Updated 9/14/2022

Topic 1: SQL
L03: SQL intermediate

Wolfgang Gatterbauer

CS3200 Database design (fa22)

https://northeastern-datalab.github.io/cs3200/fa22s3/

9/14/2022

Class warm-up

- Last class recapitulation
- name plates
- more SQL (Please keep up the great questions!)

Quiz

Our colorful hands represent "team exercises"
If we are online, please make a screenshot!

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM
WHERE
```

Quiz: Answer 1

Our colorful hands represent "team exercises"
If we are online, please make a screenshot!

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and P.price < 20
    and P.price > 25
    and P.manufacturer = cName
```

What about this query?

?

Quiz: Answer 1

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and P.price < 20
    and P.price > 25
    and P.manufacturer = cName
```

Wrong! Gives empty result: There is no product with price <20 and >25

Quiz: Answer 1

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

C

country='USA'

P

P.price < 20 and
P.price > 25

not possible!
→ Empty result

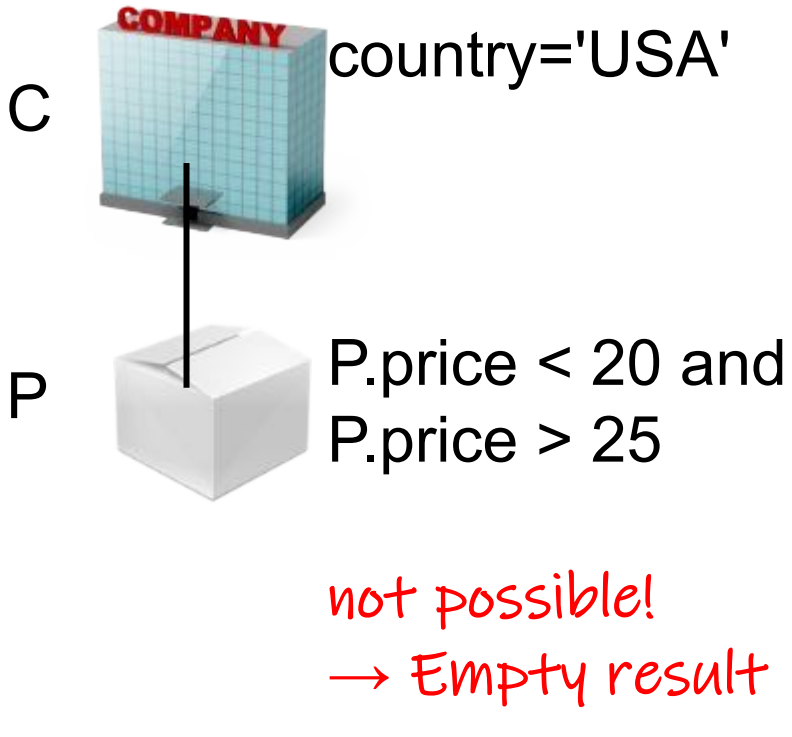
What do we actually want?

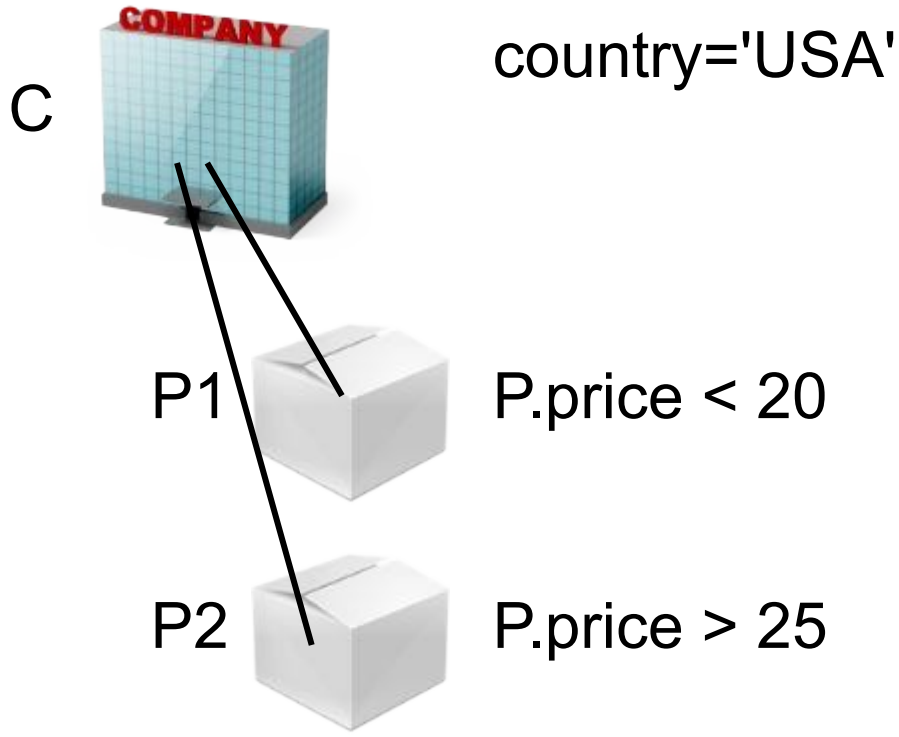
?

Quiz: Answer 1 vs. what we actually want


Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.





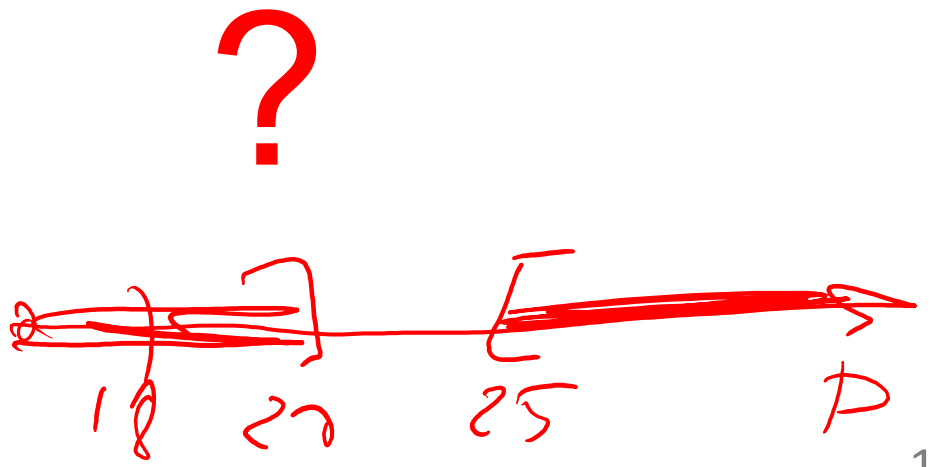
Quiz: Answer 2

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT   DISTINCT cName
FROM     Product as P, Company
WHERE    country = 'USA'
    and  (P.price < 20
    or   P.price > 25)
    and  P.manufacturer = cName
```

