Updated 1/21/2022

Topic 1: Data models and query languages
Unit 1: SQL
Lecture 2

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

https://northeastern-datalab.github.io/cs7240/sp22/

1/21/2022

Pre-class conversations

- Last class recapitulation
- Any questions on class procedures?
  - Piazza vs Canvas announcements?
  - Hybrid for next Tuesday
  - "Class scribes": You will continue to see some "minimum examples" today in class; a note about slide quality
  - Already installed Postgres?
  - The downsides of no regular homeworks

- Today:
  - SQL continued

The "Surfer Analogy" for time management

Source: http://stwww.surfermag.com/files/2013/10/Yak_Charlie-970x646.jpg

Outline: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Nulls & Outer joins
  - Top-k

Table Alias (Tuple Variables)

Person (pName, address, works_for)
University (uName, address)

```
SELECT  DISTINCT pName, address
FROM    Person, University
WHERE   works_for = uName
```

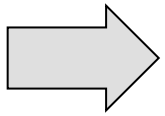
What will this query return ?

Table Alias (Tuple Variables)

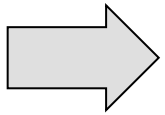
Person (pName, address, works_for)
University (uName, address)

which address?
Error!

```
SELECT  DISTINCT pName, address
FROM    Person, University
WHERE   works_for = uName
```



```
SELECT  DISTINCT pName, University.address
FROM    Person, University
WHERE   Person.works_for = University.uName
```

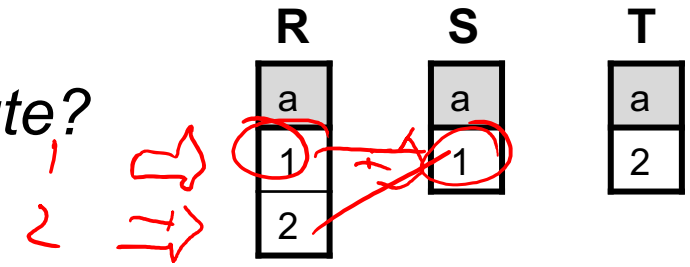


```
SELECT  DISTINCT X.pName, Y.address
FROM    Person as X, University Y
WHERE   X.works_for = Y.uName
```

Notice that the use of "as" is not necessary, it is optional !!

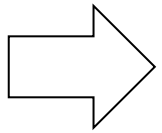
Using the Formal Semantics

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



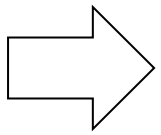
What do these queries compute?

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



?

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



?

Using the Formal Semantics

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

What do these queries compute?

R

a
1
2

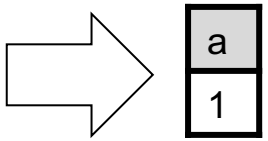
S

a
1

T

a
2

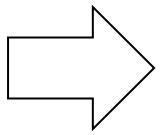
```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



a
1

Returns R ∩ S (intersection)

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



?

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), T(a)

What do these queries compute?

R

a
1
2

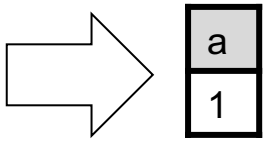
S

a
1

T

a
2

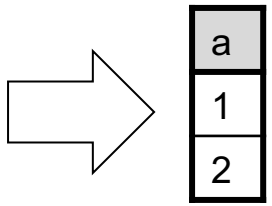
```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



a
1

Returns R ∩ S
(intersection)

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



a
1
2

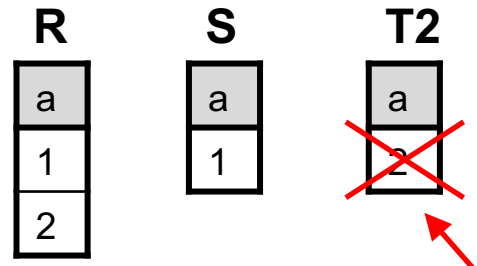
Returns R ∩ (S ∪ T)
if S ≠ ∅ and T ≠ ∅

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

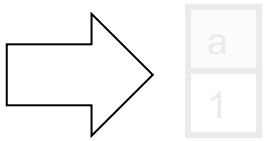
Using the Formal Semantics

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

What do these queries compute?




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



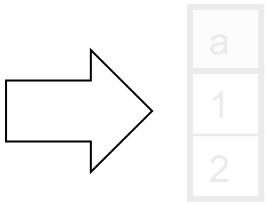
a
1

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
```



a
1
2

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

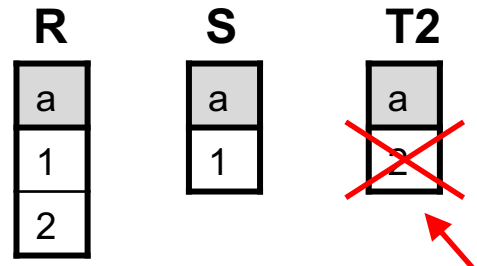
?

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

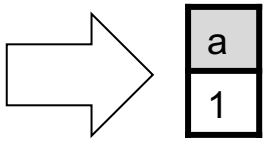
Using the Formal Semantics

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

What do these queries compute?



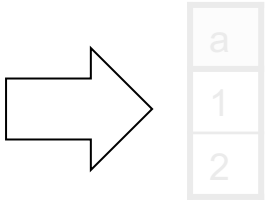

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



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
```



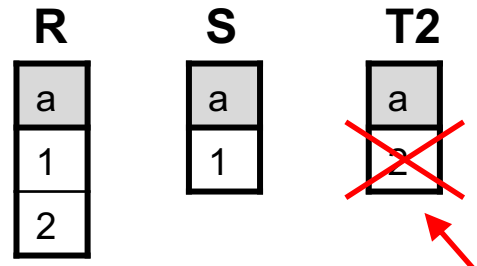
Returns R ∩ (S ∪ T) if S ≠ ∅ and T ≠ ∅

?

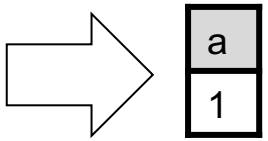
Using the Formal Semantics

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

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. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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 ---

Proces 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)

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.

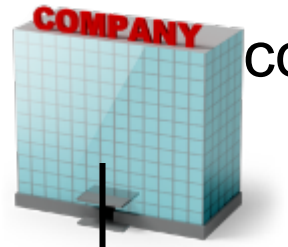
```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and P.price < 20
    and P.price > 25
    and P.manufacturer = cName
```

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

What do we actually want?

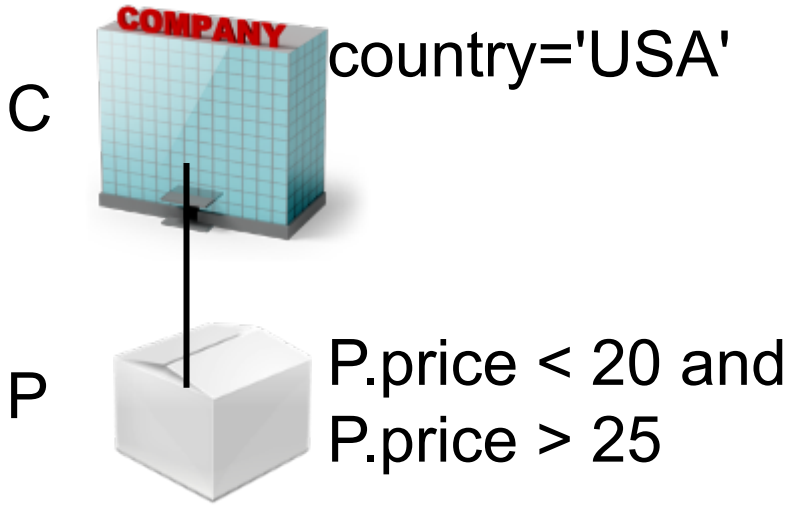
?

