Updated 9/21/2022

Topic 1: SQL
L05: SQL intermediate

Wolfgang Gatterbauer

CS3200 Database design (fa22)

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

9/21/2022

Class warm-up

- Last class summary
- Extra office hours with Grishma Alshi: 1-3pm on THU
- groupme.com (2010, 2011 Skype, 2011 MSN)
- Keep notes on feedback, esp. collaboration policy on homeworks

- SQL today: nested queries
- SQL next: Nulls, outer joins, "witnesses" (traditionally students find this topic the conceptually most difficult)

1. Aggregates
2. Groupings
3. Having

From → Where → Group By → Select

Purchase

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

Product	TotalQuantities
Bagel	40
Banana	20

```
SELECT      product, sum(quantity) as TotalQuantities
FROM        Purchase
WHERE       price > 1
GROUP BY    product
```

From → Where → Group By → Select

308

Purchase

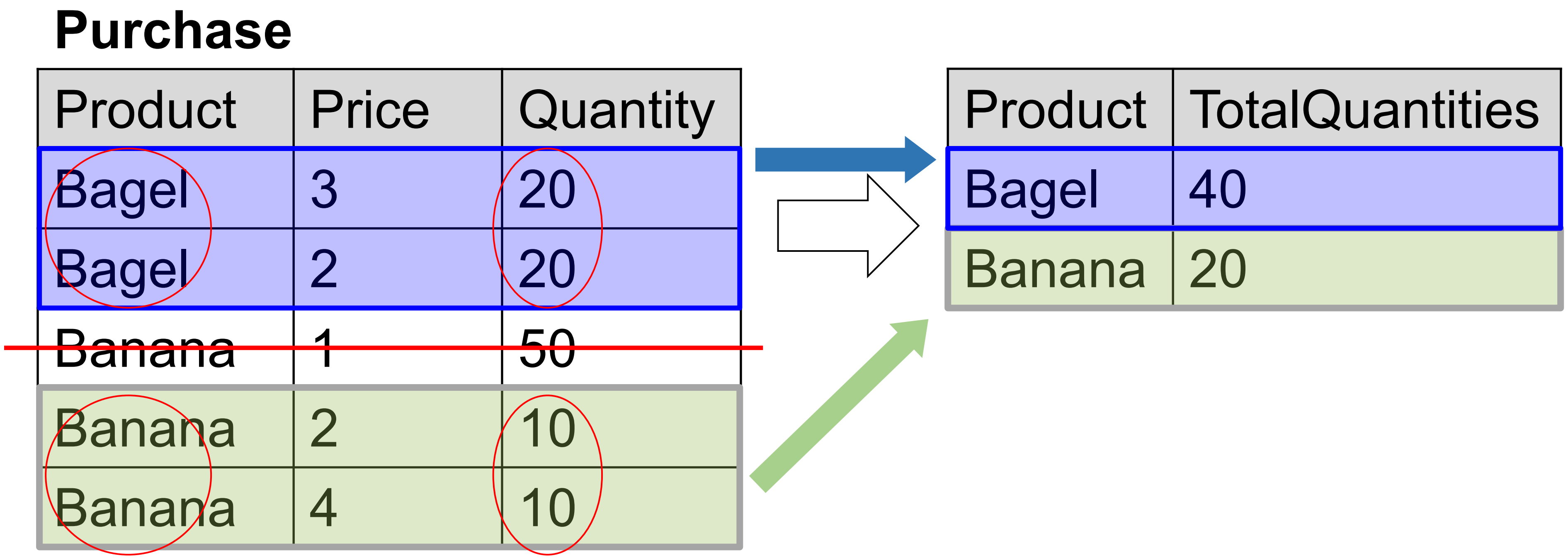
Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

→

Product	TotalQuantities
	?

```
SELECT      product, sum(quantity) as TotalQuantities
FROM        Purchase
WHERE       price > 1
GROUP BY    product, quantity
```

From → Where → Group By → Select



Purchase

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

Product	TotalQuantities
Bagel	40
Banana	20

```
SELECT     product, sum(quantity) as TotalQuantities
FROM       Purchase
WHERE      price > 1
GROUP BY   product, quantity
```

From → Where → Group By → Select

308

Purchase

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

Product	TotalQuantities
	?

```
SELECT     product, sum(quantity) as TotalQuantities
FROM       Purchase
WHERE      price > 1
GROUP BY   product, price
```

From → Where → Group By → Select

Purchase

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

Product	TotalQuantities
Bagel	20
Bagel	20
Banana	10
Banana	10

```
SELECT      product, sum(quantity) as TotalQuantities
FROM        Purchase
WHERE       price > 1
GROUP BY    product, price
```

From → Where → Group By → Select

308

Purchase

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

Product	TotalQuantities
	?

```
SELECT    product, quantity-1
FROM      Purchase
WHERE     price > 1
```

From → Where → Group By → Select

Purchase

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

Product	TotalQuantities
Bagel	19
Bagel	19
Banana	9
Banana	9

```
SELECT      product, quantity-1
FROM        Purchase
WHERE       price > 1
```

From → Where → Group By → Select

Purchase

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

Product	TotalQuantities
	?

```
SELECT     product, quantity--1
FROM       Purchase
WHERE      price > 1
```

From → Where → Group By → Select

Purchase

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

Product	TotalQuantities
Bagel	20
Bagel	20
Banana	10
Banana	10

```
SELECT      product, quantity--1
FROM        Purchase
WHERE       price > 1
```

?