What about this query?



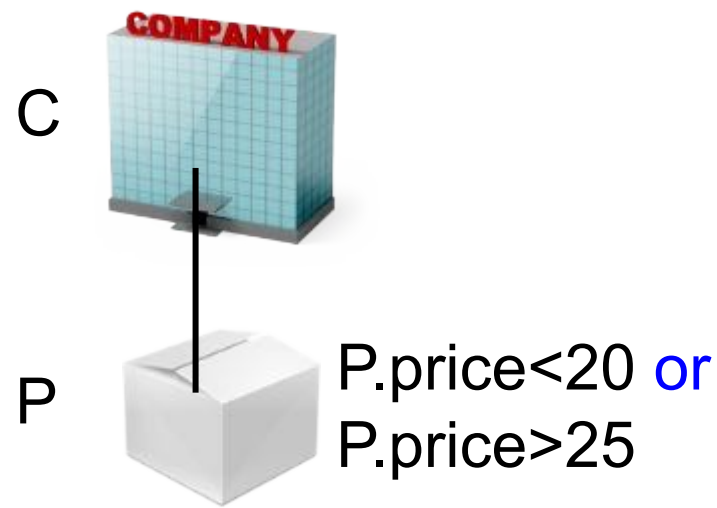
Quiz: Answer 2

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

Returns companies with single product w/price (<20 or >25)

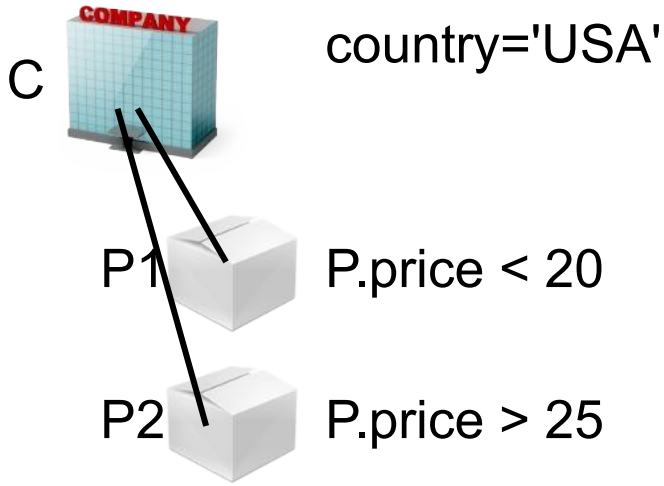
```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and (P.price < 20
    or   P.price > 25)
    and  P.manufacturer = cName
```



Quiz: correct answer: we need "self-joins"!


Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

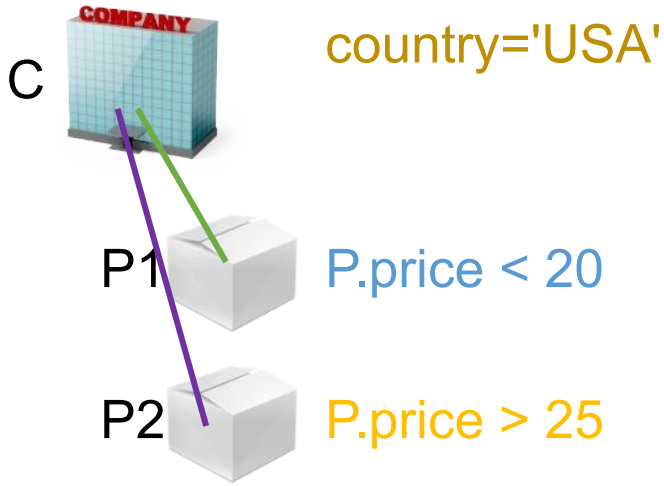



```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

Quiz: correct answer: we need "self-joins"!


Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.