not possible!
→ Empty result

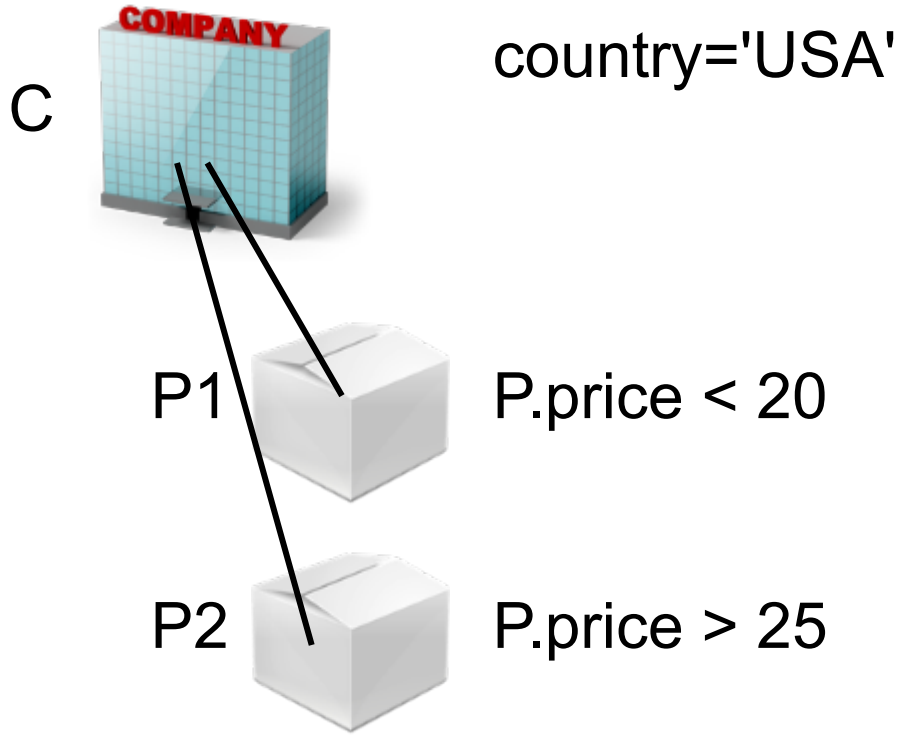
Quiz: Answer 1 vs. what we actually want


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.



not possible!
→ Empty result



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?

?

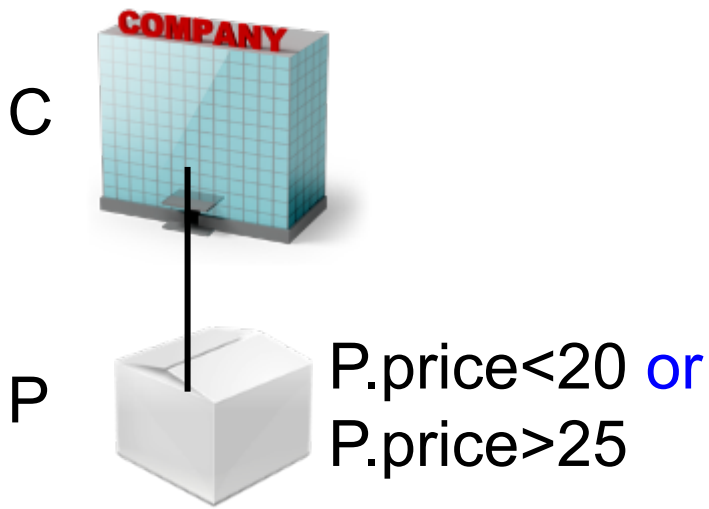
Quiz: Answer 2


Product (<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.

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

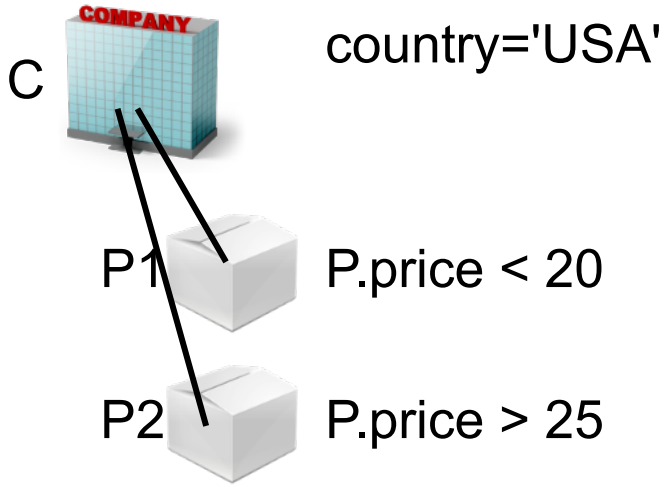
```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and (P.price < 20
    or   P.price > 25)
    and  P.manufacturer = cName
```



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


Product (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 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
```

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

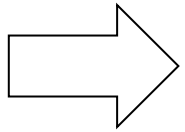
Outline: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Nulls & Outer joins
  - Top-k

Grouping and Aggregation

Purchase

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



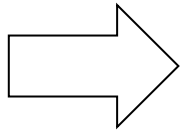
?

Q: For each product, find Total Quantities (TQ = sum of quantities) purchased, for all products with price >1.

Grouping and Aggregation

Purchase

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



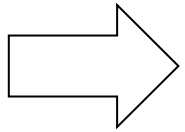
Product	TQ
Bagel	?
Banana	?

Q: For each product, find Total Quantities (TQ = sum of quantities) purchased, for all products with price >1.

Grouping and Aggregation

Purchase

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



Product	TQ
Bagel	40
Banana	20

Q: For each product, find Total Quantities (TQ = sum of quantities) purchased, for all products with 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

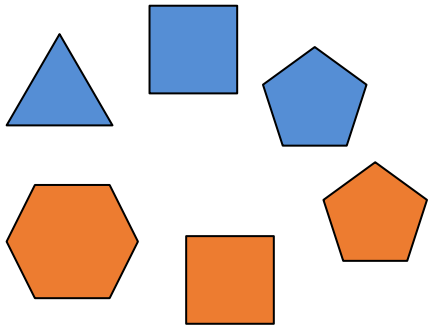
Product	TQ
Bagel	40
Banana	20

Select contains
- grouped attributes
- and aggregates

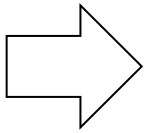
```
4  SELECT    product, sum(quantity) as TQ
1  FROM      Purchase
2  WHERE     price > 1
3  GROUP BY  product
```

Groupings illustrated with colored shapes

group by color

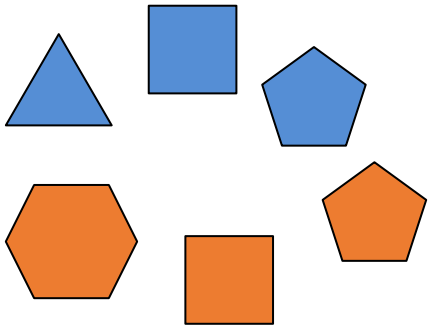


```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

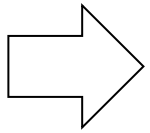


?

group by numc (# of corners)



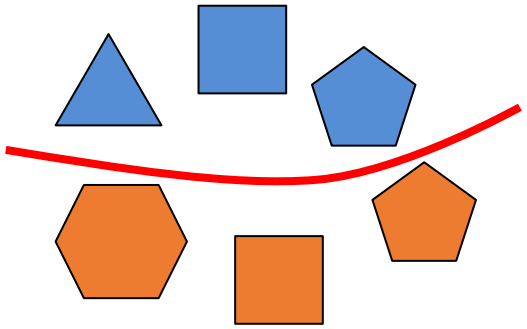
```
SELECT  numc
FROM    Shapes
GROUP BY numc
```



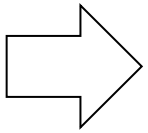
?

Groupings illustrated with colored shapes

group by color

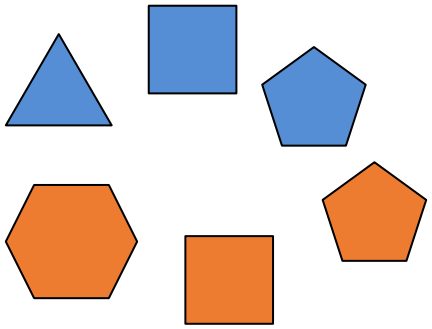


```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

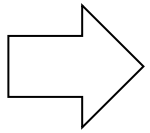


?

group by numc (# of corners)



```
SELECT  numc
FROM    Shapes
GROUP BY numc
```



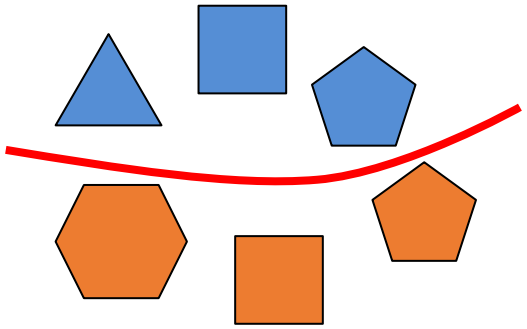
?

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

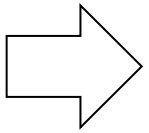
Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

group by color

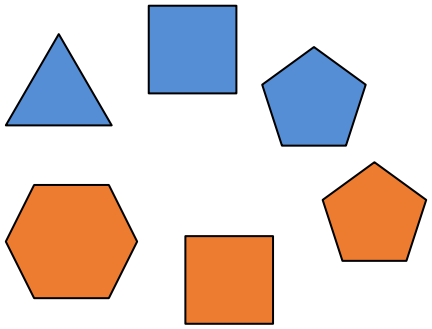


