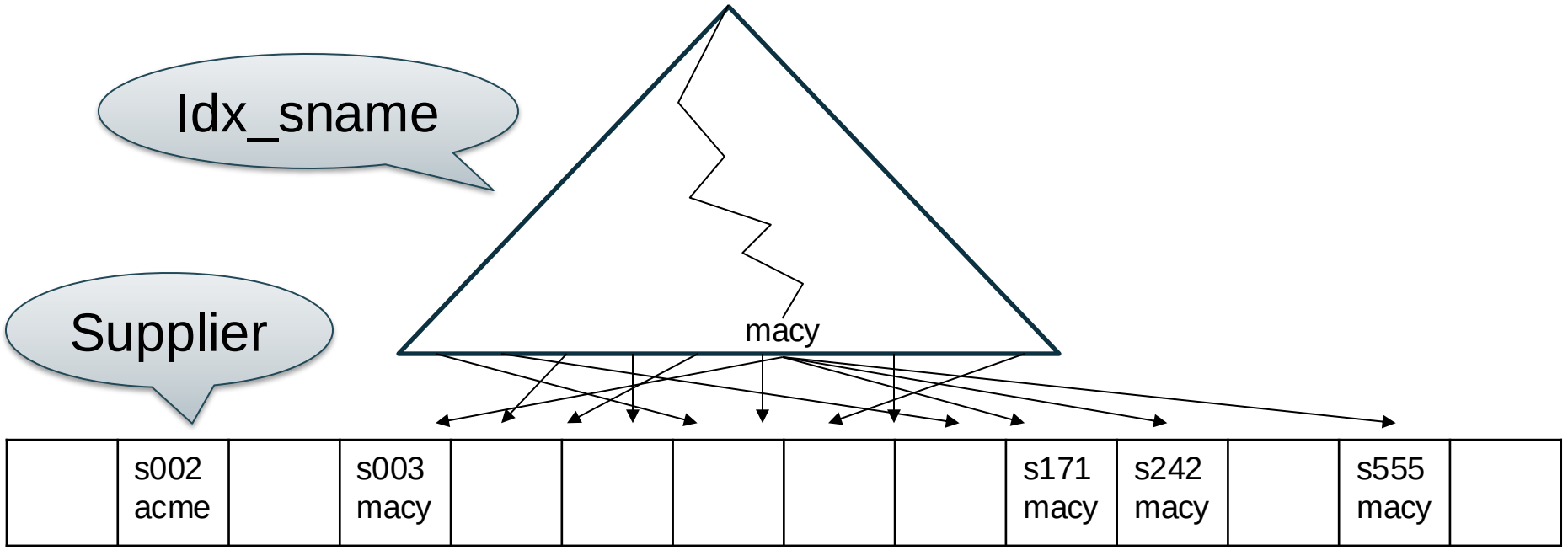
```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname)
```