```
SELECT   DISTINCT cName
FROM     Product as P1, Product as P2, Company
WHERE    country = 'USA'
     and P1.price < 20
     and P2.price > 25
     and P1.manufacturer = cName
     and P2.manufacturer = cName
```

Quiz: correct answer: we need "self-joins"!


P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

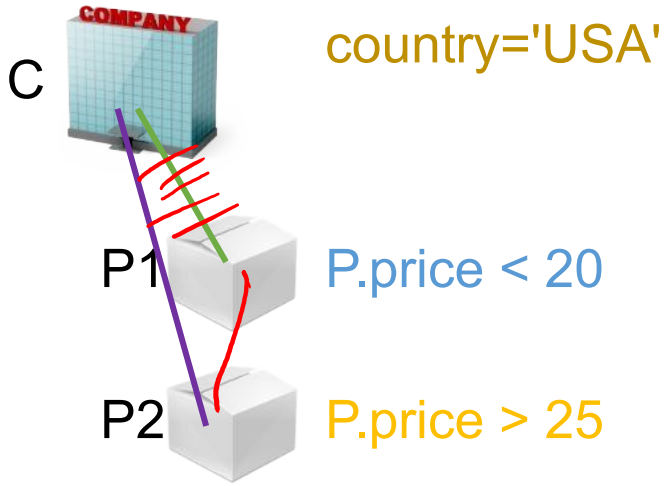
P2

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT   DISTINCT cName
FROM     Product as P1, Product as P2, Company
WHERE    country = 'USA'
     and P1.price < 20
     and P2.price > 25
     and P1.manufacturer = cName
     and P2.manufacturer = cName
```



Quiz response: we need "self-joins"!


P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

P2

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

CName
GizmoWorks

Inner Joins

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT  *
FROM    Employee E, Department D
WHERE   E.DepartmentID = D.DepartmentID
```

?

Source: http://en.wikipedia.org/wiki/Join_(SQL)#Cross_join
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

Inner Joins

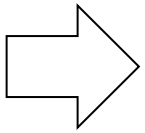
Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT  *
FROM    Employee E, Department D
WHERE   E.DepartmentID = D.DepartmentID
```



E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Robinson	34	34	Clerical
Jones	33	33	Engineering
Smith	34	34	Clerical
Steinberg	33	33	Engineering
Rafferty	31	31	Sales

Source: http://en.wikipedia.org/wiki/Join_(SQL)#Cross_join

Cross Joins: usually not what you want ☹

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT  *
FROM    Employee E, Department D
```

WHERE 1 = 1

?

Cross Joins: usually not what you want ☹

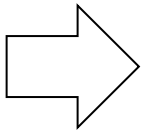
Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT  *
FROM    Employee E, Department D
```



E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Rafferty	31	31	Sales
Jones	33	31	Sales
Steinberg	33	31	Sales
Smith	34	31	Sales
Robinson	34	31	Sales
Rafferty	31	33	Engineering
Jones	33	33	Engineering
Steinberg	33	33	Engineering
Smith	34	33	Engineering
Robinson	34	33	Engineering
Rafferty	31	34	Clerical
Jones	33	34	Clerical
Steinberg	33	34	Clerical
Smith	34	34	Clerical
Robinson	34	34	Clerical
Rafferty	31	35	Marketing
Jones	33	35	Marketing
Steinberg	33	35	Marketing
Smith	34	35	Marketing
Robinson	34	35	Marketing

Definitions (for job interviews?)

- An equi-join is a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table
- A natural join is an equi-join in which one of the duplicate columns is eliminated in the result table
- A cross join returns the Cartesian product (also "cross product") of rows from tables in the join
  - (i.e. it will produce rows which combine each row from the first table with each row from the second table, that's usually *not* what you want)

Definitions (for job interviews?)

Equi-join

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Robinson	34	34	Clerical
Jones	33	33	Engineering
Smith	34	34	Clerical
Steinberg	33	33	Engineering
Rafferty	31	31	Sales

Natural join

E.LastName	DepartmentID	D.DepartmentName
Robinson	34	Clerical
Jones	33	Engineering
Smith	34	Clerical
Steinberg	33	Engineering
Rafferty	31	Sales

Cross join

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Rafferty	31	31	Sales
Jones	33	31	Sales
Steinberg	33	31	Sales
Smith	34	31	Sales
Robinson	34	31	Sales
Rafferty	31	33	Engineering
...	...	...	...