From → Where → Group By → Select

Purchase

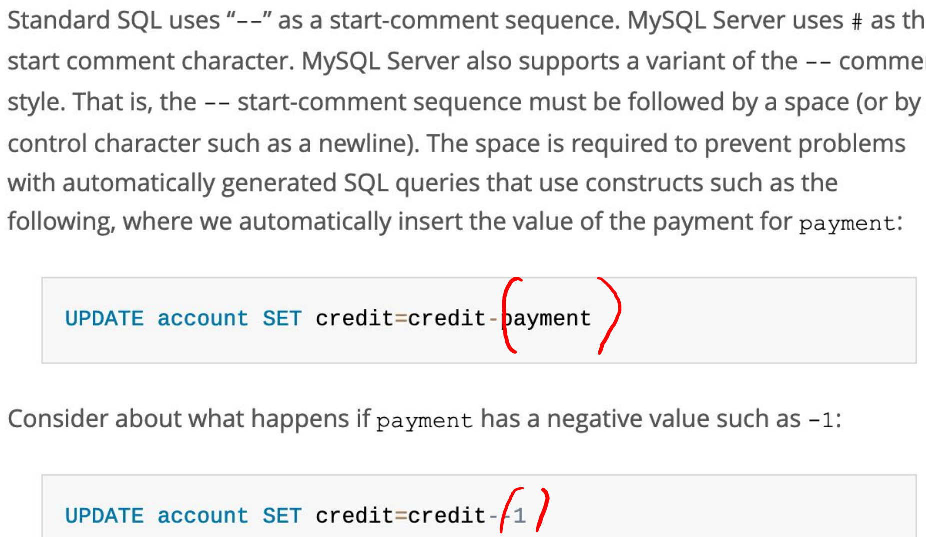
Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

Product	TotalQuantities
Bagel	20
Bagel	20
Banana	10
Banana	10

Interpreted as comment, contrast with "quantity-(-1)"!

```
SELECT    product, quantity--1
FROM      Purchase
WHERE     price > 1
```

Also contrast with MySQL: MySQL



Standard SQL uses "--" as a start-comment sequence. MySQL Server uses # as the start comment character. MySQL Server also supports a variant of the -- comment style. That is, the -- start-comment sequence must be followed by a space (or by a control character such as a newline). The space is required to prevent problems with automatically generated SQL queries that use constructs such as the following, where we automatically insert the value of the payment for payment:

```
UPDATE account SET credit=credit-(payment)
```

Consider about what happens if payment has a negative value such as -1:

```
UPDATE account SET credit=credit-(-1)
```

Screenshot from: https://dev.mysql.com/doc/refman/8.0/en/ansi-diff-comments.html
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

How to think about a query? Step-by-step, with intermediate results

Thinking like SQL

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

302

Product

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   cName
FROM     Product P, Company C
WHERE    manufacturer = cName
     and country = 'USA'
GROUP by cName
HAVING   count(*) >= 2
```

Q: Find all US companies that manufacture at least two different products.

Thinking like SQL

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

P

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

C

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

```
SELECT   cName
FROM     Product P, Company C
WHERE    manufacturer = cName
     and country = 'USA'
GROUP by cName
HAVING   count(*) >= 2
```

P

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

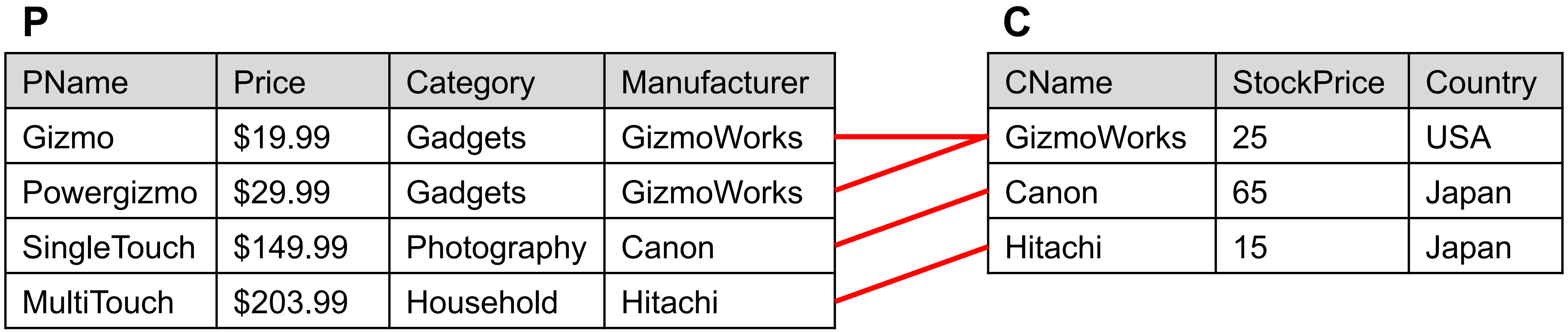
C

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

Q: Find all US companies that manufacture at least two different products.

Thinking like SQL

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



P

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

C

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

```
SELECT   cName
FROM     Product P, Company C
WHERE    manufacturer = cName
    and  country = 'USA'
GROUP by cName
HAVING   count(*) >= 2
```

P **C**

PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
SingleTouch	$149.99	Photography	Canon	GizmoWorks	25	USA
MultiTouch	$203.99	Household	Hitachi	GizmoWorks	25	USA
Gizmo	$19.99	Gadgets	GizmoWorks	Canon	65	Japan
Powergizmo	$29.99	Gadgets	GizmoWorks	Canon	65	Japan
SingleTouch	$149.99	Photography	Canon	Canon	65	Japan
MultiTouch	$203.99	Household	Hitachi	Canon	65	Japan
Gizmo	$19.99	Gadgets	GizmoWorks	Hitachi	15	Japan
Powergizmo	$29.99	Gadgets	GizmoWorks	Hitachi	15	Japan
SingleTouch	$149.99	Photography	Canon	Hitachi	15	Japan
MultiTouch	$203.99	Household	Hitachi	Hitachi	15	Japan

Q: Find all US companies that manufacture at least two different products.

Thinking like SQL

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

302

P				C		
PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
SingleTouch	$149.99	Photography	Canon	Canon	65	Japan
MultiTouch	$203.99	Household	Hitachi	Hitachi	15	Japan

```
SELECT  cName
FROM    Product P, Company C
WHERE   manufacturer = cName
    and country = 'USA'
GROUP by cName
HAVING  count(*) >= 2
```

Thinking like SQL

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

302

P				C		
PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
~~SingleTouch~~	~~$149.99~~	~~Photography~~	~~Canon~~	~~Canon~~	~~65~~	~~Japan~~
~~MultiTouch~~	~~$203.99~~	~~Household~~	~~Hitachi~~	~~Hitachi~~	~~15~~	~~Japan~~

