CS 6400 A

Midterm Review

Lecture 9

09/17/25

Midterm Logistics

- Midterm will be held Monday Sep 22 from 5:00pm - 6:15pm (during class time).
- **Please arrive early -** the exam is going to start at 5:00pm.
- Open notes, no electronic devices. Can bring calculator.
- Contents covered: Lec 2 (SQL I) – Lec 7 (Design Theory II)
- Past Exam: available on canvas, under Files->Past Exams

SQL

SQL Query

- Basic form (there are many many more bells and whistles)

```
SELECT <attributes>
FROM   <one or more relations>
WHERE  <conditions>
```

Call this a **<u>SFW</u>** query.

LIKE: Simple String Pattern Matching

```
SELECT *
FROM   Products
WHERE  PName LIKE '%gizmo%'
```

DISTINCT: Eliminating Duplicates

```
SELECT DISTINCT Category
FROM   Product
```

ORDER BY: Sorting the Results (ascending by default)

```
SELECT   PName, Price
FROM     Product
WHERE    Category='gizmo'
ORDER BY Price, Pname
LIMIT    10;
```

Joins

Product

PName	Price	Category	Manuf
Gizmo	$19	Gadgets	GWorks
Powergizmo	$29	Gadgets	GWorks
SingleTouch	$149	Photography	Canon
MultiTouch	$203	Household	Hitachi

Company

Cname	Stock	Country
GWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
         AND Country='Japan'
     AND Price <= 200
```

PName	Price
SingleTouch	$149

An example of SQL semantics

```sql
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

A
1
3

B	C
2	3
3	4
3	5

Cross Product

A	B	C
1	2	3
1	3	4
1	3	5
3	2	3
3	3	4
3	3	5

Apply Selections / Conditions

A	B	C
3	3	4
3	3	5

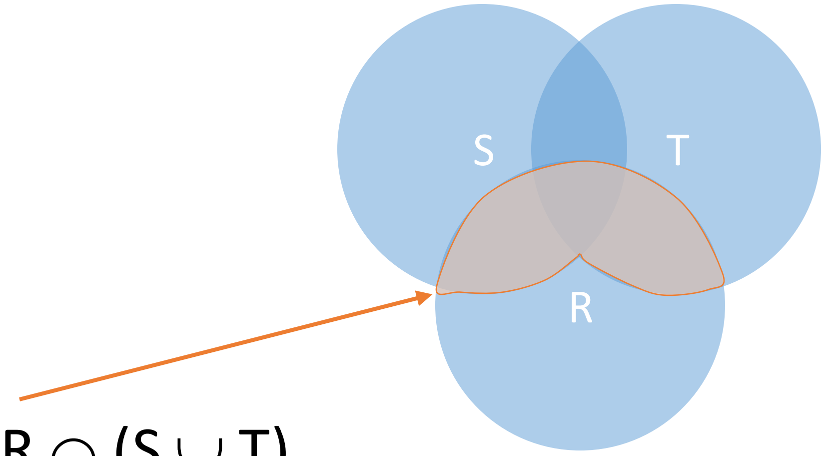
Apply Projection

A
3
3

Output

An Unintuitive Query

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```



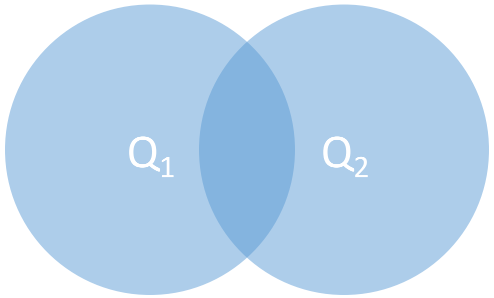
Computes R ∩ (S ∪ T)

But what if S = ϕ?

Go back to the semantics!

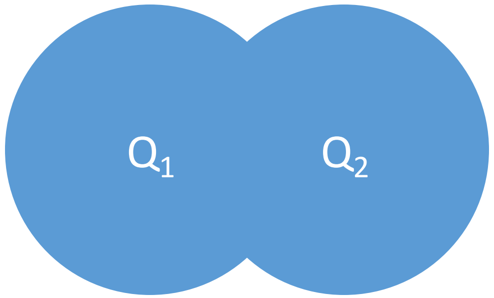
INTERSECT

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
INTERSECT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```



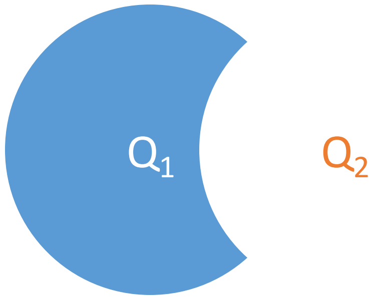
UNION

```
SELECT  R.A
FROM    R, S
WHERE   R.A=S.A
UNION
SELECT R.A
FROM    R, T
WHERE   R.A=T.A
```



EXCEPT

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
EXCEPT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```



Nested queries: Sub-queries Returning Relations

Company(<u>name</u>, city)
Product(<u>name</u>, maker)
Purchase(<u>id</u>, product, buyer)

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
          SELECT pr.maker
    FROM   Purchase p, Product pr
    WHERE  p.product = pr.name
           AND  p.buyer = 'Joe Blow')
```

"Cities where one can find companies that manufacture products bought by Joe Blow"

Nested Queries

Are these queries equivalent?

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
    SELECT pr.maker
    FROM   Purchase p, Product pr
    WHERE  p.name = pr.product
      AND p.buyer = 'Joe Blow')
```

```
SELECT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
  AND  pr.name = p.product
  AND  p.buyer = 'Joe Blow'
```

Beware of duplicates!