Alternative Join Syntax

Alternative JOIN Syntax ("inner joins")

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT   *
FROM     Employee E, Department D
WHERE    E.DepartmentID = D. DepartmentID
AND      E.DepartmentID = 34
```

```
SELECT   *
FROM     Employee E JOIN Department D
     ON  E.DepartmentID = D. DepartmentID
WHERE    E.DepartmentID = 34
```

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Robinson	34	34	Clerical
Smith	34	34	Clerical

NATURAL JOIN Syntax

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT    LastName, E.DepartmentID, Department Name
FROM      Employee E, Department D
WHERE     E.DepartmentID = D. DepartmentID
AND       E.DepartmentID = 34
```

```
SELECT    *
FROM      Employee E NATURAL JOIN Department D

WHERE     E.DepartmentID = 34
```

LastName	DepartmentID	DepartmentName
Robinson	34	Clerical
Smith	34	Clerical

Natural join syntax is not supported by all DBMS's and not recommended

(someone reading the query does not see the join keys. That makes is harder to debug queries or to modify & enhance them)

Also imagine two tables with "address" field …

More Practice of the Conceptual Evaluation Strategy

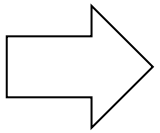
Using the Formal Semantics

R(a), S(a), T(a)

R	S	T
a	a	a
1	1	2
2		

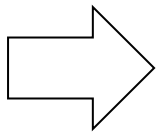
What do these queries compute?

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



?

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
    or  R.a=T.a
```



?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

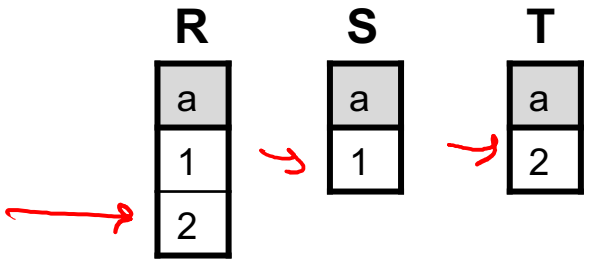
Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

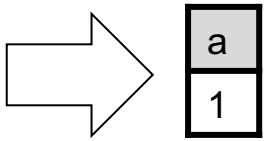
Using the Formal Semantics

R(a), S(a), T(a)

What do these queries compute?

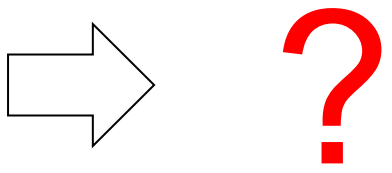



```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



Returns R ∩ S (intersection)

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
    or  R.a=T.a
```



SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

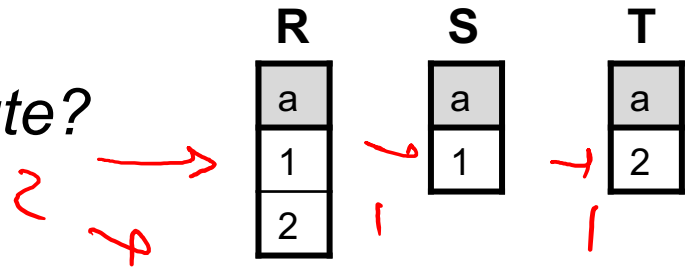
Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

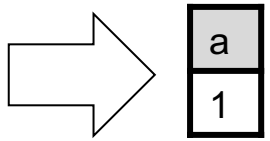
Using the Formal Semantics

R(a), S(a), T(a)

What do these queries compute?

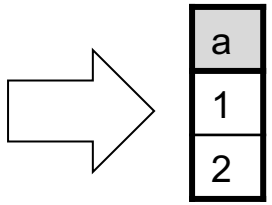



```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



Returns R ∩ S
(intersection)

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
    or  R.a=T.a
```



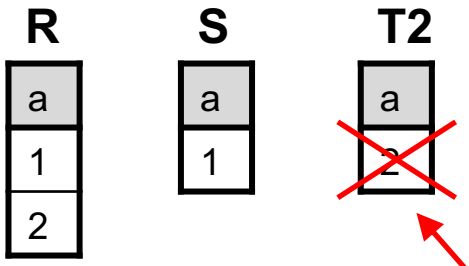
Returns R ∩ (S ∪ T)
if S ≠ ∅ and T ≠ ∅

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

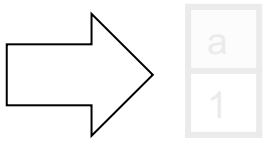
Using the Formal Semantics


R(a), S(a), T2(a)

What do these queries compute?




```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

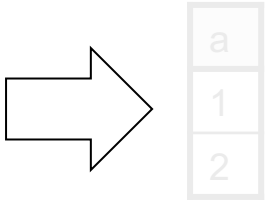


Returns R ∩ S (intersection)

?