```
SELECT  cName
FROM    Product P, Company C
WHERE   manufacturer = cName
   and  country = 'USA'
GROUP by cName
HAVING  count(*) >= 2
```

Thinking like SQL

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

302

P **C**

PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
~~SingleTouch~~	~~$149.99~~	~~Photography~~	~~Canon~~	~~Canon~~	~~65~~	~~Japan~~
~~MultiTouch~~	~~$203.99~~	~~Household~~	~~Hitachi~~	~~Hitachi~~	~~15~~	~~Japan~~

```
SELECT  cName
FROM    Product P, Company C
WHERE   manufacturer = cName
    and country = 'USA'
GROUP by cName
HAVING  count(*) >= 2
```

Thinking like SQL

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

P				C		
PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
~~SingleTouch~~	~~$149.99~~	~~Photography~~	~~Canon~~	~~Canon~~	~~65~~	~~Japan~~
~~MultiTouch~~	~~$203.99~~	~~Household~~	~~Hitachi~~	~~Hitachi~~	~~15~~	~~Japan~~

count(*) >= 2

```
SELECT  cName
FROM    Product P, Company C
WHERE   manufacturer = cName
    and country = 'USA'
GROUP by cName
HAVING  count(*) >= 2
```

Thinking like SQL

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

302

P | C

PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
~~SingleTouch~~	~~$149.99~~	~~Photography~~	~~Canon~~	~~Canon~~	~~65~~	~~Japan~~
~~MultiTouch~~	~~$203.99~~	~~Household~~	~~Hitachi~~	~~Hitachi~~	~~15~~	~~Japan~~

count(*) >= 2

```
SELECT   cName
FROM     Product P, Company C
WHERE    manufacturer = cName
    and  country = 'USA'
GROUP by cName
HAVING   count(*) >= 2
```

Cname
GizmoWorks

How to think about a query? Step-by-step, with intermediate results (a slight variant)

Thinking like SQL

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

302

Product

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   cName, category
FROM     Product P, Company C
WHERE    manufacturer = cName
     and country = 'USA'
GROUP by cName, category
HAVING   count(*) >= 2
```

Q: Find all US companies that manufacture at least two different products in the same category. Return Company name and category.

Thinking like SQL

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

P

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

C

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

```
SELECT   cName, category
FROM     Product P, Company C
WHERE    manufacturer = cName
     and  country = 'USA'
GROUP by cName, category
HAVING   count(*) >= 2
```

P

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

C

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

Thinking like SQL

302

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

P

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

C

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

```
SELECT   cName, category
FROM     Product P, Company C
WHERE    manufacturer = cName
    and  country = 'USA'
GROUP by cName, category
HAVING   count(*) >= 2
```

P

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

C

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

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

Thinking like SQL

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

302

P				C		
PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
SingleTouch	$149.99	Photography	Canon	Canon	65	Japan
MultiTouch	$203.99	Household	Hitachi	Hitachi	15	Japan

```
SELECT   cName, category
FROM     Product P, Company C
WHERE    manufacturer = cName
    and  country = 'USA'
GROUP by cName, category
HAVING   count(*) >= 2
```

Thinking like SQL

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

302

P **C**

PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
~~SingleTouch~~	~~$149.99~~	~~Photography~~	~~Canon~~	~~Canon~~	~~65~~	~~Japan~~
~~MultiTouch~~	~~$203.99~~	~~Household~~	~~Hitachi~~	~~Hitachi~~	~~15~~	~~Japan~~

```
SELECT   cName, category
FROM     Product P, Company C
WHERE    manufacturer = cName
     and country = 'USA'
GROUP by cName, category
HAVING   count(*) >= 2
```

Thinking like SQL

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

302

P

C

PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
~~SingleTouch~~	~~$149.99~~	~~Photography~~	~~Canon~~	~~Canon~~	~~65~~	~~Japan~~
~~MultiTouch~~	~~$203.99~~	~~Household~~	~~Hitachi~~	~~Hitachi~~	~~15~~	~~Japan~~

```
SELECT   cName, category
FROM     Product P, Company C
WHERE    manufacturer = cName
     and country = 'USA'
GROUP by cName, category
HAVING   count(*) >= 2
```

Thinking like SQL

302

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

P C

PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
~~SingleTouch~~	~~$149.99~~	~~Photography~~	~~Canon~~	~~Canon~~	~~65~~	~~Japan~~
~~MultiTouch~~	~~$203.99~~	~~Household~~	~~Hitachi~~	~~Hitachi~~	~~15~~	~~Japan~~

count(*) >= 2

```
SELECT  cName, category
FROM    Product P, Company C
WHERE   manufacturer = cName
    and country = 'USA'
GROUP by cName, category
HAVING  count(*) >= 2
```

Thinking like SQL

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

302

P | C

PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
~~SingleTouch~~	~~$149.99~~	~~Photography~~	~~Canon~~	~~Canon~~	~~65~~	~~Japan~~
~~MultiTouch~~	~~$203.99~~	~~Household~~	~~Hitachi~~	~~Hitachi~~	~~15~~	~~Japan~~