```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

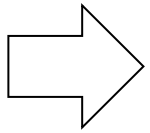


color	anc
blue	4
orange	5

group by numc (# of corners)



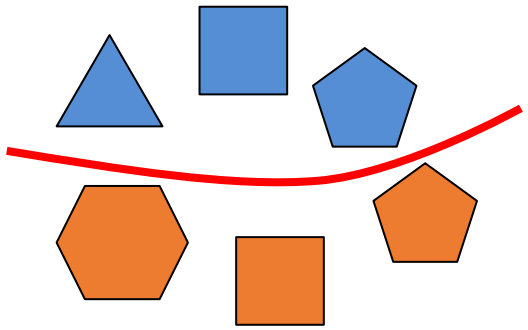
```
SELECT  numc
FROM    Shapes
GROUP BY numc
```



Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

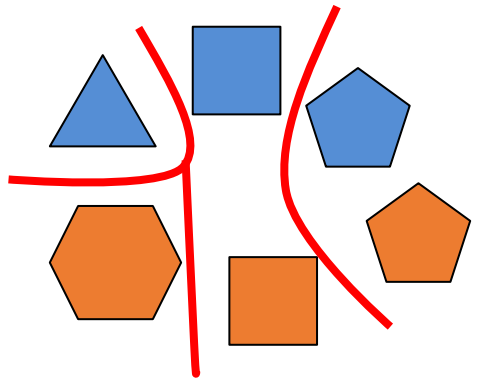
group by color



```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

color	anc
blue	4
orange	5

group by numc (# of corners)



```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

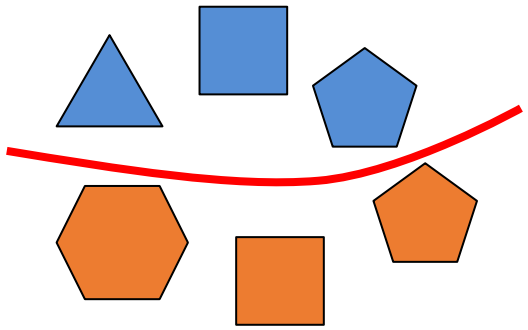
numc
3
4
5
6

Without group by ?

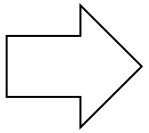
Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

group by color

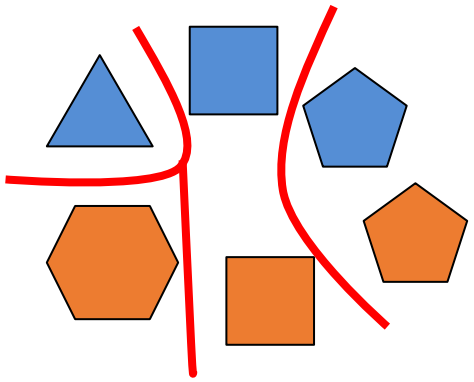


```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

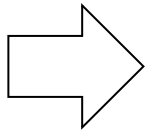


color	anc
blue	4
orange	5

group by numc (# of corners)



```
SELECT  numc
FROM    Shapes
GROUP BY numc
```



numc
3
4
5
6

Same as:

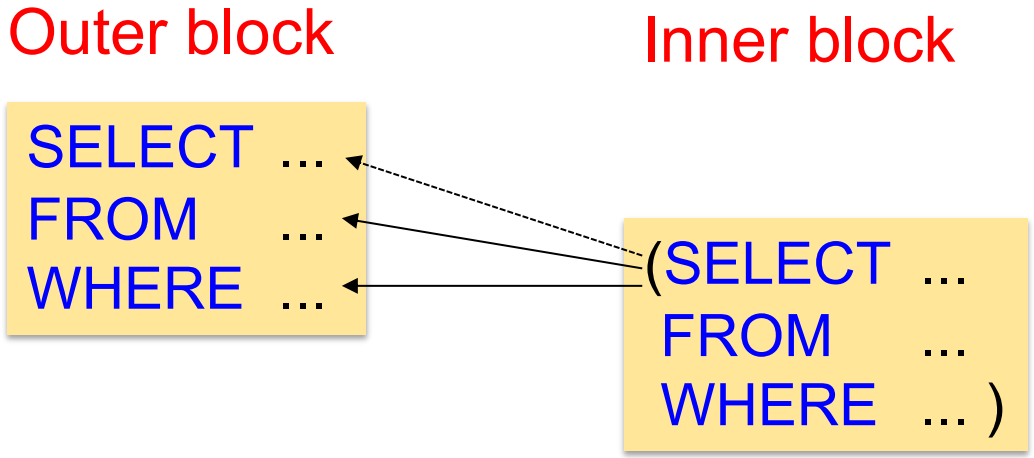
```
SELECT  DISTINCT numc
FROM    Shapes
```

Without group by!

Outline: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Nulls & Outer joins
  - Top-k

Subqueries = Nested queries



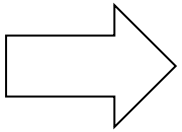
- We focus mainly on nestings in the WHERE clause, which is the most expressive type of nesting.
- But we start with nesting in FROM clause which are also called "derived tables"

- 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, yet can also quickly get complicated

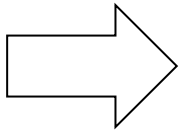
Subqueries in FROM clause = Derived tables

Purchase

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



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.

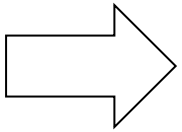
```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

?

Subqueries in FROM clause = Derived tables

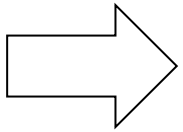
Purchase

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.

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

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

?

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
```

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.

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

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

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

Common Table Expressions (CTE): WITH clause

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

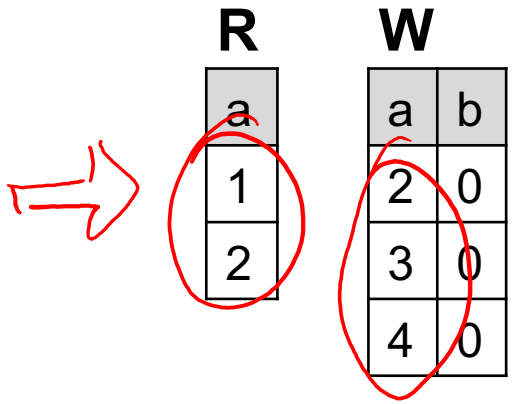
CTE (Common Table Expression)

Query using CTE

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

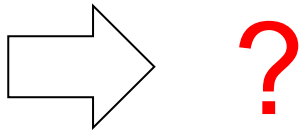
The WITH clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

Subqueries in WHERE clause



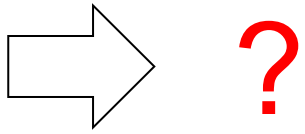
What do these queries return?

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



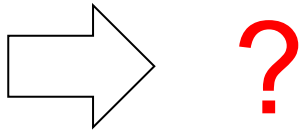
?

```
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

R

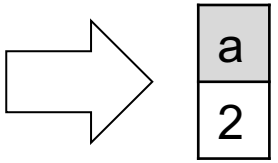
a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

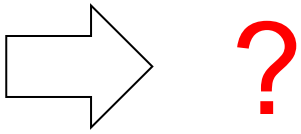
```
SELECT  a
FROM    R
WHERE   a IN
        (SELECT a from W)
```



a
2

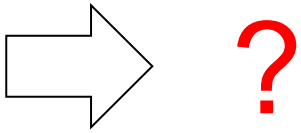
Since 2 is in the set (bag) (2, 3, 4)

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



?

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



?

Subqueries in WHERE clause

R

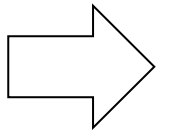
a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

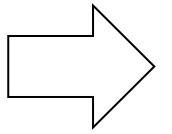
```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a from W)
```



a
2

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

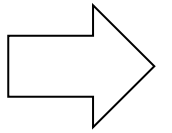
```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a from W)
```



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

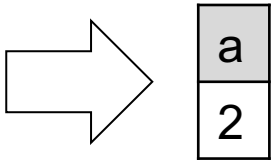
a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

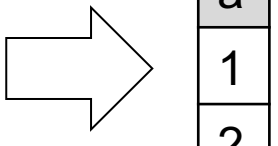
```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a from W)
```



a
2

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

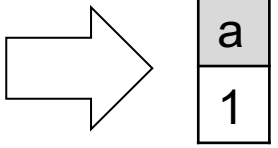
```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a from W)
```



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

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
  - 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 ∃)