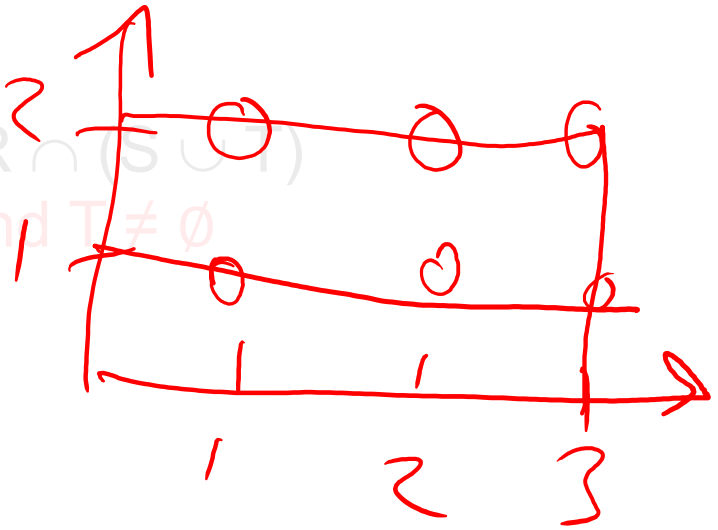
Next, we are removing the input tuple "(2)"

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
    or  R.a=T.a
```



Returns R ∩ (S ∪ T) if S ≠ ∅ and T ≠ ∅

?



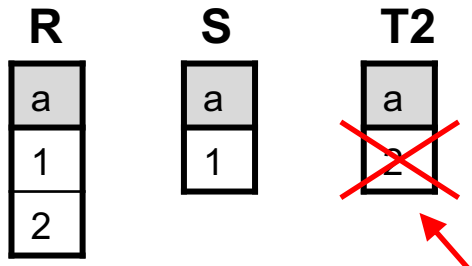
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

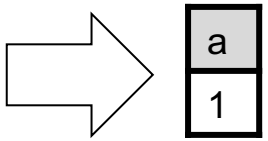
Using the Formal Semantics


R(a), S(a), T2(a)

What do these queries compute?




```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



Returns R ∩ S (intersection)

Next, we are removing the input tuple "(2)"

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
    or  R.a=T.a
```


Returns R ∩ (S ∪ T) if S ≠ ∅ and T ≠ ∅

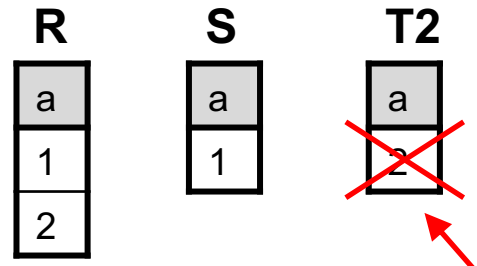
?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

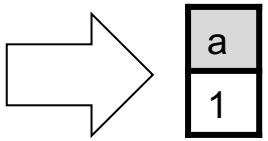
Using the Formal Semantics

R(a), S(a), T2(a)

What do these queries compute?



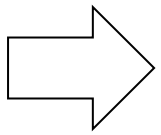

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



Returns R ∩ S (intersection)

Next, we are removing the input tuple "(2)"

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
   or   R.a=T.a
```



Returns ∅ if S = ∅ or T = ∅

Can seem counterintuitive! But remember conceptual evaluation strategy: Nested loops. If one table is empty -> no looping

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

Illustration with Python

Python file

```
'''
Created on 3/23/2015
Illustrates nested Loop Join in SQL
__author__ = 'gatt'
'''

print "--- 1st nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(2):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 2nd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(1):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 3rd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(0):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print
```

```
/Library/Frameworks/Python.framework/Versio
--- 1st nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=0, k=1:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=1, k=1:
i=0, j=2, k=0:  TRUE
i=0, j=2, k=1:
i=1, j=0, k=0:
i=1, j=0, k=1:  TRUE
i=1, j=1, k=0:  TRUE
i=1, j=1, k=1:  TRUE
i=1, j=2, k=0:
i=1, j=2, k=1:  TRUE

--- 2nd nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=2, k=0:  TRUE
i=1, j=0, k=0:
i=1, j=1, k=0:  TRUE
i=1, j=2, k=0:

--- 3rd nested loop ---

Process finished with exit code 0
```

The comparison gets never evaluated

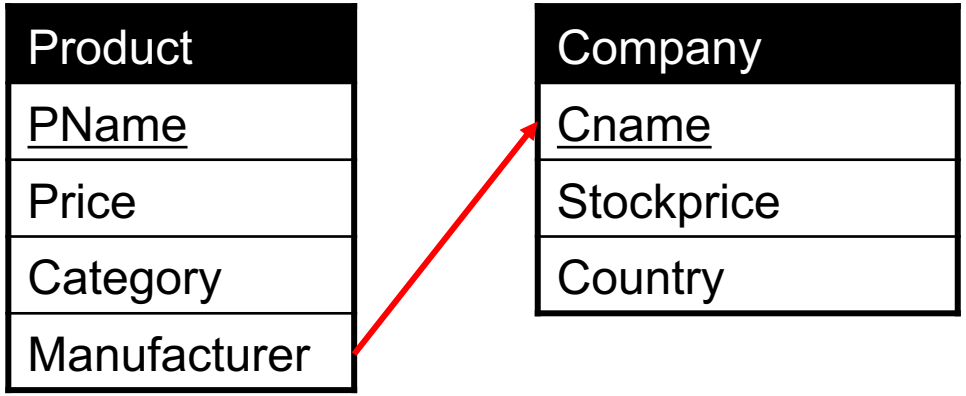
"Premature optimization is the root of all evil."
Donald Knuth (1974)