Nested Queries: Operator Semantics

Product(name, price, category, maker)

ALL

```
SELECT name
FROM   Product
WHERE  price > ALL(X)
```

Price must be > all entries in multiset X

ANY

```
SELECT name
FROM   Product
WHERE  price > ANY(X)
```

Price must be > at least one entry in multiset X

EXISTS

```
SELECT name
FROM   Product p1
WHERE  EXISTS (X)
```

X must be non-empty

*Note that p1 can be referenced in X (correlated query!)

Nested Queries: Operator Semantics

Product(name, price, category, maker)

ALL

```
SELECT name
FROM   Product
WHERE  price > ALL(
  SELECT price
  FROM   Product
  WHERE  maker = 'G')
```

Find products that are more expensive than *all products* produced by "G"

ANY

```
SELECT name
FROM   Product
WHERE  price > ANY(
  SELECT price
  FROM   Product
  WHERE  maker = 'G')
```

Find products that are more expensive than *any one product* produced by "G"

EXISTS

```
SELECT name
FROM   Product p1
WHERE  EXISTS (
  SELECT *
  FROM   Product p2
  WHERE  p2.maker = 'G'
    AND  p1.price =
           p2.price)
```

Find products where there *exists some product* with the same price produced by "G"

Correlated Queries

Movie(title, year, director, length)

```
SELECT DISTINCT title
FROM   Movie AS m
WHERE  year <> ANY(
                SELECT  year
        FROM    Movie
        WHERE   title =  m.title)
```

Find movies whose title appears more than once.

Note the scoping of the variables!

Simple Aggregations

Purchase

Product	Date	Price	Quantity
bagel	10/21	1	20
banana	10/3	0.5	10
banana	10/10	1	10
bagel	10/25	1.50	20

```
SELECT SUM(price * quantity)
FROM   Purchase
WHERE  product = ‘bagel’
```

→ 50 (= 1*20 + 1.50*20)

Grouping & Aggregations: GROUP BY

```
SELECT   product, SUM(price*quantity)
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING   SUM(quantity) > 10
```

Find total sales after 10/1/2005, only for products that have more than
10 total units sold

HAVING clauses contains conditions on **aggregates**

*Whereas WHERE clauses condition on **individual tuples***

Order of Operations

```
SELECT   product, SUM(price*quantity)
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING   SUM(quantity) > 10
```

HAVING clauses contains conditions on **aggregates**

*Whereas WHERE clauses condition on **individual tuples…***

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

GROUP BY: (1) Compute FROM-WHERE

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING SUM(quantity) > 10
```

FROM
WHERE

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Craisins	11/1	2	5
Craisins	11/3	2.5	3

GROUP BY: (2) Aggregate by the GROUP BY

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING SUM(quantity) > 10
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Craisins	11/1	2	5
Craisins	11/3	2.5	3

GROUP BY

Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10
Craisins	11/1	2	5
	11/3	2.5	3

GROUP BY: (3) Filter by the HAVING clause

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING SUM(quantity) > 30
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10
Craisins	11/1	2	5
	11/3	2.5	3

HAVING →

Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10

GROUP BY: (3) SELECT clause

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING SUM(quantity) > 100
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10

SELECT →

Product	TotalSales
Bagel	50
Banana	15

General form of Grouping and Aggregation

```
SELECT    S
FROM      R1,…,Rn
WHERE     C1
GROUP BY  a1,…,ak
HAVING    C2
```

Evaluation steps:

1. Evaluate FROM-WHERE: apply condition C_1 on the attributes in $R_1,\ldots,R_n$
2. GROUP BY the attributes $a_1,\ldots,a_k$
3. Apply HAVING condition C_2 to each group (may have aggregates)
4. Compute aggregates in SELECT, S, and return the result

General form of Grouping and Aggregation