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)
```

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

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

?

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 ∀)

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 ∄)

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

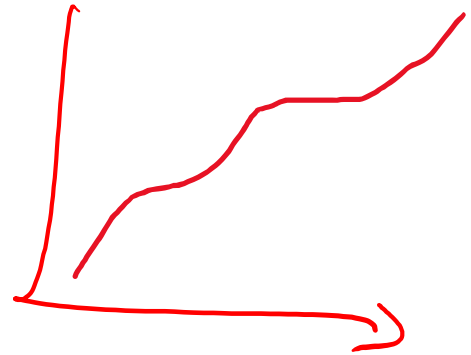
Q_2: Find all companies that make only products with price < 25

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

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

?

Queries that must be nested



- 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
- Fact: all unnested queries are monotone
  - Proof: using the "nested for loops" semantics
- Fact: Query with universal quantifier is not monotone
  - Add one tuple violating the condition. Then "all" returns fewer tuples
- Consequence: we cannot unnest a query with a universal quantifier

Understanding nested queries with QueryVis

The sailors database

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

340

Sailor

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance $S3$ of Sailors

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance $R2$ of Reserves

Boat

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance $B1$ of Boats

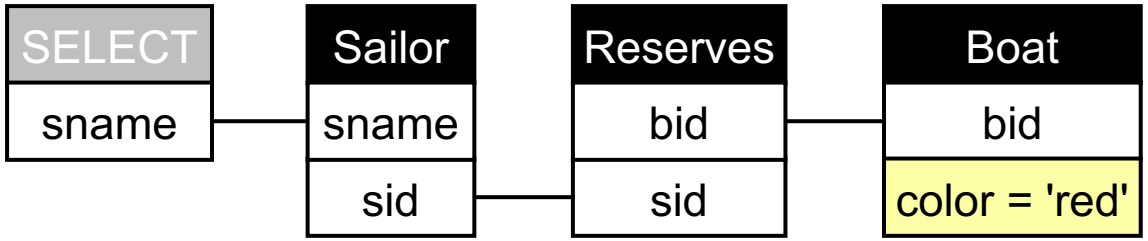
Nested query 1

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'))
```



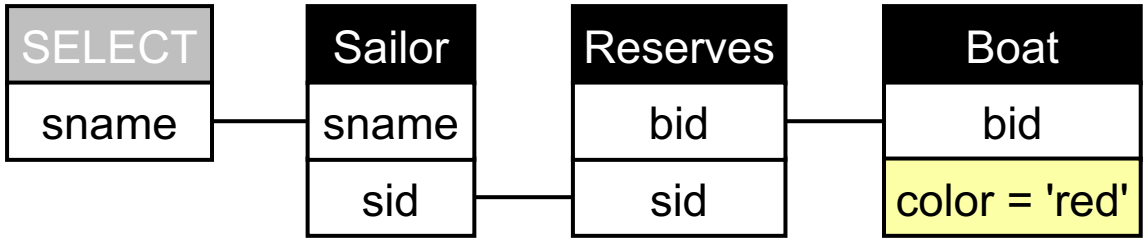
Nested query 1

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: Find the names of sailors who have reserved a red boat.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'))
```



{S.sname | ∃S∈Sailor.(∃R∈Reserves.(R.sid=S.sid ∧ ∃B∈Boat.(B.bid=R.bid ∧ B.color='red')))}

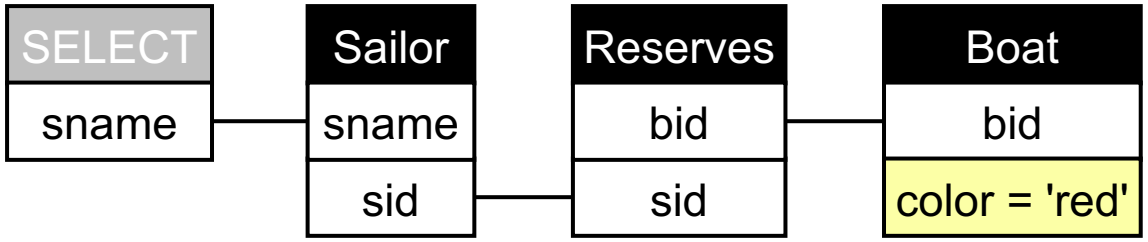
Nested query 1

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: Find the names of sailors who have reserved a red boat.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE EXISTS
      (SELECT R.sid
      FROM Reserves R
      WHERE R.sid=S.sid
      AND EXISTS
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'
            AND B.bid=R.bid))
```



This is an alternative way to write the previous query with EXISTS and correlated nested queries that matches the Relational Calculus below.

$$\{S.sname \mid \exists S \in Sailor.(\exists R \in Reserves.(R.sid=S.sid \wedge \exists B \in Boat.(B.bid=R.bid \wedge B.color='red')))\}$$

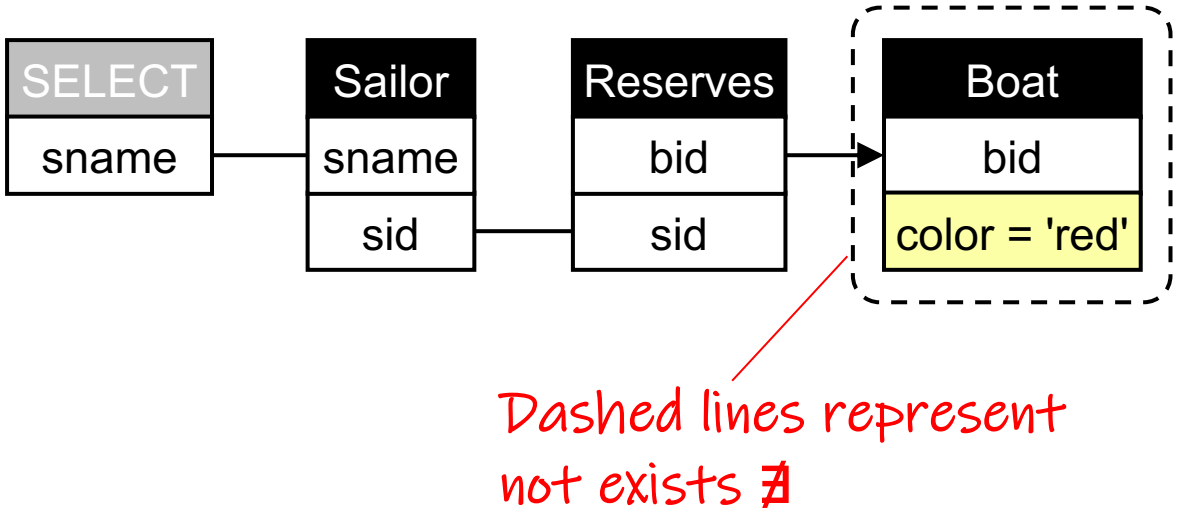
Nested query 2

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid not IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'))
```



{S.sname | ∃S∈Sailor.(∃R∈Reserves.(R.sid=S.sid ∧ ∄B∈Boat.(B.bid=R.bid ∧ B.color='red')))}

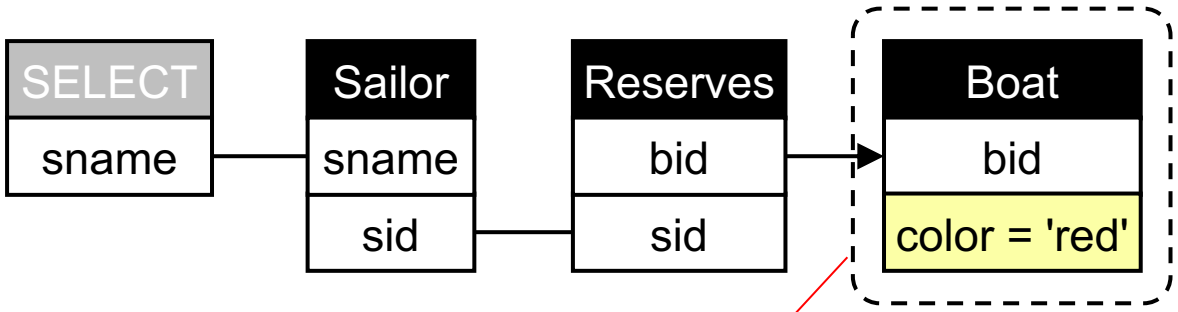
Nested query 2

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: Find the names of sailors who have reserved a boat that is not red.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid not IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'))
```



Dashed lines represent not exists ∄

They must have reserved at least one boat in another color. They can also have reserved a red boat in addition.