"When you are diagnosing problems, don't think about how you will solve them—just diagnose them. Blurring the steps leads to suboptimal outcomes because it interferes with uncovering the true problems."
Ray Dalio (Principles, 2017)

Python example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

CREATE Tables
&
INSERT Values

(Relational Database) Schema



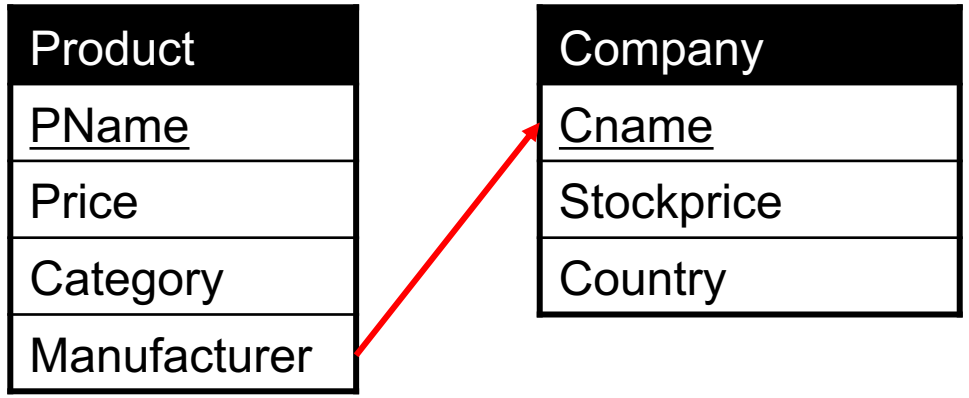
"Schema": describes the structure of data in terms of the relational data model.

A schema includes tables, columns, PKs, FKs, and other constraints

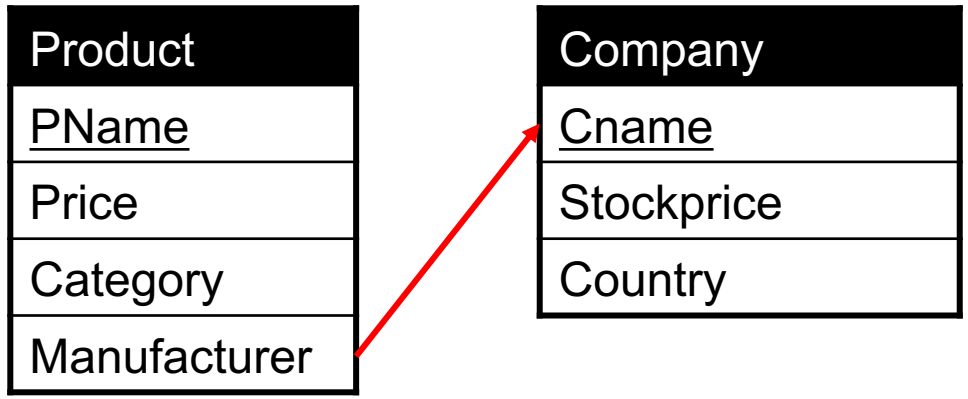
Product(pname, price, category, manufacturer)
Company(cname, stockprice, country)

Product.manufacturer is FK to Company


```
-----------------------------
-- Create the tables
-----------------------------

create table Company (

                        );

create table Product (

                                                    );
```

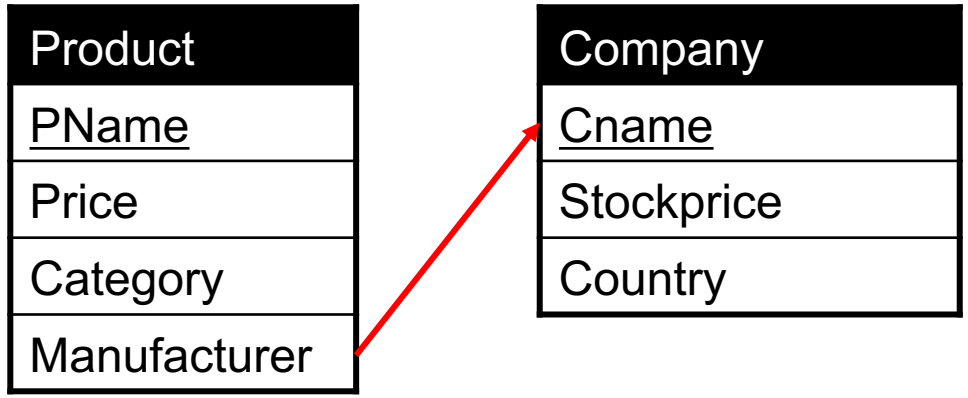


```
-----------------------------
-- Create the tables
-----------------------------

create table Company (
      CName char(20)                    ,
      StockPrice int,
      Country char(20) );

create table Product (
      PName char(20),
      Price decimal(9, 2),
      Category char(20),
      Manufacturer char(20),

                                        );
```