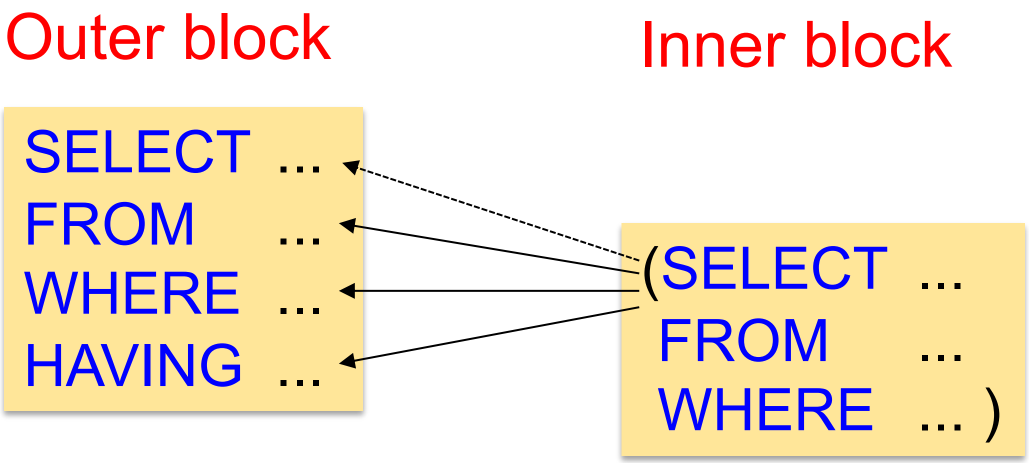
count(*) >= 2

```
SELECT   cName, category
FROM     Product P, Company C
WHERE    manufacturer = cName
     and country = 'USA'
GROUP by cName, category
HAVING   count(*) >= 2
```

Cname	category
GizmoWorks	Gadgets

Nested queries (Subqueries)

Subqueries = Nested queries



Outer block

SELECT ...
FROM ...
WHERE ...
HAVING ...

Inner block

(SELECT ...
FROM ...
WHERE ...)

We focus mainly on nestings in the WHERE and HAVING clauses, which are the most expressive type of nesting.

- We can nest queries because SQL is compositional:
  - Input & Output are represented as relations (multisets)
  - Subqueries also return relations; thus the output of one query can thus be used as the input to another (nesting)
- This is extremely powerful (think in terms of input/output)
- A complication: subqueries can be correlated (not just in-/output)

Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in a:
  - SELECT clause
  - FROM clause
  - WHERE clause
  - HAVING clause
- Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it’s impossible

not recommended

"table subqueries"

Subqueries in

SELECT clause (not recommended)

FROM clause

WHERE clause

HAVING clause

Subqueries in SELECT (not recommended)

Product (pName, price, category, cid)
Company (cid, cname, stockprice, country)

Q: For each product return the city where it is manufactured

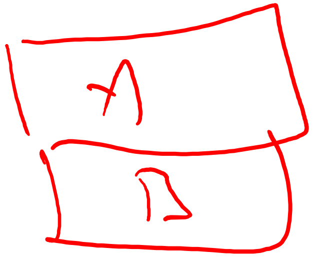
?

Subqueries in SELECT (not recommended)

Product (pName, price, category, cid)
Company (cid, cname, stockprice, country)

Q: For each product return the city where it is manufactured

```
SELECT  P.pname, (SELECT  C.city
                  FROM    Company C
                  WHERE   C.cid = P.cid)
FROM    Product P
```



What happens if the subquery returns more than one city ?

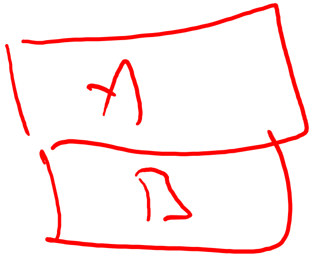
?

Subqueries in SELECT (not recommended)

Product (pName, price, category, cid)
Company (cid, cname, stockprice, country)

Q: For each product return the city where it is manufactured

```
SELECT  P.pname, (SELECT  C.city
                  FROM    Company C
                  WHERE   C.cid = P.cid)
FROM    Product P
```



What happens if the subquery returns more than one city ?

Runtime error ☹

→ "Scalar subquery": returns exactly one row with one column. See e.g.: https://www.postgresql.org/docs/current/sql-expressions.html#SQL-SYNTAX-SCALAR-SUBQUERIES

Subqueries in SELECT (not recommended)

Product (pName, price, category, cid)
Company (cid, cname, stockprice, country)

Q: For each product return the city where it is manufactured

```
SELECT  P.pname, (SELECT  C.city
                  FROM    Company C
                  WHERE   C.cid = P.cid)
FROM    Product P
```

Can you "unnest" the query?

?

Subqueries in SELECT (not recommended)

Product (pName, price, category, cid)
Company (cid, cname, stockprice, country)

Q: For each product return the city where it is manufactured

```
SELECT  P.pname, (SELECT  C.city
                  FROM    Company C
                  WHERE   C.cid = P.cid)
FROM    Product P
```

"unnesting the query"

```
SELECT  P.pname, C.city
FROM    Product P, Company C
WHERE   C.cid = P.cid
```

Whenever possible,
don't use nested queries

Subqueries in SELECT (not recommended)

Product (pName, price, category, cid)
Company (cid, cname, stockprice, country)

Q: Compute the number of products made by each company

?

Subqueries in SELECT (not recommended)

Product (pName, price, category, cid)
Company (cid, cname, stockprice, country)

Q: Compute the number of products made by each company

```
SELECT  C.cname, ( SELECT count (*)
                   FROM   Product P
                   WHERE  P.cid = C.cid)
FROM    Company C
```

Can you "unnest"
the query?

?

Subqueries in SELECT (not recommended)

Product (pName, price, category, cid)
Company (cid, cname, stockprice, country)

Q: Compute the number of products made by each company

```
SELECT  C.cname, ( SELECT count (*)
                   FROM   Product P
                   WHERE  P.cid = C.cid)
FROM    Company C
```

```
SELECT  C.cname, count(*)
FROM    Company C, Product P
WHERE   C.cid=P.cid
GROUP BY C.cname, C.cid
```

- We can unnest by using GROUP BY: that's more elegant ☺
- Notice the "C.cid" in the GROUP BY. Do we need it?