```
SELECT    S
FROM      R1,…,Rn
WHERE     C1
GROUP BY  a1,…,ak
HAVING    C2
```

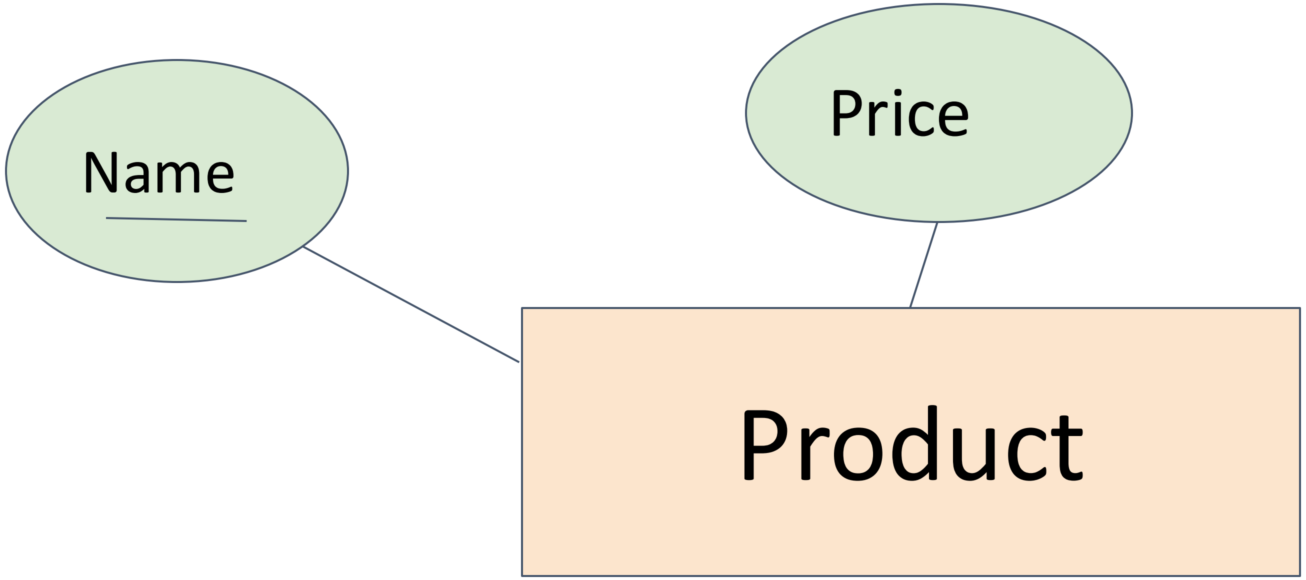
- S = Can ONLY contain attributes $a_1,\ldots,a_k$ and/or aggregates over other attributes
- C_1 = is any condition on the attributes in $R_1,\ldots,R_n$
- C_2 = is any condition on the aggregate expressions

Practice Question

- 2. SQL Reading
  - 2.2

E/R Diagram

Entity Set

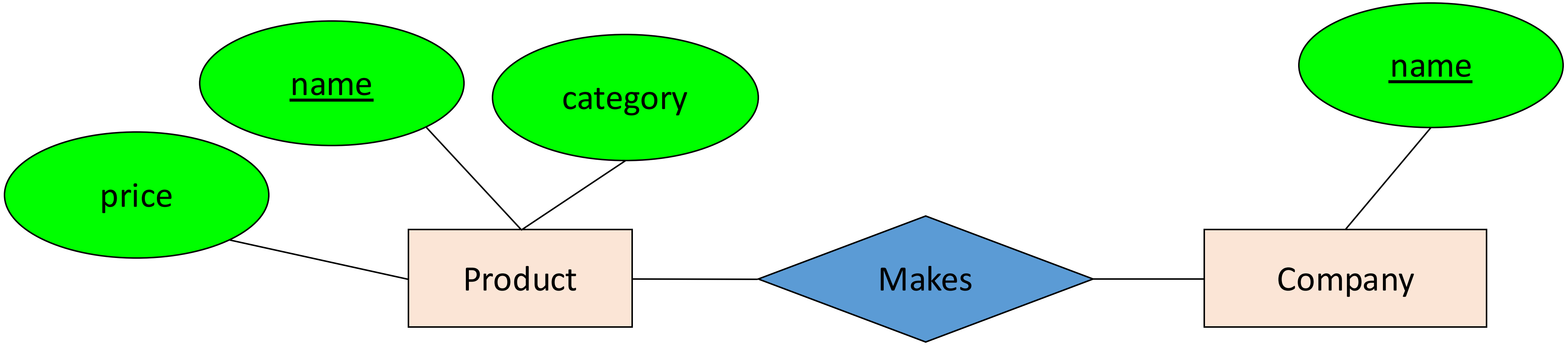


A *key* is a **minimal** set of attributes that uniquely identifies an entity.

Relationship

A **relationship** is between two entities

- Represented by diamonds



What is a Relationship?

Company

name
GizmoWorks
GadgetCorp

Product

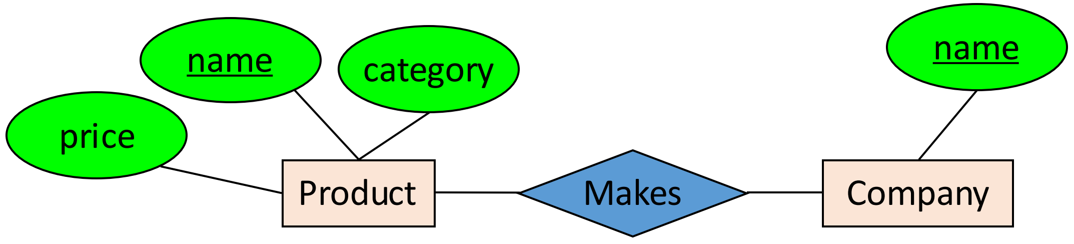
name	category	price
Gizmo	Electronics	$9.99
GizmoLite	Electronics	$7.50
Gadget	Toys	$5.50

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

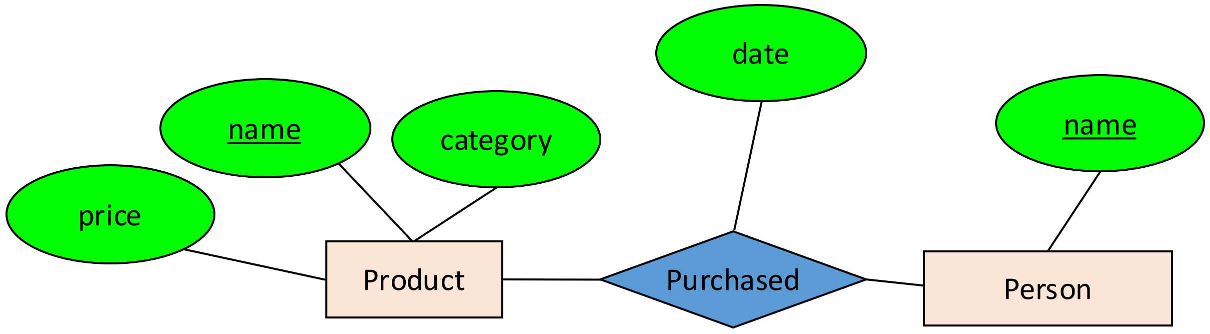
C.name	P.name
GizmoWorks	Gizmo
GizmoWorks	GizmoLite
GadgetCorp	Gadget



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

Modeling something as a relationship makes it unique