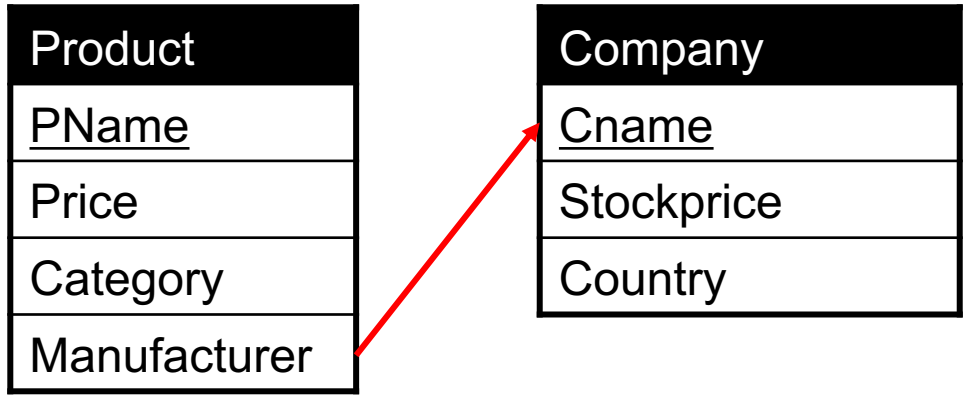
- attribute decimal(9,2), "insert ... values (12.34567, ...)" ⇒ will store 12.35 ! Use decimal = numeric to perform calculations exactly (e.g., decimal(precision,scale): 12.345 has precision of 5 and scale of 3)
- double = real: approximate types (e.g., 0.1+0.2=0.30...004, see "What every programmer should know about floating-point arithmetic": https://floating-point-gui.de/)


```
-----------------------------
-- Create the tables
-----------------------------

create table Company (
      CName char(20) PRIMARY KEY,
      StockPrice int,
      Country char(20) );

create table Product (
      PName char(20),
      Price decimal(9, 2),
      Category char(20),
      Manufacturer char(20),
```

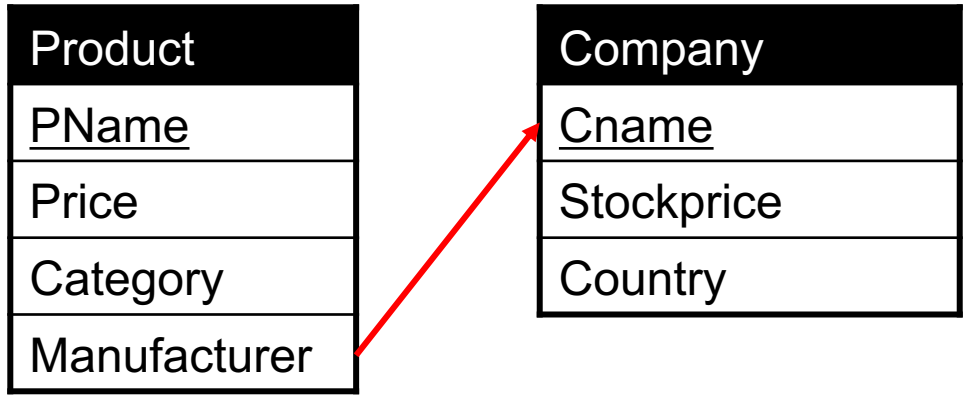



```
);
```

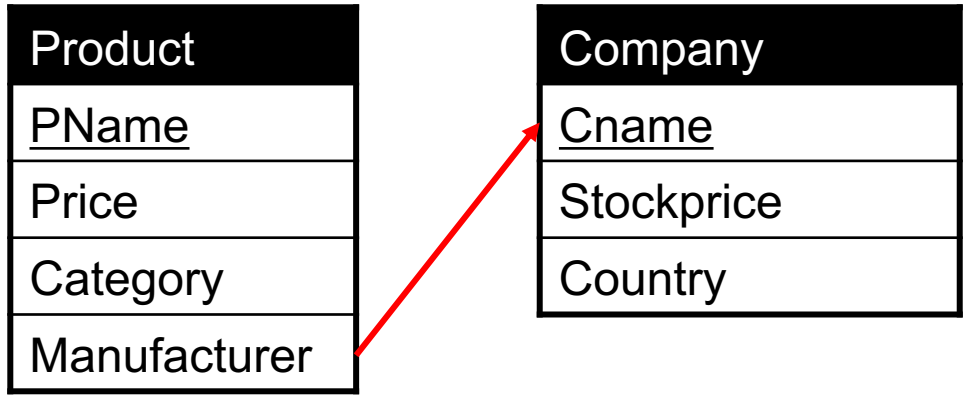
```
-----------------------------
-- Create the tables
-----------------------------

create table Company (
      CName char(20) PRIMARY KEY,
      StockPrice int,
      Country char(20) );

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




```
-----------------------------
-- Create the tables
-----------------------------

create table Company (
      CName char(20) PRIMARY KEY,
      StockPrice int,
      Country char(20) );