{S.sname | ∃S∈Sailor.(∃R∈Reserves.(R.sid=S.sid ∧ ∄B∈Boat.(B.bid=R.bid ∧ B.color='red')))}

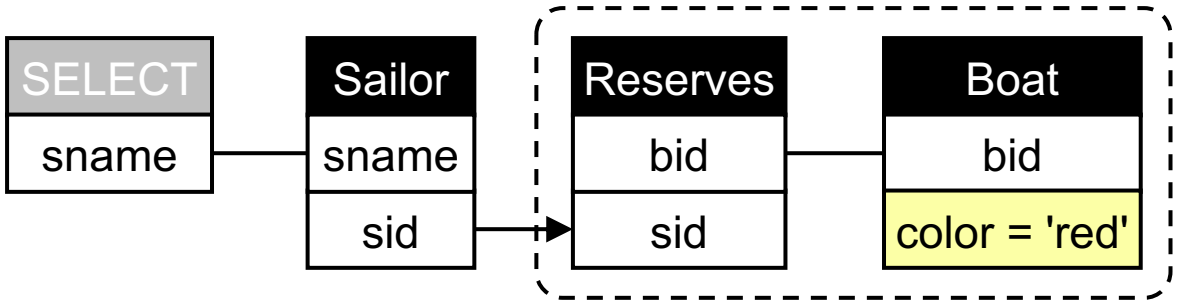
Nested query 3

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid not IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'))
```



{S.sname | ∃S∈Sailor.(∄R∈Reserves.(R.sid=S.sid ∧ ∃B∈Boat.(B.bid=R.bid ∧ B.color='red')))}

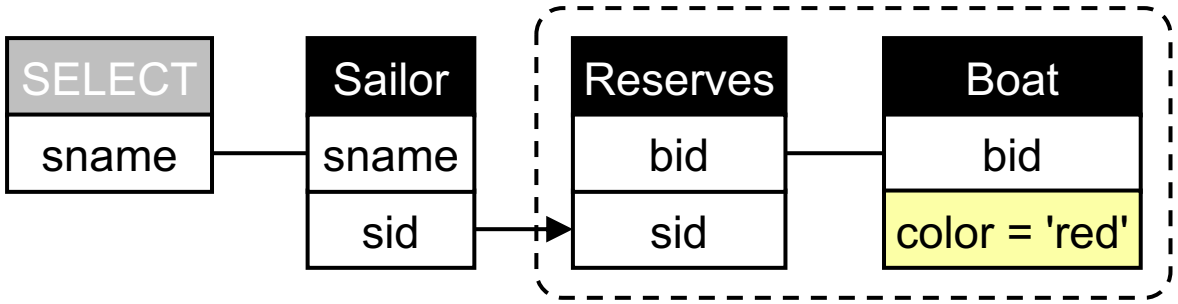
Nested query 3

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: Find the names of sailors who have not reserved a red boat.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid not IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'))
```



They can have reserved 0 or more boats in another color, but must not have reserved any red boat.

$$\{S.sname \mid \exists S \in Sailor.(\nexists R \in Reserves.(R.sid=S.sid \wedge \exists B \in Boat.(B.bid=R.bid \wedge B.color='red')))\}$$

Quiz: Dustin?


340

Sailor

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance $S3$ of Sailors

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance $R2$ of Reserves

Boat

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance $B1$ of Boats

Should Dustin be in the output of each of the two queries?

Q2: Find the names of sailors who have reserved a boat that is not red.

Q3: Find the names of sailors who have not reserved a red boat.

?

Quiz: Dustin?


340

Sailor

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance $S3$ of Sailors

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance $R2$ of Reserves

Boat

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance $B1$ of Boats

Should Dustin be in the output of each of the two queries?

Q2: Find the names of sailors who have reserved a boat that is not red. Yes!

Q3: Find the names of sailors who have not reserved a red boat. No!

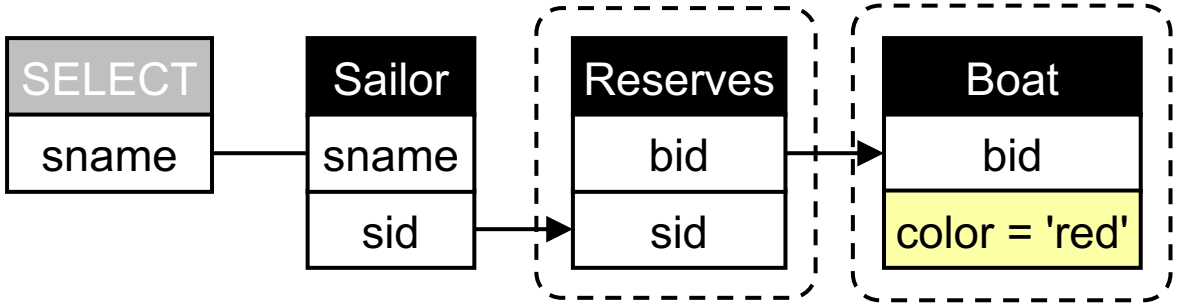
Nested query 4

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid not IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid not IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'))
```



{S.sname | ∃S∈Sailor.(∄R∈Reserves.(R.sid=S.sid ∧ ∄B∈Boat.(B.bid=R.bid ∧ B.color='red')))}

Nested query 4

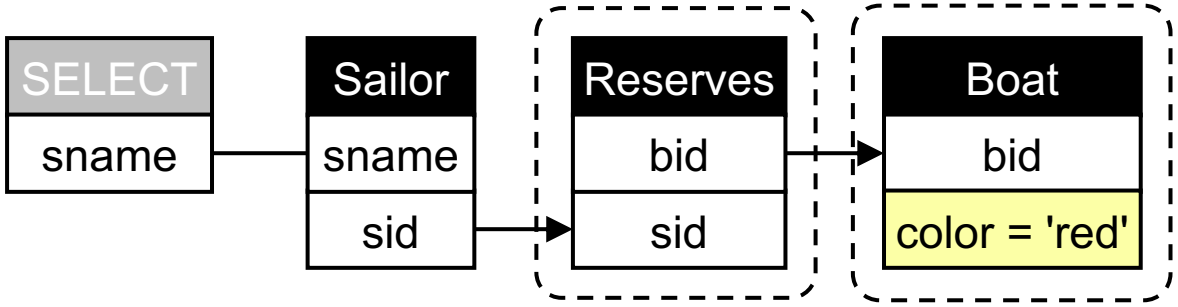
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find the names of sailors who have reserved only red boats

Q: Find the names of sailors who have not reserved a boat that is not red.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid not IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid not IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'))
```



They can have reserved 0 or more boats in red, just no other color.

0 boats ok

$$\{S.sname \mid \exists S \in Sailor.(\nexists R \in Reserves.(R.sid=S.sid \wedge \nexists B \in Boat.(B.bid=R.bid \wedge B.color='red')))\}$$

Nested query 4 (another variant)

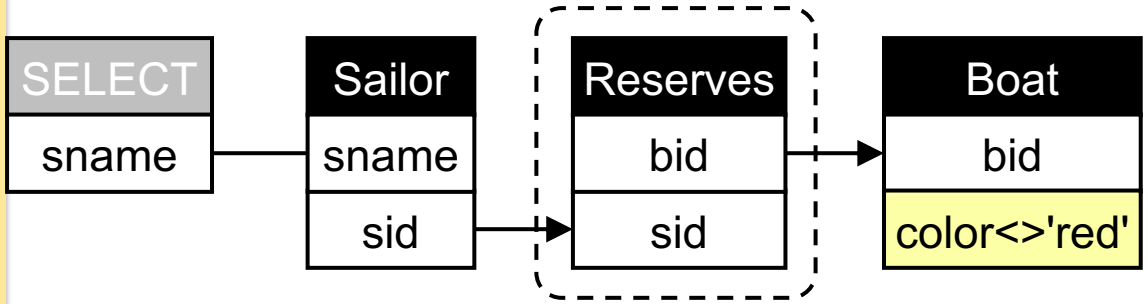
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find the names of sailors who have reserved only red boats

Q: Find the names of sailors who have not reserved a boat that is not red.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid not IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color<>'red'))
```



They can have reserved 0 or more boats in red, just no other color.

$$\{S.sname \mid \exists S \in Sailor.(\nexists R \in Reserves.(R.sid=S.sid \wedge \exists B \in Boat.(B.bid=R.bid \wedge B.color<>'red')))\}$$

Nested query 4 (universal)

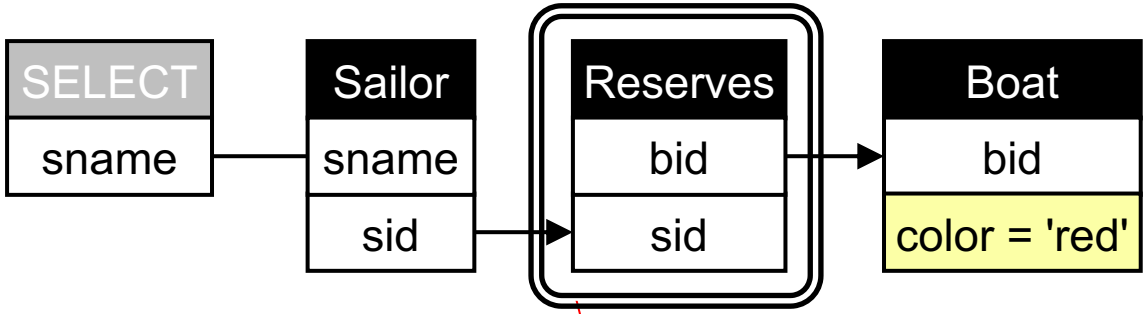
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find the names of sailors who have reserved only red boats

Q: Find the names of sailors who have not reserved a boat that is not red.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid not IN
      (SELECT R.sid
      FROM Reserves R
      WHERE R.bid not IN
            (SELECT B.bid
            FROM Boat B
            WHERE B.color='red'))
```