Subqueries in

SELECT clause

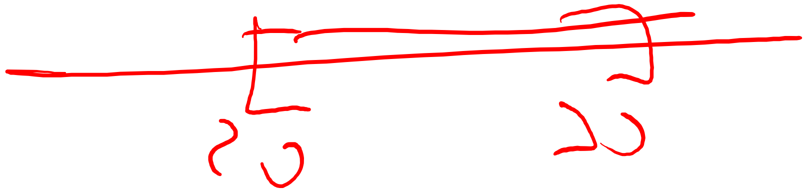
FROM clause (also called "derived tables")

WHERE clause

HAVING clause

Subqueries in FROM clause

316



Q: Find all products whose prices are > 20 and < 30!

```
SELECT  X.pname
FROM   ( SELECT  *
         FROM    Product as P
         WHERE   price >20 ) as X
WHERE   X.price < 30
```

Product

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

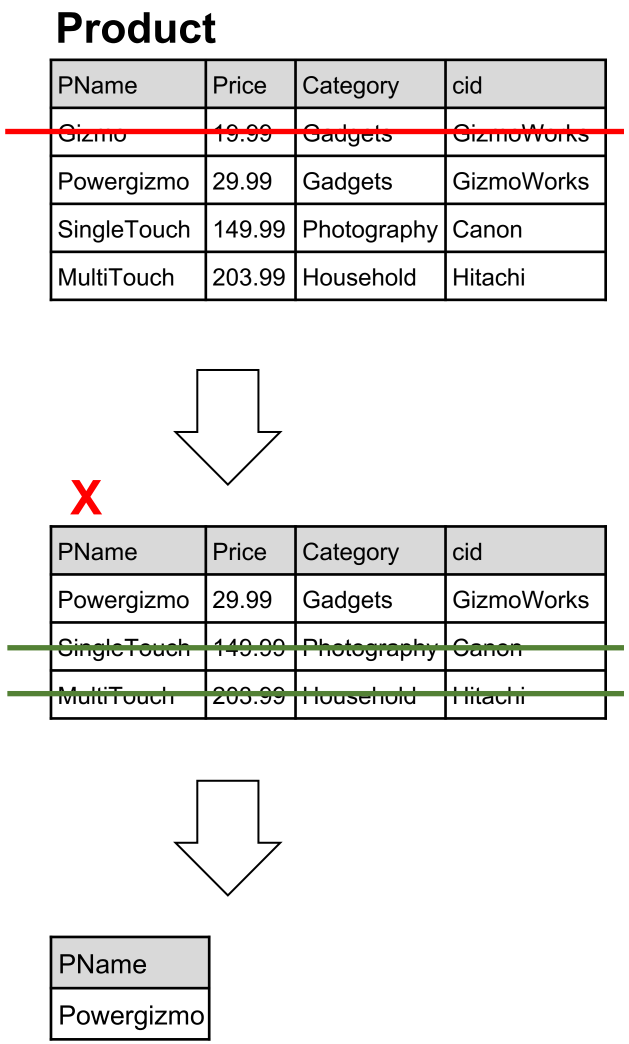
Subqueries in FROM clause

316

Composition: Think of a workflow ("data flow"): input / output

Q: Find all products whose prices are > 20 and < 30!

```
SELECT  X.pname
FROM   ( SELECT  *
         FROM    Product as P
         WHERE   price >20 ) as X
WHERE   X.price < 30
```



Product

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

X

PName	Price	Category	cid
Powergizmo	29.99	Gadgets	GizmoWorks
~~SingleTouch~~	~~149.99~~	~~Photography~~	~~Canon~~
~~MultiTouch~~	~~203.99~~	~~Household~~	~~Hitachi~~

PName
Powergizmo

Subqueries in FROM clause

Composition: Think of a workflow ("data flow"): input / output

Q: Find all products whose prices are > 20 and < 30!

Product

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

```
SELECT  *
FROM    Product as P
WHERE   price >20
```

X

PName	Price	Category	cid
Powergizmo	29.99	Gadgets	GizmoWorks
~~SingleTouch~~	~~149.99~~	~~Photography~~	~~Canon~~
~~MultiTouch~~	~~203.99~~	~~Household~~	~~Hitachi~~

```
SELECT  X.pname
FROM    X
WHERE   X.price < 30
```

PName
Powergizmo

Subqueries in FROM clause

316

Product (pname, price, cid)
Company (cid, cname, city)

Q: Find all products whose prices are > 20 and < 30!

```
SELECT  X.pname
FROM  ( SELECT  *
        FROM    Product as P
        WHERE   price >20 ) as X
WHERE   X.price < 30
```

Can you rewrite the query without nestings?

?

Product

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

X

PName	Price	Category	cid
Powergizmo	29.99	Gadgets	GizmoWorks
~~SingleTouch~~	~~149.99~~	~~Photography~~	~~Canon~~
~~MultiTouch~~	~~203.99~~	~~Household~~	~~Hitachi~~

PName
Powergizmo

Subqueries in FROM clause

```
Product (pname, price, cid)
Company (cid, cname, city)
```

Q: Find all products whose prices are > 20 and < 30!

```
SELECT  X.pname
FROM    ( SELECT   *
          FROM     Product as P
          WHERE    price >20 ) as X
WHERE   X.price < 30
```

No need to write this query as nested query either ☺

```
SELECT  pname
FROM    Product
WHERE   price > 20 and price < 30
```

Product

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

⇩

X

PName	Price	Category	cid
Powergizmo	29.99	Gadgets	GizmoWorks
~~SingleTouch~~	~~149.99~~	~~Photography~~	~~Canon~~
~~MultiTouch~~	~~203.99~~	~~Household~~	~~Hitachi~~

⇩

PName
Powergizmo

Subqueries in

SELECT clause

FROM clause (also called "derived tables")

WHERE clause

HAVING clause

Subqueries in FROM clause = Derived tables

Purchase

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

?

Q1: For each product, find total quantities (sum of quantities) purchased.

Subqueries in FROM clause = Derived tables

308

Purchase

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

Product	TQ
Bagel	40
Banana	70

Q1: For each product, find total quantities (sum of quantities) purchased.

? in SQL

Subqueries in FROM clause = Derived tables

308

Purpose

Purchase

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

⇨

Product	TQ
Bagel	40
Banana	70

⇨

?

Q1: For each product, find total quantities (sum of quantities) purchased.