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

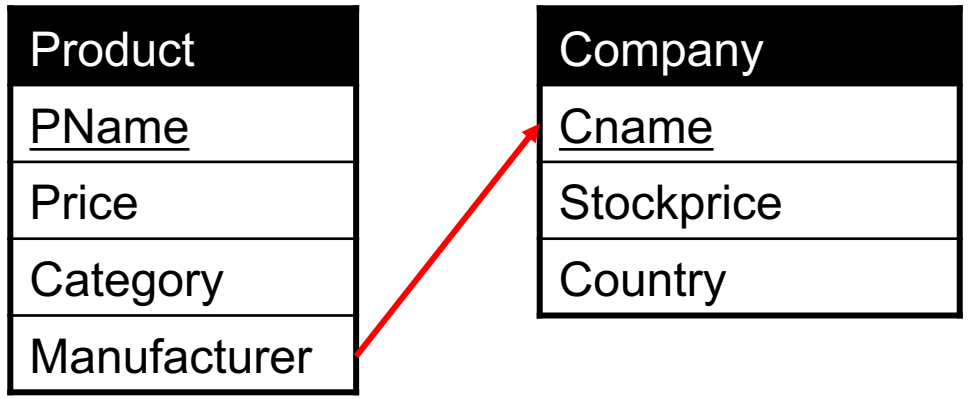


```
-----------------------------
-- Create the tables
-----------------------------

create table Company (
      CName char(20) PRIMARY KEY,
      StockPrice int,
      Country char(20) );

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




```
-----------------------------
-- Create the tables
-----------------------------

create table Company (
      CName char(20) PRIMARY KEY,
      StockPrice int,
      Country char(20) );

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

```
-----------------------------
-- Drop tables if they already exist
-----------------------------

DROP TABLE IF EXISTS Product;
DROP TABLE IF EXISTS Company;
```



If you try to drop Company first, then you might get an error. Why?


```
-----------------------------
-- Populate the tables
-----------------------------

insert into Company values ('GizmoWorks', 25, 'USA');
insert into Company values ('Canon', 65, 'Japan');
insert into Company values ('Hitachi', 15, 'Japan');

insert into Product values ('Gizmo', 19.99, 'Gadgets', 'GizmoWorks');
insert into Product values ('PowerGizmo', 29.99, 'Gadgets', 'GizmoWorks');
insert into Product values ('SingleTouch', 149.99, 'Photography', 'Canon');
insert into Product values ('MultiTouch', 203.99, 'Household', 'Hitachi');
```

1. Aggregates
2. Groupings
3. Having

Aggregation

Car (name, price, maker)


```
SELECT  avg(price)
FROM    Car
WHERE   maker='Toyota'
```

```
SELECT  count(*)
FROM    Car
WHERE   price > 100
```

SQL supports several aggregation operations. Most important:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute (!)

Aggregation


```
SELECT  avg(price)
FROM    Car
WHERE   maker='Toyota'
```

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
Prius	50	Toyota
Lexus1	75	Toyota
Lexus2	100	Toyota

?

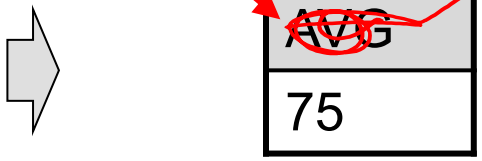
Aggregation


```
SELECT  avg(price)
FROM    Car
WHERE   maker='Toyota'
```

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
Prius	50	Toyota
Lexus1	75	Toyota
Lexus2	100	Toyota

Database creates new attribute name
(e.g. "(no column name)" in SQLlite)



Aggregation with rename


```
SELECT  count(*) as n
FROM    Car
WHERE   price > 100
```

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
Prius	50	Toyota
Lexus1	75	Toyota
Lexus2	100	Toyota

?

Aggregation with rename

"as" optional

```
SELECT  count(*) as n
FROM    Car
WHERE   price > 100
```

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
Prius	50	Toyota
Lexus1	75	Toyota
Lexus2	100	Toyota

Database creates *our* new attribute name

n
2

Aggregation: Count Distinct


```
SELECT  count(maker)
FROM    Car
WHERE   price > 100
```

?

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
~~Prius~~	~~50~~	~~Toyota~~
~~Lexus1~~	~~75~~	~~Toyota~~
~~Lexus2~~	~~100~~	~~Toyota~~

Aggregation: Count Distinct


```
SELECT  count(maker)
FROM    Car
WHERE   price > 100
```

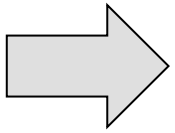
Same as count(*)

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
~~Prius~~	~~50~~	~~Toyota~~
~~Lexus1~~	~~75~~	~~Toyota~~
~~Lexus2~~	~~100~~	~~Toyota~~

We probably want to ignore duplicates:

```
SELECT  count(DISTINCT maker)
FROM    Car
WHERE   price > 100
```



COUNT
1

Simple Aggregation 1/3

Purchase (product, price, quantity)

```
SELECT  sum(price * quantity)
FROM    Purchase
```

```
SELECT  sum(price * quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```

What do these queries mean?

(next slides)

Simple Aggregation 3/3

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
~~Banana~~	~~2~~	~~10~~
~~Banana~~	~~4~~	~~10~~

```
SELECT  sum(price) * sum(quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```

?