Double lines represent for all ∀

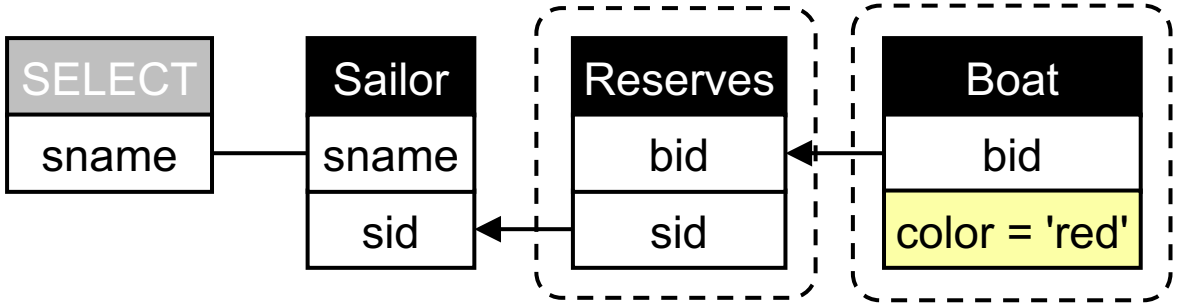
{S.sname | ∃S∈Sailor.(∀R∈Reserves.(R.sid=S.sid → ∃B∈Boat.(B.bid=R.bid ∧ B.color='red')))}

Nested query 5

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE not exists
      (SELECT B.bid
      FROM Boat B
      WHERE B.color = 'red'
      AND not exists
            (SELECT R.bid
            FROM Reserves R
            WHERE R.bid = B.bid
            AND R.sid = S.sid))
```



$\{S.sname \mid \exists S \in Sailor.(\nexists B \in Boat.(B.color='red' \wedge \nexists R \in Reserves.(B.bid=R.bid \wedge R.sid=S.sid)))\}$

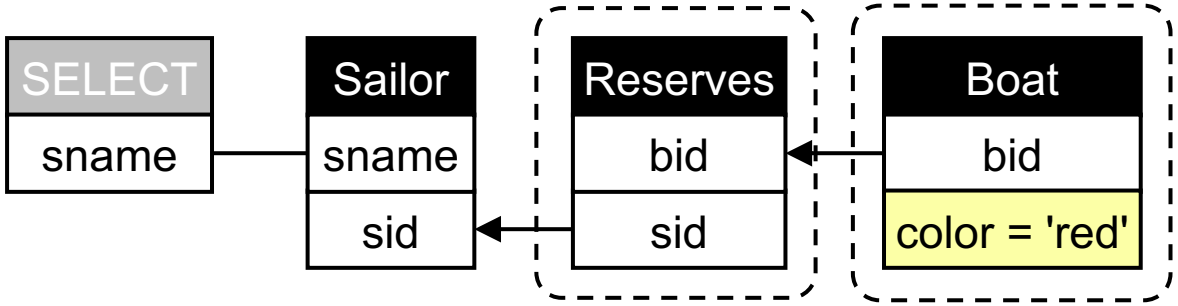
Nested query 5

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

= Find the names of sailors who have reserved all red boats

Q: Find the names of sailors so there is no red boat that is not reserved by the sailor.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE not exists
      (SELECT B.bid
      FROM Boat B
      WHERE B.color = 'red'
      AND not exists
            (SELECT R.bid
            FROM Reserves R
            WHERE R.bid = B.bid
            AND R.sid = S.sid))
```



I don't know of a way to write that query with IN instead of EXISTS and without an explicit cross product between sailors and red boats. (More on that in a moment and also later when we discuss this query in relational algebra.)

$$\{S.sname \mid \exists S \in Sailor.(\nexists B \in Boat.(B.color='red' \wedge \nexists R \in Reserves.(B.bid=R.bid \wedge R.sid=S.sid)))\}$$

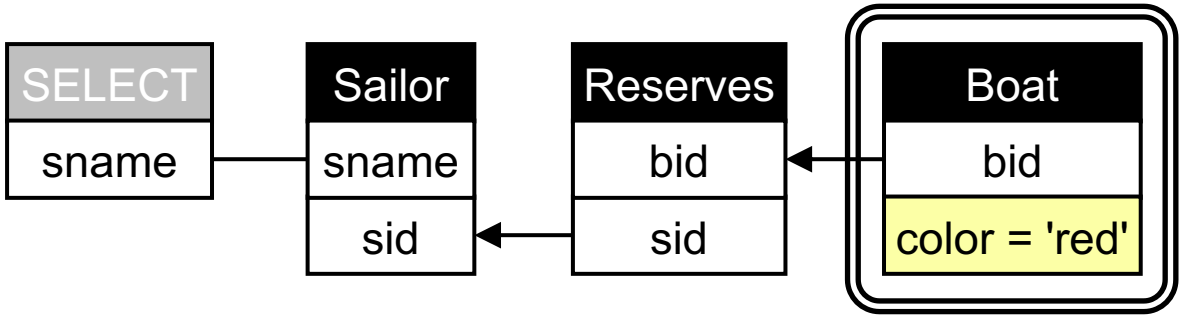
Nested query 5 (universal)

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

= Find the names of sailors who have reserved all red boats

Q: Find the names of sailors so there is no red boat that is not reserved by the sailor.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE not exists
      (SELECT B.bid
      FROM Boat B
      WHERE B.color = 'red'
      AND not exists
            (SELECT R.bid
            FROM Reserves R
            WHERE R.bid = B.bid
            AND R.sid = S.sid))
```



{S.sname | ∃S∈Sailor.(∀B∈Boat.(B.color='red' → ∃R∈Reserves.(B.bid=R.bid ∧ R.sid=S.sid)))}

Nested query 5 (w/o correlation)

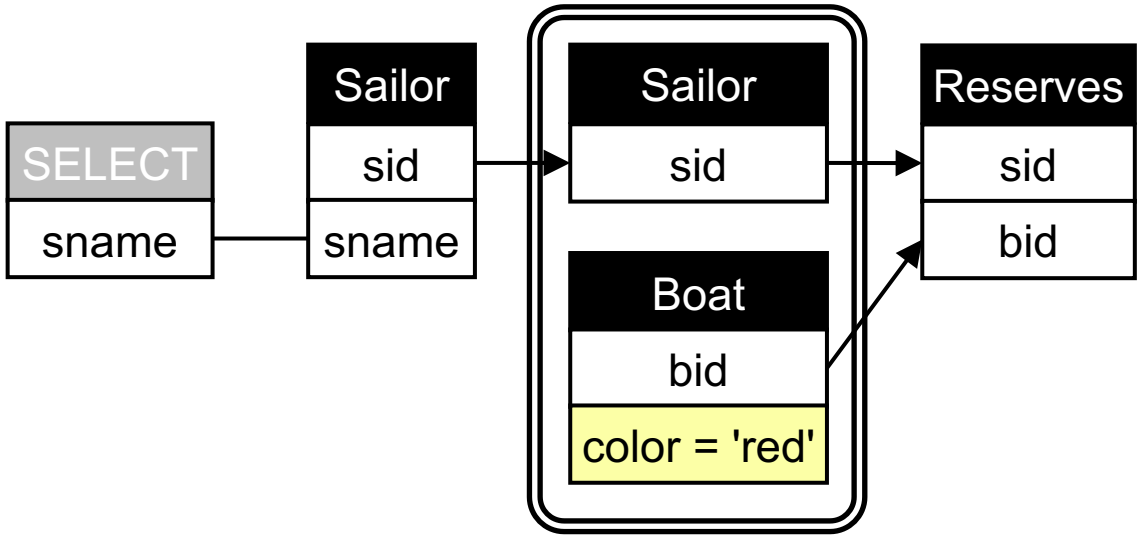
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find the names of sailors who have reserved all red boats

Q: Find the names of sailors so there is no red boat that is not reserved by the sailor.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid not in
      (SELECT S2.sid
      FROM Sailor S2, Boat B
      WHERE B.color = 'red'
      AND (S2.sid, B.bid) not in
            (SELECT R.sid, R.bid
            FROM Reserves R))
```



{S.sname | ∃S∈Sailor.(∀S2∈Sailor ∀B∈Boat.(B.color='red' ∧ S2.sid=S.sid → ∃R∈Reserves.(B.bid=R.bid ∧ S2.sid=R.sid)))}