Q2: Find the maximal total quantities purchased across all products [MTQ]

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

Subqueries in FROM clause = Derived tables

Purchase

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

X

Product	TQ
Bagel	40
Banana	70

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

Q2: Find the maximal total quantities purchased across all products [MTQ]

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

? in SQL

Subqueries in FROM clause = Derived tables

Purchase

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

→

X

Product	TQ
Bagel	40
Banana	70

→

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

```
(
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product            ) X
```

Q2: Find the maximal total quantities purchased across all products.

```
SELECT MAX(TQ) as MTQ
FROM X
```

Subqueries in FROM clause = Derived tables

Purchase

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

```
SELECT MAX(TQ) as MTQ
FROM (SELECT product, SUM(quantity) as TQ
      FROM Purchase
      GROUP BY product) X
```

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

Q2: Find the maximal total quantities purchased across all products.

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

```
SELECT MAX(TQ) as MTQ
FROM X
```

Is the Having Clause really necessary?

Purchase

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

Product	SumQ	MaxP
Banana	?	?
Bagel	?	?

```
SELECT    product,
          sum(quantity) as SumQ,
          max(price) as MaxP
FROM      Purchase
GROUP BY  product
```

Is the Having Clause really necessary?

Purchase

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

Product	SumQ	MaxP
Banana	70	4
Bagel	40	3

```
SELECT     product,
           sum(quantity) as SumQ,
           max(price) as MaxP
FROM       Purchase
GROUP BY product
```

Is the Having Clause really necessary?

Purchase

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

⇨

Product	SumQ	MaxP
Banana	70	4
Bagel	40	3

```
SELECT    product,
          sum(quantity) as SumQ,
          max(price) as MaxP
FROM      Purchase
GROUP BY  product
HAVING    sum(quantity) > 50
```

?

Is the Having Clause really necessary?

Purchase

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

Product	SumQ	MaxP
Banana	70	4
~~Bagel~~	~~40~~	~~3~~

```
SELECT    product,
          sum(quantity) as SumQ,
          max(price) as MaxP
FROM      Purchase
GROUP BY  product
HAVING    sum(quantity) > 50
```

Can you rewrite the query without the HAVING clause?

?

Is the Having Clause really necessary?

Purchase

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

```
SELECT     product,
           sum(quantity) as SumQ,
           max(price) as MaxP
FROM       Purchase
GROUP BY   product
HAVING     sum(quantity) > 50
```

X

Product	SumQ	MaxP
Banana	70	4
~~Bagel~~	~~40~~	~~3~~

Can you rewrite the query without the HAVING clause?

?

```
SELECT *
FROM X




WHERE SumQ > 50
```

Is the Having Clause really necessary?

Purchase

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

Product	SumQ	MaxP
Banana	70	4
~~Bagel~~	~~40~~	~~3~~

```
SELECT    product,
          sum(quantity) as SumQ,
          max(price) as MaxP
FROM      Purchase
GROUP BY  product
HAVING    sum(quantity) > 50
```

Can you rewrite the query without the HAVING clause?

?

```
SELECT *
FROM
  ( SELECT    product,
              sum(quantity) as SumQ,
              max(price) as MaxP
    FROM      Purchase
    GROUP BY  product
  ) as X
WHERE SumQ > 50
```

Subqueries in

SELECT clause

FROM clause

WHERE clause (including IN, ANY, ALL)

HAVING clause

Subqueries in WHERE clause

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

⇨ ?

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

⇨ ?

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

⇨ ?

Subqueries in WHERE clause

What do these queries return?

R

a
1
2

W

a	b
2	0
3	0
4	0

```
SELECT  a
FROM    R
WHERE   a IN
        (SELECT a FROM W)
```

a
2

Since 2 is in the set (bag) (2, 3, 4)

```
SELECT  a
FROM    R
WHERE   a < ANY
        (SELECT a FROM W)
```

?

```
SELECT  a
FROM    R
WHERE   a < ALL
        (SELECT a FROM W)
```

?

Subqueries in WHERE clause

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

a
2

Since 2 is in the set (bag) (2, 3, 4)

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

a
1
2

Since 1 and 2 are < than at least one ("any") of 2, 3 or 4

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

?

Subqueries in WHERE clause

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

SQLlite does not support "ANY" or "ALL" ☹

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

a
2

Since 2 is in the set (bag) (2, 3, 4)

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

a
1
2

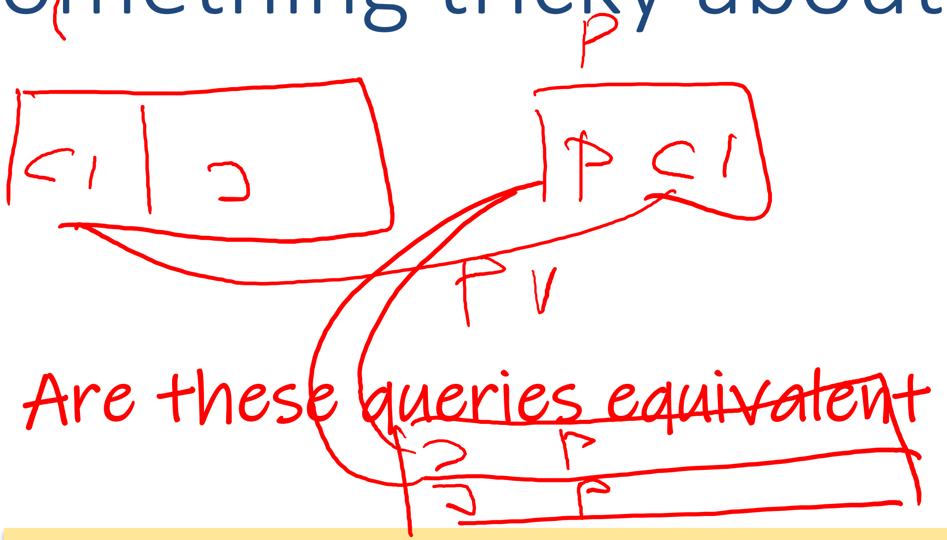
Since 1 and 2 are < than at least one ("any") of 2, 3 or 4

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

a
1

Since 1 is < than each ("all") of 2, 3, and 4

Something tricky about Nested Queries



Are these queries equivalent ?