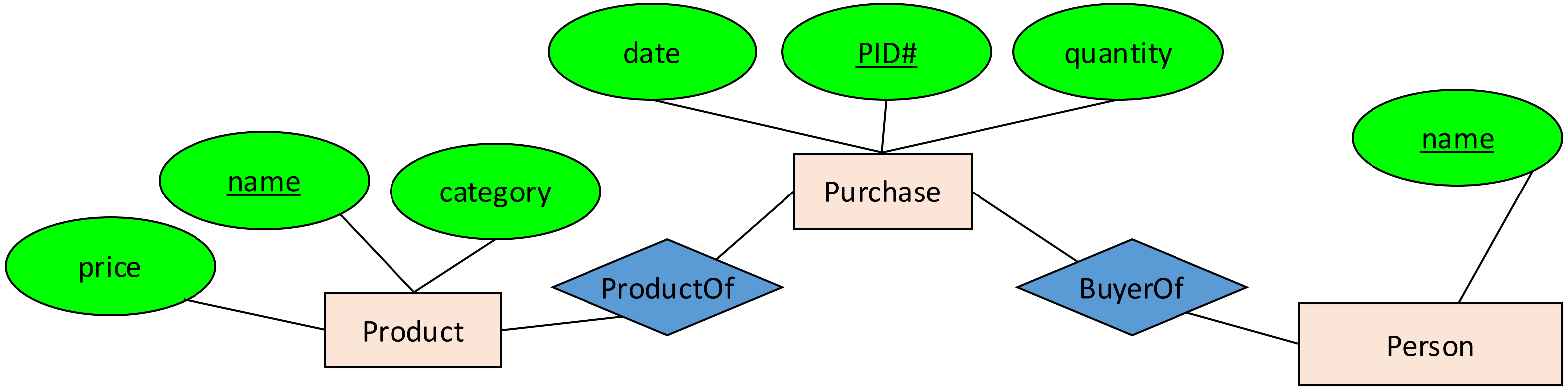
Q: What does this say?



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

Modeling something as a relationship makes it unique

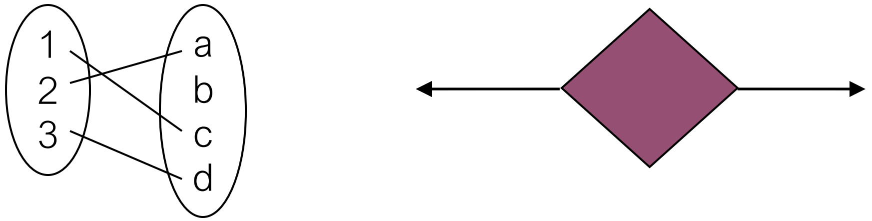
Q: What about this way?



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

Multiplicity of binary relationships

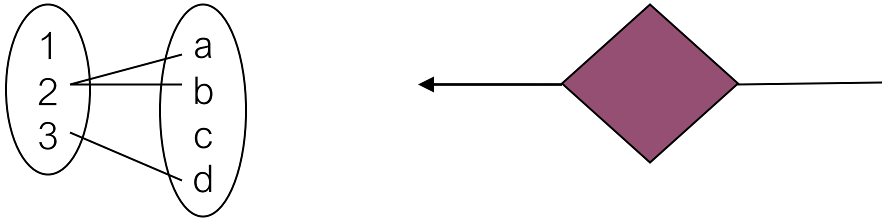
One-to-one:



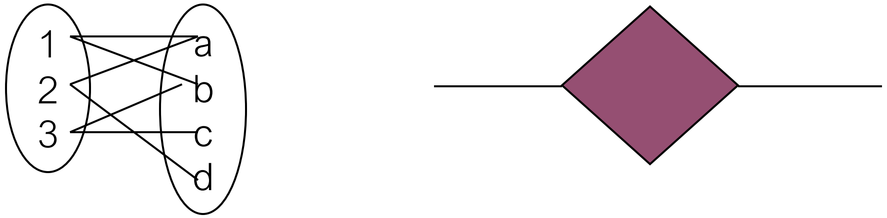
Indicated using arrows

Many-to-one:

One-to-many:

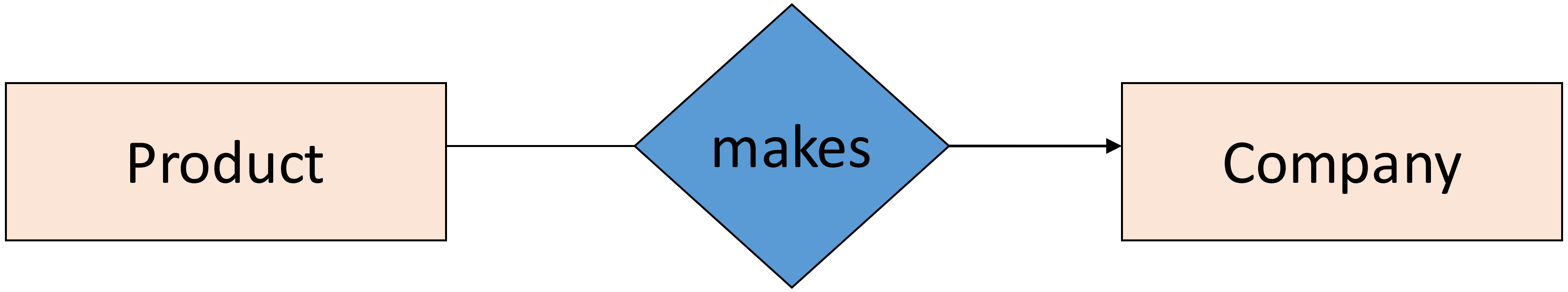


Many-to-many:

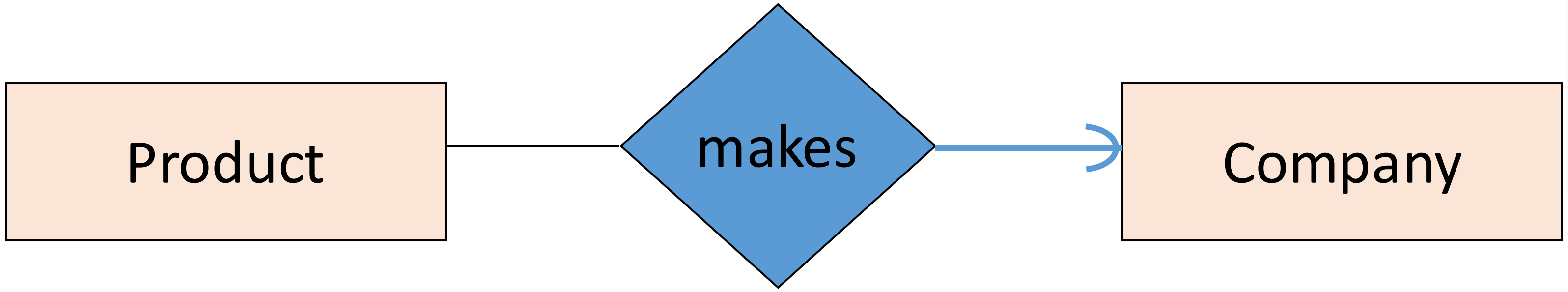


Referential Integrity Constraints

Arrow: <= 1
Rounded arrow: =1



Each product made by <u>at most one</u> company.
Some products made by no company?



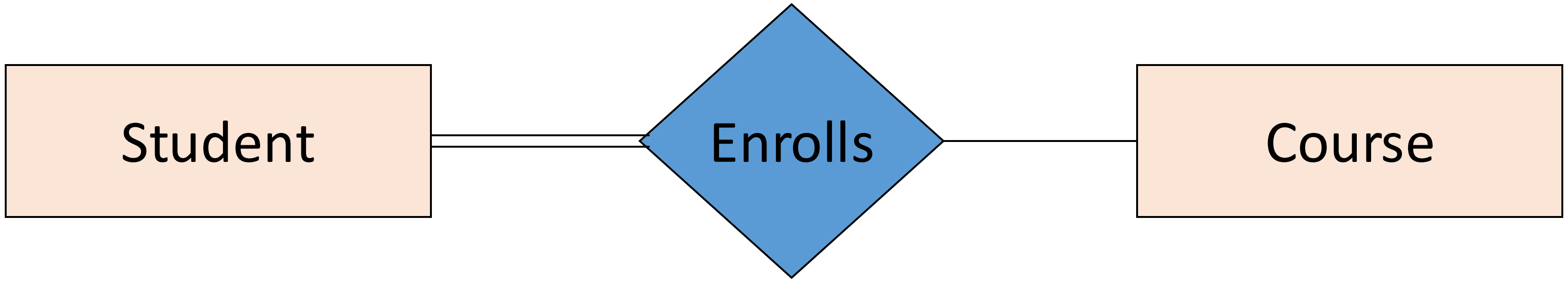
A rounded arrow to F means

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

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

Participation Constraints: Partial v. Total

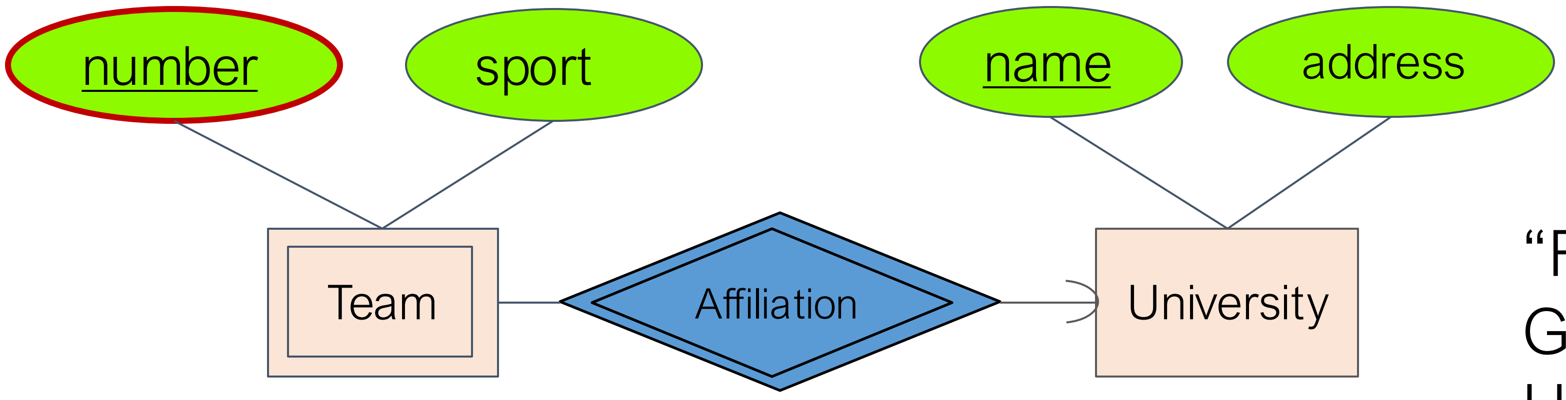
- **Partial participation** (single line): Some entities may exist without being associated with the relationship.