Nested query 5 (w/o correlation)

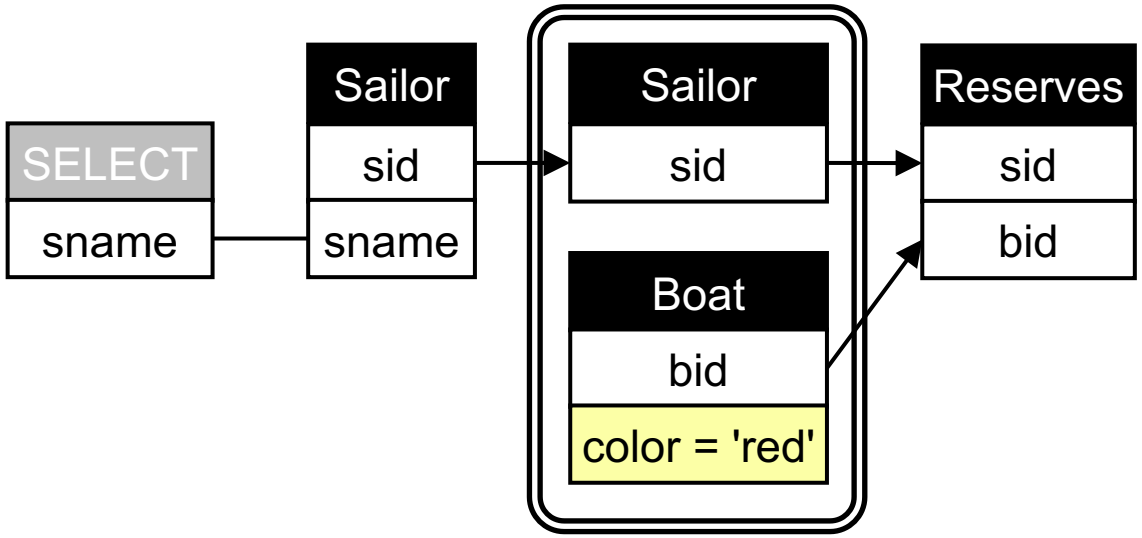
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find the names of sailors who have reserved all red boats

Q: Find the names of sailors so there is no red boat that is not reserved by the sailor.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE not exists
      (SELECT *
      FROM Sailor S2, Boat B
      WHERE B.color = 'red'
      AND S.sid = S2.sid
      AND not exists
            (SELECT *
            FROM Reserves R
            WHERE B.bid=R.bid
            AND S2.sid = R.sid))
```



{S.sname | ∃S∈Sailor.(∀S2∈Sailor ∀B∈Boat.(B.color='red' ∧ S2.sid=S.sid → ∃R∈Reserves.(B.bid=R.bid ∧ S2.sid=R.sid)))}

Towards SQL patterns

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

	Sailors who **have not** reserved a red boat	Sailors who reserved **only** red boats	Sailors who reserved **all** red boats
SQL	SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS(SELECT * FROM Reserves R, Boat B WHERE R.sid = S.sid AND R.bid = B.bid AND B.color = 'red')	SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS(SELECT * FROM Reserves R WHERE R.sid = S.sid AND NOT EXISTS(SELECT * FROM Boat B WHERE B.color = 'red' AND R.bid = B.bid))	SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS(SELECT * FROM Boat B WHERE B.color = 'red' AND NOT EXISTS(SELECT * FROM Reserves R WHERE R.bid = B.bid AND R.sid = S.sid))

Towards SQL patterns

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

	Sailors who **have not** reserved a red boat	Sailors who reserved **only** red boats	Sailors who reserved **all** red boats
SQL	see below	see below	see below
QV	see below	see below	see below

SQL — Sailors who have not reserved a red boat

```
SELECT  DISTINCT S.sname
FROM    Sailor S
WHERE   NOT EXISTS(
        SELECT  *
        FROM    Reserves R, Boat B
        WHERE   R.sid = S.sid
        AND     R.bid = B.bid
        AND     B.color = 'red')
```

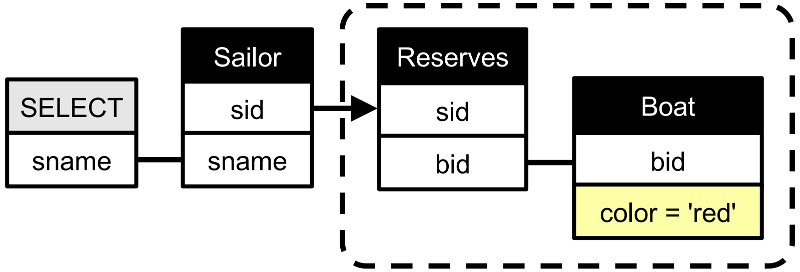
SQL — Sailors who reserved only red boats

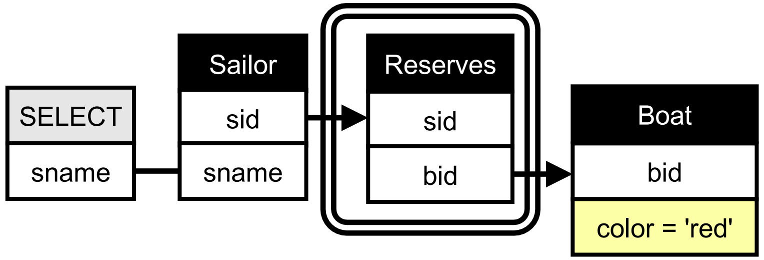
```
SELECT  DISTINCT S.sname
FROM    Sailor S
WHERE   NOT EXISTS(
        SELECT  *
        FROM    Reserves R
        WHERE   R.sid = S.sid
        AND     NOT EXISTS(
                SELECT  *
                FROM    Boat B
                WHERE   B.color = 'red'
                AND     R.bid = B.bid))
```

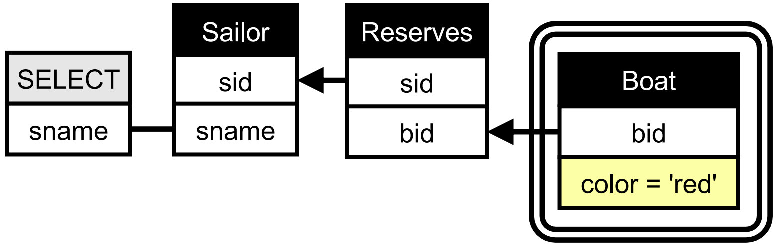
SQL — Sailors who reserved all red boats