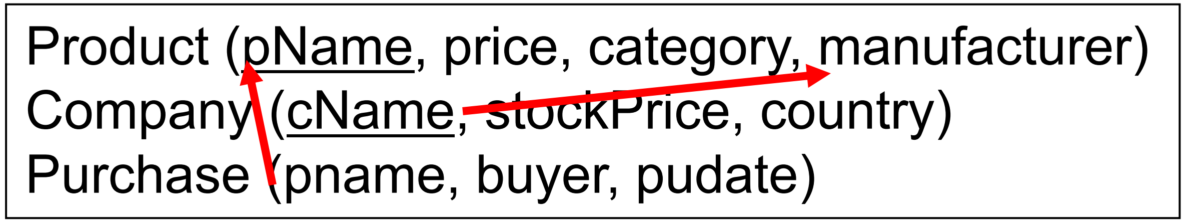
Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)
Purchase (pname, buyer, pudate)

```
SELECT C.country
FROM   Company C
WHERE  C.cname  IN (
SELECT P.manufacturer
FROM   Purchase PU, Product P
WHERE  P.pname = PU.pname
  AND  PU.buyer = 'Joe B')
```

```
SELECT C.country
FROM   Company C,
       Product P,
       Purchase PU
WHERE  C.cname = P.manufacturer
  AND  P.pname = PU.pname
  AND  PU.buyer = 'Joe B'
```

Notice that "equivalent" means give the same results over any database!

Something tricky about Nested Queries



Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)
Purchase (pname, buyer, pudate)

Are these queries equivalent ?

Beware of duplicates!

```
SELECT C.country
FROM   Company C
WHERE  C.cname  IN (
SELECT P.manufacturer
FROM   Purchase PU, Product P
WHERE  P.pname = PU.pname
  AND  PU.buyer = 'Joe B')
```

```
SELECT C.country
FROM   Company C,
       Product P,
       Purchase PU
WHERE  C.cname = P.manufacturer
  AND  P.pname = PU.pname
  AND  PU.buyer = 'Joe B'
```

Something tricky about Nested Queries

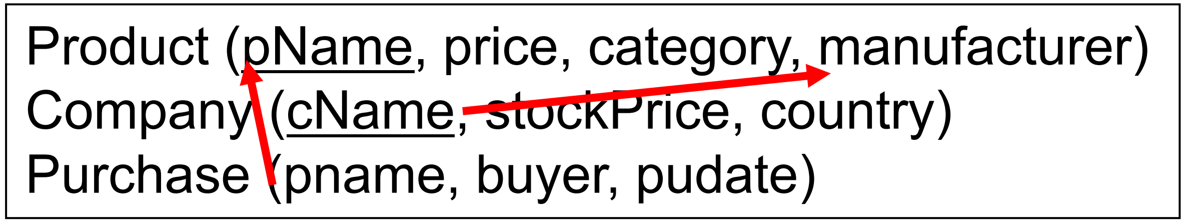
Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)
Purchase (pname, buyer, pudate)

Are they now equivalent ?

```
SELECT C.country
FROM   Company C
WHERE  C.cname  IN (
SELECT P.manufacturer
FROM   Purchase PU, Product P
WHERE  P.pname = PU.pname
  AND  PU.buyer = 'Joe B')
```

```
SELECT DISTINCT C.country
FROM   Company C,
       Product P,
       Purchase PU
WHERE  C.cname = P.manufacturer
  AND  P.pname = PU.pname
  AND  PU.buyer = 'Joe B'
```

Something tricky about Nested Queries



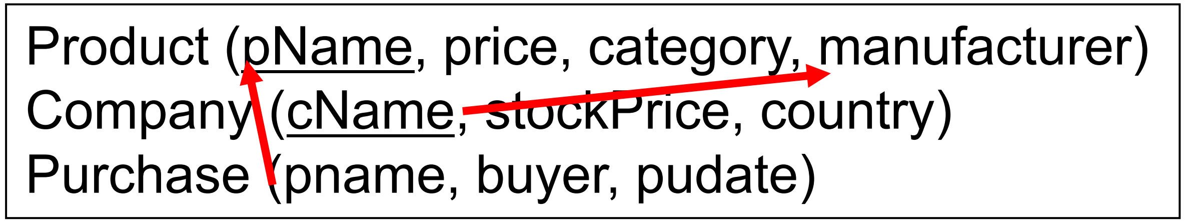
Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)
Purchase (pname, buyer, pudate)

What about now? Are they now equivalent ?

```
SELECT DISTINCT C.country
FROM   Company C
WHERE  C.cname  IN (
SELECT P.manufacturer
FROM   Purchase PU, Product P
WHERE  P.pname = PU.pname
  AND  PU.buyer = 'Joe B')
```

```
SELECT DISTINCT C.country
FROM   Company C,
       Product P,
       Purchase PU
WHERE  C.cname = P.manufacturer
  AND  P.pname = PU.pname
  AND  PU.buyer = 'Joe B'
```

Something tricky about Nested Queries



Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)
Purchase (pname, buyer, pudate)

Now they are equivalent:

```
SELECT DISTINCT C.country
FROM   Company C
WHERE  C.cname  IN (
SELECT P.manufacturer
FROM   Purchase PU, Product P
WHERE  P.pname = PU.pname
  AND  PU.buyer = 'Joe B')
```

```
SELECT DISTINCT C.country
FROM   Company C,
       Product P,
       Purchase PU
WHERE  C.cname = P.manufacturer
  AND  P.pname = PU.pname
  AND  PU.buyer = 'Joe B'
```

Correlated subqueries (in WHERE clause)

Correlated subqueries

- In all previous cases, the nested subquery in the inner select block could be entirely evaluated before processing the outer select block.
  - Recall the "compositional" nature of relational queries (input/output)
  - This is no longer the case for correlated nested queries.
- Whenever a condition in the WHERE clause of a nested query references some column of a table declared in the outer query, the two queries are said to be correlated.
  - The nested query is then evaluated once for each tuple (or combination of tuples) in the outer query (that's the conceptual evaluation strategy)

Correlated subquery (existential ∃)

316

slightly different product database!