- **Total participation** (double line): all entities must be associated with at least one instance of the relationship.



- Every student must enroll in at least one course
- Some courses might not have any students.

Weak entity set

Entity sets are <u>weak</u> when their key comes from other classes to which they are related.



"Football team" v. "The GT Football team" (E.g., UGA has a football team too)

- number is a *partial key.*
- The key also contains keys of the University entity set
- Affiliation must have referential integrity from Team to University

Practice Question

- 3. ER Diagram

Design Theory

Data Anomalies

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

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

Finding Functional Dependencies

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

Inference problem: How do we decide?

Three simple rules called **Armstrong's Rules.**

1. **Reflexivity,**
2. **Augmentation, and**
3. **Transitivity…**

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs **F:**
Then the **closure**, $\{A_1, \ldots, A_n\}^+$ is the set of attributes **B** s.t. $\{A_1, \ldots, A_n\} \rightarrow B$

Example: F =

{name} → {color}
{category} → {department}
{color, category} → {price}

Example Closures:

$\{name\}^+$ = {name, color}
$\{name, category\}^+$ =
{name, category, color, dept, price}
$\{color\}^+$ = {color}

Closure algorithm

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

Repeat until X doesn't change; **do**:

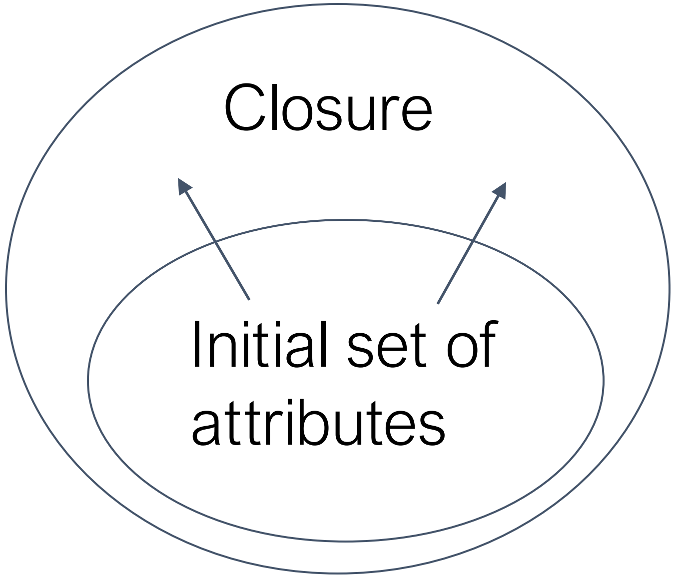
if $\{B_1, \ldots, B_n\} \rightarrow C$ is entailed by F

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



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

Meaning that no subset of a key is also a superkey

Computing Keys and Superkeys

- <u>Superkey?</u>
  - Compute the closure of A
  - See if it = the full set of attributes
- <u>Key?</u>
  - Confirm that A is superkey
  - Make sure that no subset of A is a superkey
    - *Only need to check one 'level' down!*

Let A be a set of attributes, R set of all attributes, F set of FDs:

```
IsSuperkey(A, R, F):
A+ = ComputeClosure(A, F)
Return (A+==R)?
```

```
IsKey(A, R, F):
If not IsSuperkey(A, R, F):
        return False
For B in SubsetsOf(A, size=len(A)-1):
        if IsSuperkey(B, R, F):
                return False
return True
```

Boyce-Codd Normal Form

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: ∀ sets of attributes X, either ($X^+ = X$) or (X^+ = all attributes)

Example

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

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

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

Lossless Decompositions

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

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

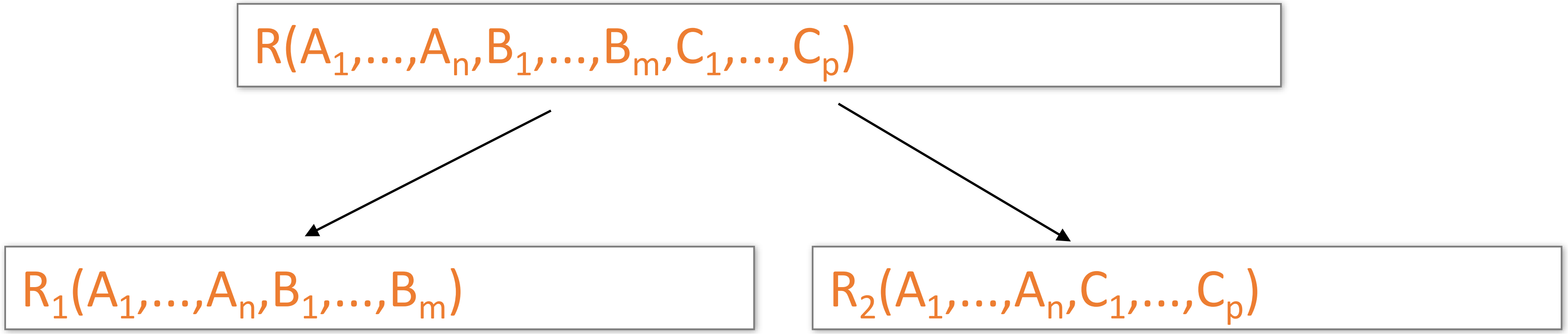
$R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

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

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

A decomposition R to (R1, R2) is <u>lossless</u>
if R = R1 Join R2

Lossless Decompositions



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.

Lossy vs. Lossless

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

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

Lossy vs. Lossless

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

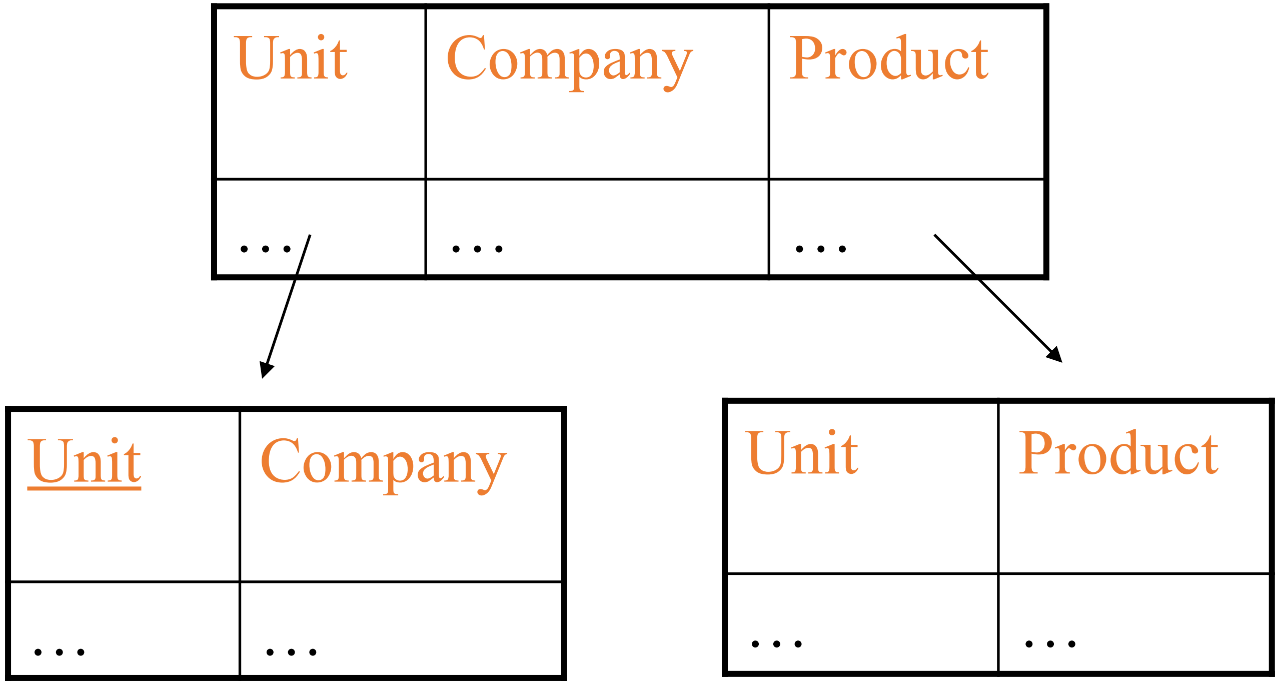
{Category} → {Name}

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Recorder

Price	Category
19.99	Gadget
24.99	Camera
19.99	Recorder

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

A Problem with BCNF



{Unit} → {Company}

{Company,Product} → {Unit}

{Unit} → {Company}

We do a BCNF decomposition on a “bad” FD:

$\{Unit\}^+ = \{Unit, Company\}$

We lose the FD {Company,Product} → {Unit}!!

Third normal form (3NF)

A relation R is in 3NF if:

For every non-trivial FD $A_1, ..., A_n \rightarrow B$, either

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

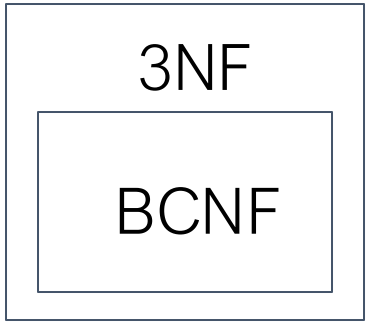
Example:

- 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)

$AC \rightarrow B$
$B \rightarrow C$

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

Practice Question

- 5.2 3CNF

MVD Example

Movie_ Star (A)	Address (B)	Movie (C)
Leonardo DiCaprio	Los Angeles	Titanic
Leonardo DiCaprio	Los Angeles	Inception
Leonardo DiCaprio	New York	Titanic
Leonardo DiCaprio	New York	Inception
Scarlett Johansson	Los Angeles	Black Widow
Scarlett Johansson	Los Angeles	Her
Scarlett Johansson	Paris	Black Widow
Scarlett Johansson	Paris	Her

- **Independence**: The set of addresses is independent of the set of movies for a given movie star.
- **Redundancy**: Notice how each movie is repeated for every address that the star lives in, and vice versa.

MVD Example

	Movie_ Star (A)	Address (B)	Movie (C)
t_1	Leonardo DiCaprio	Los Angeles	Titanic
t_3	Leonardo DiCaprio	Los Angeles	Inception
	Leonardo DiCaprio	New York	Titanic
t_2	Leonardo DiCaprio	New York	Inception
	Scarlett Johansson	Los Angeles	Black Widow
	Scarlett Johansson	Los Angeles	Her
	Scarlett Johansson	Paris	Black Widow
	Scarlett Johansson	Paris	Her

We write $\mathbf{A} \twoheadrightarrow \mathbf{B}$ if for any tuples t_1,t_2 s.t. $t_1[A] = t_2[A]$ there is a tuple t_3 s.t.

- $t_3[A] = t_1[A]$
- $t_3[B] = t_1[B]$
- and $t_3[R \backslash B] = t_2[R \backslash B]$

Where R\B is "R minus B" i.e. the attributes of R not in B

Multi-Value Dependencies (MVDs)

One less formal, literal way to phrase the definition of an MVD:

The MVD $\boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$ holds on R if for any pair of tuples with the same X values, the tuples with the same X values, but the other permutations of Y and A\Y values, is also in R

Ex: X = {x}, Y = {y}:

x	y	z
1	0	1
1	1	0

For $\boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$ to hold must have…

x	y	z
1	0	1
1	1	0
1	0	0
1	1	1

Practice Question

- 4.4 MVD

Relational Algebra

Relational Algebra (RA)

- <u>Five **basic** operators:</u>
  1. Selection: σ
  2. Projection: Π
  3. Cartesian Product: $\times$
  4. Union: $\cup$
  5. Difference: -
- <u>Derived or auxiliary operators:</u>
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ
  - Grouping: γ

RDBMSs use *multisets*, however in relational algebra formalism we will consider **sets!**

Selection (σ)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- The condition c can be =, <, >, <>

Students(sid,sname,gpa)

SQL:

```
SELECT *
FROM Students
WHERE gpa > 3.5;
```

RA:

$$\sigma_{gpa > 3.5}(Students)$$

Projection (Π)

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1,\ldots,An}(R)$

Students(sid,sname,gpa)

SQL:

```
SELECT DISTINCT
  sname,
  gpa
FROM Students;
```

RA:

$$\Pi_{sname,gpa}(Students)$$

Renaming (ρ)

- Changes the schema, not the instance
- A ‘special’ operator- neither basic nor derived
- Notation: $\rho_{B1,\ldots,Bn}(R)$

- **Note: this is shorthand for the proper form (since names, not order matters!):**
  - $\rho_{A1\rightarrow B1,\ldots,An\rightarrow Bn}(R)$

Students(sid,sname,gpa)

SQL:

```
SELECT
  sid AS studId,
  sname AS name,
  gpa AS gradePtAvg
FROM Students;
```

RA:

$$\rho_{studId,name,gradePtAvg}(Students)$$

Natural Join (⋈)

- Notation: $R_1 \bowtie R_2$
- Joins R_1 and R_2 on *equality of all shared attributes*
  - If R_1 has attribute set A, and R_2 has attribute set B, and they share attributes A⋂B = C, can also be written: $R_1 \bowtie_C R_2$
- Our first example of a *derived* RA operator:
  - Meaning: $R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \to D}(R_1) \times R_2))$
  - Where:
    - The rename $\rho_{C \to D}$ renames the shared attributes in one of the relations
    - The selection $\sigma_{C=D}$ checks equality of the shared attributes
    - The projection $\Pi_{A \cup B}$ eliminates the duplicate common attributes

Students(sid,name,gpa)
People(ssn,name,address)

SQL:

```
SELECT DISTINCT
  ssid, S.name, gpa,
  ssn, address
FROM
  Students S,
  People P
WHERE S.name = P.name;
```

RA:

$$Students \bowtie People$$

Converting SFW Query -> RA

You should also be able to convert RA -> SQL query.

Remember to add the DISTINCT keyword

SELECT DISTINCT $A_1,\ldots,A_n$
FROM $R_1,\ldots,R_m$
WHERE c_1 AND … AND c_k;

$$\Pi_{A_1,\ldots,An}(\sigma_{c_1} \ldots \sigma_{c_k}(R_1 \bowtie \cdots \bowtie R_m))$$

The selections should happen before the projections