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

Indexes

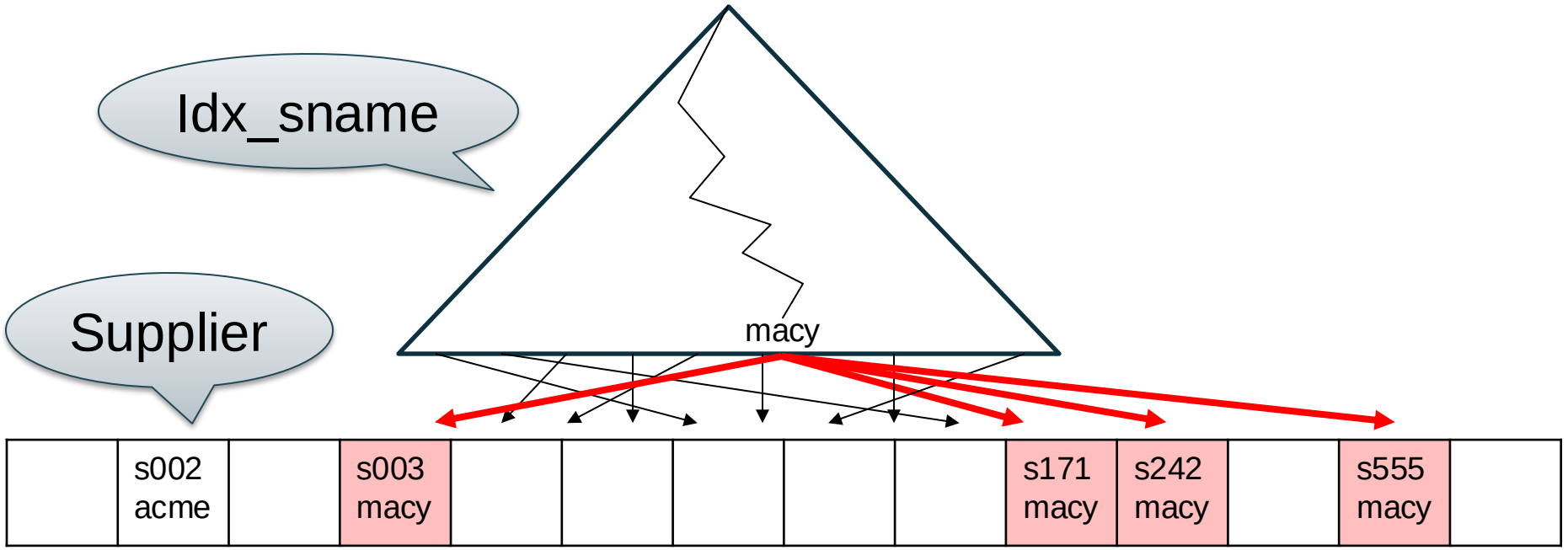
```
CREATE TABLE Supplier(sno int primary key, sname text);
CREATE INDEX Idx_sname on Supplier(sname)
```



```
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)
```



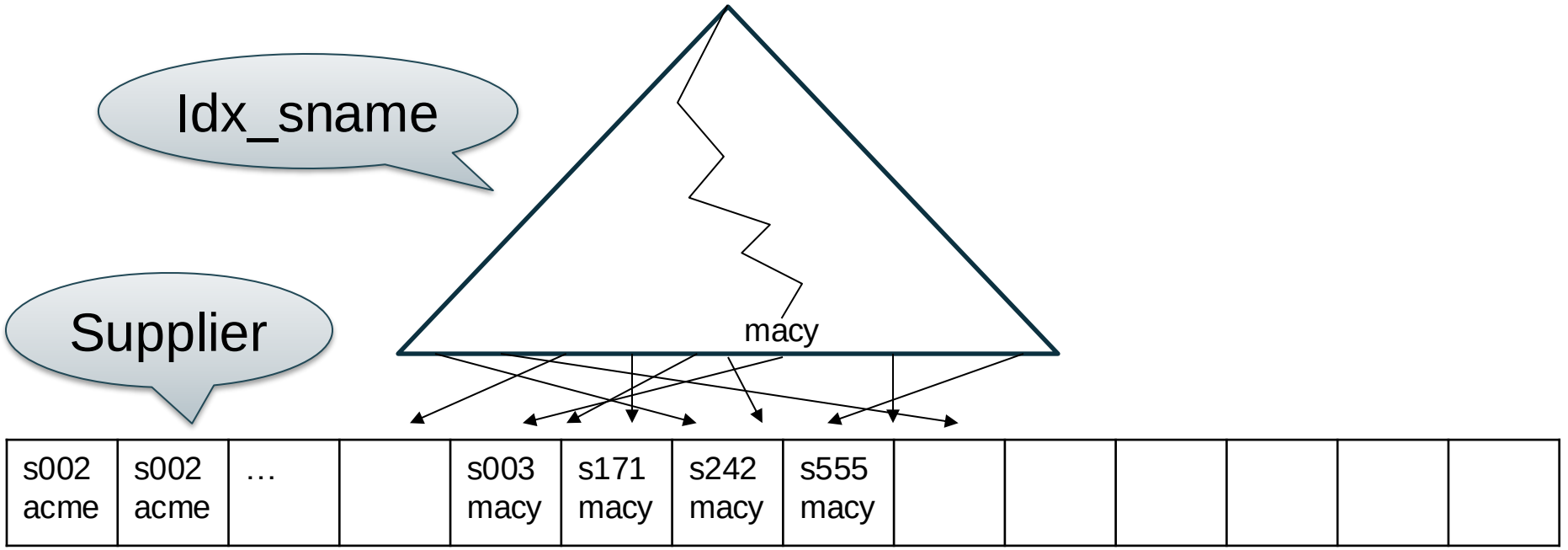
Indexes

- 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)

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