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

Using IN: Set / Bag membership

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product P
                   WHERE   P.price < 25)
```

Is this a correlated nested query ?

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

Correlated subquery (existential ∃)

316

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

slightly different product database!

Q_1: Find all companies that make some product(s) with price < 25

Using IN: Set / Bag membership

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product P
                   WHERE   P.price < 25)
```

Not a correlated nested query!

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN ( 1, 2 )
```

Inner query does not reference outer query! You could first evaluate the inner query by itself.

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

Using EXISTS: TRUE if the subquery's result is NOT empty

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   EXISTS ( SELECT  *
                 FROM    Product P
                 WHERE   P.cid = C.cid
                 and     P.price < 25)
```

Is this a correlated nested query

?

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

Using EXISTS: TRUE if the subquery's result is NOT empty

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   EXISTS ( SELECT  *
                 FROM    Product P
                 WHERE   P.cid = C.cid
                 and     P.price < 25)
```

This is a correlated nested query! Notice the additional join condition referencing a relation from the outer query.

Recall our conceptual evaluation strategy!

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

Using ANY (also SOME): again set / bag comparison

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   25 > ANY ( SELECT  price
                   FROM    Product P
                   WHERE   P.cid = C.cid)
```

But do we really need to write this query as nested query ?

SQLlite does not support "ANY" ☹

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

```
SELECT  DISTINCT C.cname
FROM    Company C, Product P
WHERE   C.cid = P.cid
and     P.price < 25
```

We did not need to write nested queries;
we can "unnest" it!

Existential quantifiers are easy ☺

Correlated subquery (universal ∀)

316

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

~~Q_1: Find all companies that make some product(s) with price < 25~~

Q_2: Find all companies that make only products with price < 25

≡ Q_2: Find all companies for which all products have price < 25

≡ Q_2: Find all companies that do not have any product with price >= 25

Universal quantifiers are more complicated ! ☹
(Think about the companies that should not be returned)

All three formulations are equivalent: a company with no product will be returned!

Correlated subquery (universal ∀ = not exists ∄)

Q_2: Find all companies that make only products with price < 25

Step 1: Q_2': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN        ( SELECT P.cid
                          FROM    Product P
                          WHERE   P.price >= 25)
```

First think about the companies that should not be returned!

Step 2: Q_2: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE   C.cid NOT IN ( SELECT P.cid
                       FROM    Product P
                       WHERE   P.price >= 25)
```

Correlated subquery (universal ∀ = not exists ∄)

316

Q_2: Find all companies that make only products with price < 25

Step 1: Q_2': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  EXISTS         ( SELECT  *
                        FROM     Product P
                        WHERE   C.cid = P.cid
                        and       P.price >= 25)
```

First think about the companies that should not be returned!

Step 2: Q_2: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  NOT EXISTS ( SELECT  *
                        FROM     Product P
                        WHERE   C.cid = P.cid
                        and       P.price >= 25)
```

Correlated subquery (universal ∀ = not exists ∄)

Q$_2$: Find all companies that make only products with price < 25

Step 1: Q$_2$': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  25 <= ANY    ( SELECT P.price
                      FROM    Product P
                      WHERE  C.cid = P.cid)
```

First think about the companies that should not be returned!

Step 2: Q$_2$: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  25 > ALL     ( SELECT P.price
                      FROM    Product P
                      WHERE  C.cid = P.cid)
```

A natural question

Q_2: Find all companies that make only products with price < 25

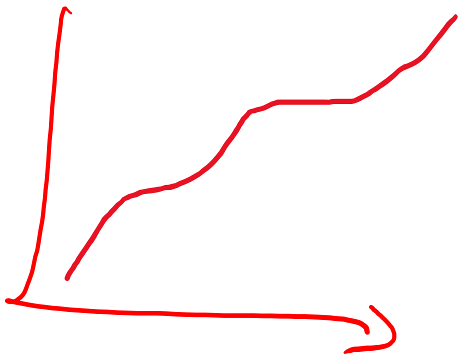
- How can we unnest (no GROUP BY) the universal quantifier query ?

```
SELECT   ...
FROM     ...
WHERE    ...
```

?

Source: Dan Suciu

Queries that must be nested



1. Definition: A query Q is monotone if:
   - Whenever we add tuples to one or more of the tables…
   - … the answer to the query cannot contain fewer tuples
2. Fact: all unnested queries are monotone
   - Proof: using the "nested for loops" semantics
3. Fact: Query with universal quantifier is not monotone
   - Add one tuple violating the condition. Then "all" returns fewer tuples
4. Consequence: we cannot unnest a query with a universal quantifier

Source: Dan Suciu

The person/bar/drinks example (formerly drinkers/bars/beers, courtesy Jeff Ullman)

Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink)

Challenge: write these in SQL.

?

Find persons that frequent some bar that serves some drink they like.

Find persons that frequent only bars that serve some drink they like.

Find persons that frequent some bar that serves only drinks they like.

Find persons that frequent only bars that serve only drinks they like.
(= Find persons who like all drinks that are served in all the bars they visit.)
(= Find persons for which there does not exist a bar they frequent that serves a drink they do not like.)

The person/bar/drinks example (formerly drinkers/bars/beers, courtesy Jeff Ullman)

Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink)

Challenge: write these in SQL.
Solutions: http://demo.queryvis.com

Tip: SQL based on First-Order Logic (FOL)

Find persons that frequent some bar that serves some drink they like.

$$x: \quad \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)$$

Find persons that frequent only bars that serve some drink they like.

$$x: \quad \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \land \text{Likes}(x,z))$$

Find persons that frequent some bar that serves only drinks they like.

$$x: \quad \exists y.\ \text{Frequents}(x, y) \land \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Find persons that frequent only bars that serve only drinks they like.
(= Find persons who like all drinks that are served in all the bars they visit.)
(= Find persons for which there does not exist a bar they frequent that serves a drink they do not like.)

$$x: \quad \forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$
$$x: \quad \nexists y.\ \text{Frequents}(x, y) \land (\exists z.\text{Serves}(y,z) \land \nexists z2.\ \text{Likes}(x,2z))$$