```
SELECT  DISTINCT S.sname
FROM    Sailor S
WHERE   NOT EXISTS(
        SELECT  *
        FROM    Boat B
        WHERE   B.color = 'red'
        AND     NOT EXISTS(
                SELECT  *
                FROM    Reserves R
                WHERE   R.bid = B.bid
                AND     R.sid = S.sid))
```

QV







Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

Student (<u>sid</u>, sname)
Takes (<u>sid, cid, semester</u>)
Course (<u>cid</u>, cname, department)

Actor (<u>aid</u>, aname)
Plays (<u>aid, mid, role</u>)
Movie (<u>mid</u>, mname, director)

	not	only	all
Sailors renting boats	have not reserved a red boat	reserved only red boats	reserved all red boats
Students taking classes	took no art class	took only art classes	took all art classes
Actors playing in movies	did not play in a Hitchcock movie	played only Hitchcock movies	played in all Hitchcock movies

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

Student (<u>sid</u>, sname)
Takes (<u>sid, cid, semester</u>)
Course (<u>cid</u>, cname, department)

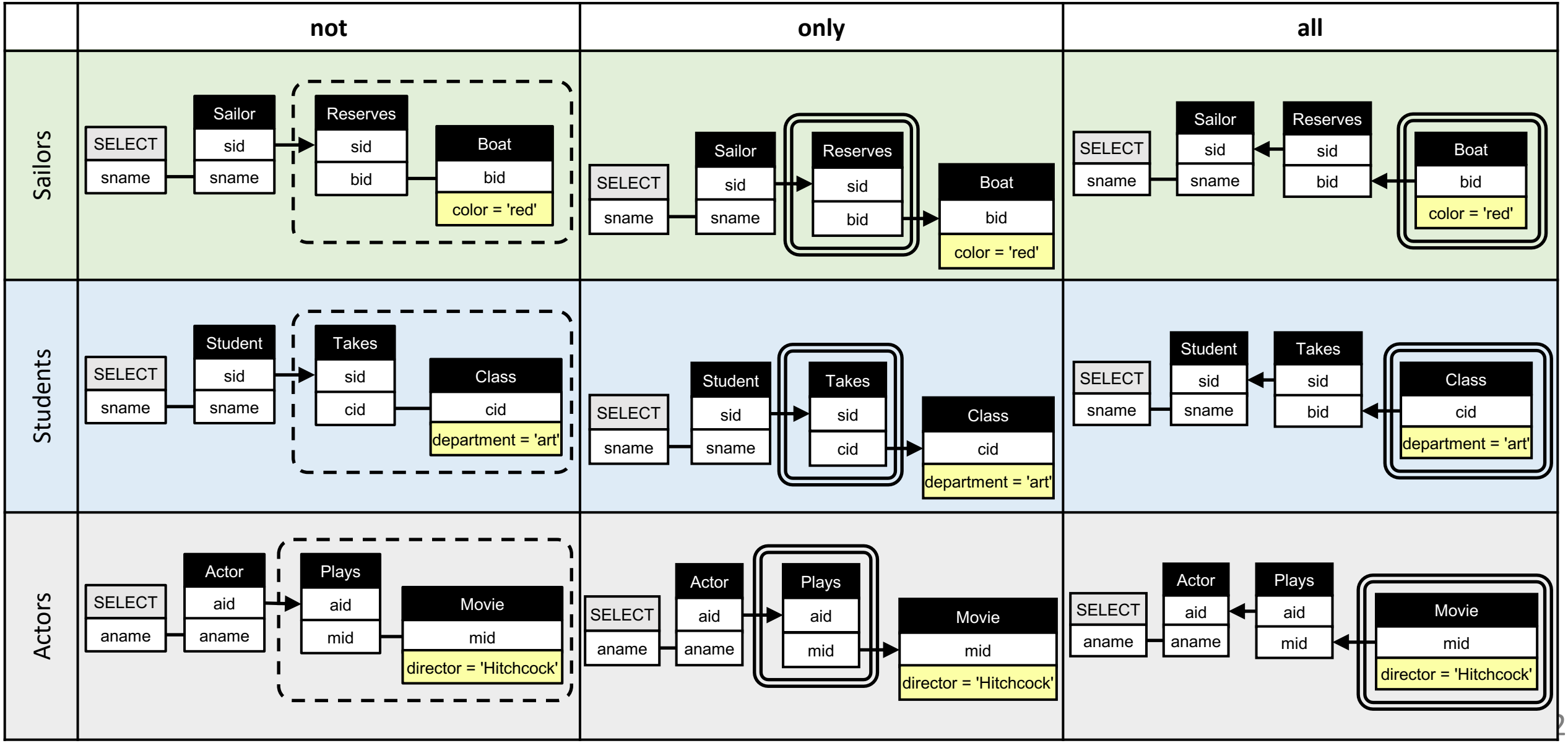
Actor (<u>aid</u>, aname)
Plays (<u>aid, mid, role</u>)
Movie (<u>mid</u>, mname, director)

	not	only	all
Sailors	SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS(SELECT * FROM Reserves R, Boat B WHERE R.sid = S.sid AND R.bid = B.bid AND B.color = 'red')	SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS(SELECT * FROM Reserves R WHERE R.sid = S.sid AND NOT EXISTS(SELECT * FROM Boat B WHERE B.color = 'red' AND B.bid = R.bid))	SELECT DISTINCT S.sname FROM Sailor S WHERE NOT EXISTS(SELECT * FROM Boat B WHERE B.color = 'red' AND NOT EXISTS(SELECT * FROM Reserves R WHERE R.bid = B.bid AND R.sid = S.sid))
Students	SELECT DISTINCT S.sname FROM Student S WHERE NOT EXISTS(SELECT * FROM Takes T, Class C WHERE T.sid = S.sid AND C.cid = T.cid AND C.department ='art')	SELECT DISTINCT S.sname FROM Student S WHERE NOT EXISTS(SELECT * FROM Takes T WHERE T.sid = S.sid AND NOT EXISTS(SELECT * FROM Class C WHERE C.department = 'art' AND C.cid= T.cid))	SELECT DISTINCT S.sname FROM Student S WHERE NOT EXISTS(SELECT * FROM Class C WHERE C.department= 'art' AND NOT EXISTS(SELECT * FROM Takes T WHERE T.cid= C.cid AND T.sid= S.sid))
Actors	SELECT DISTINCT A.aname FROM Actor A WHERE NOT EXISTS(SELECT * FROM Plays P, Movie M WHERE P.aid = A.aid AND M.mid = P.mid AND M.director = 'Hitchcock')	SELECT DISTINCT A.aname FROM Actor A WHERE NOT EXISTS(SELECT * FROM Plays P WHERE P.aid = A.aid AND NOT EXISTS(SELECT * FROM Movie M WHERE M.director = 'Hitchcock' AND M.mid = P.mid))	SELECT DISTINCT A.aname FROM Actor A WHERE NOT EXISTS(SELECT * FROM Movie M WHERE M.director = 'Hitchcock' AND NOT EXISTS(SELECT * FROM Plays P WHERE P.mid = M.mid AND P.aid = A.aid))

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

Student (sid, sname)
Takes (sid, cid, semester)
Course (cid, cname, department)

Actor (aid, aname)
Plays (aid, mid, role)
Movie (mid, mname, director)



Logical SQL Patterns

Logical patterns are the building blocks of most SQL queries.

Patterns are very hard to extract from the SQL text.

A pattern can appear across different database schemas.

Think of queries like:

- Find sailors who reserved all red boats
- Find students who took all art classes
- Find actors who played in all movies by Hitchcock

What does this query return ?

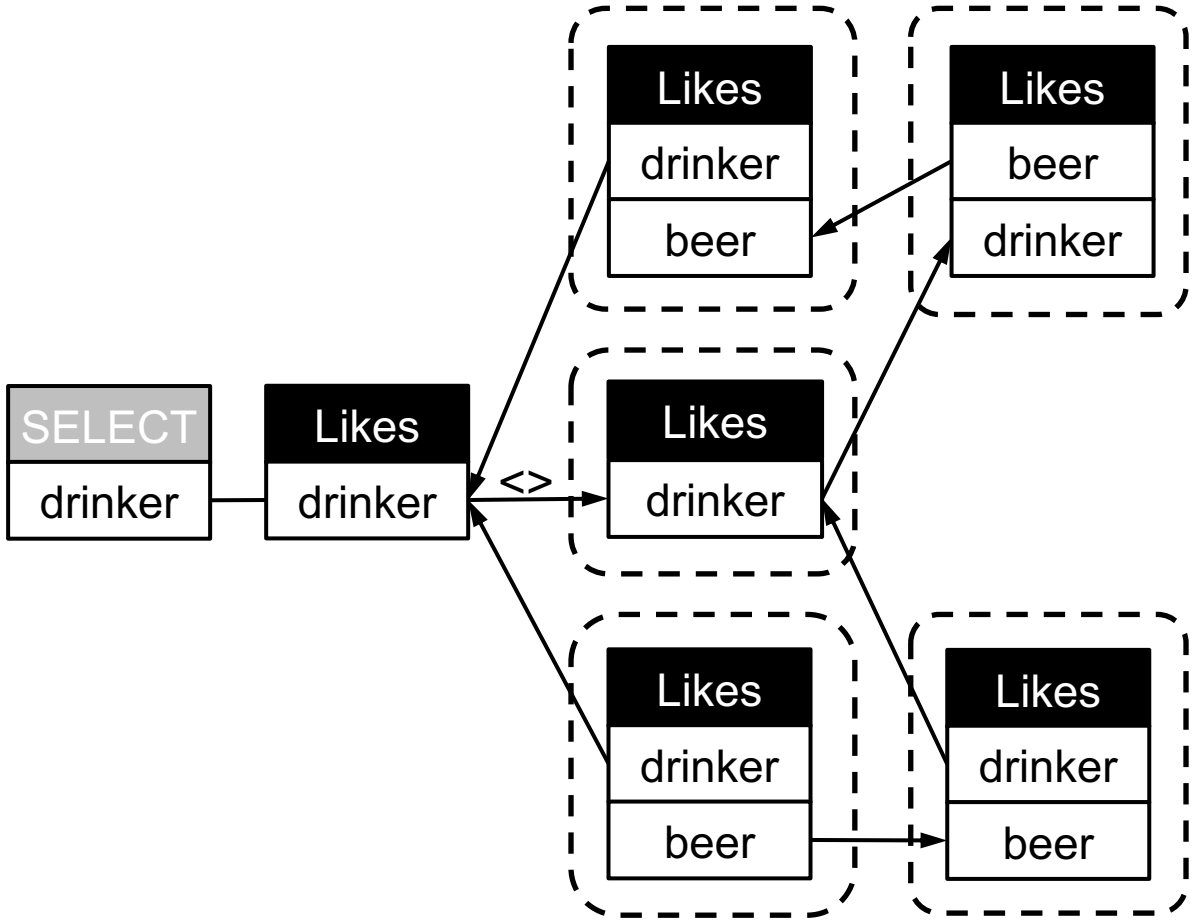
Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```

What does this query return ?

Likes(drinker,beer)

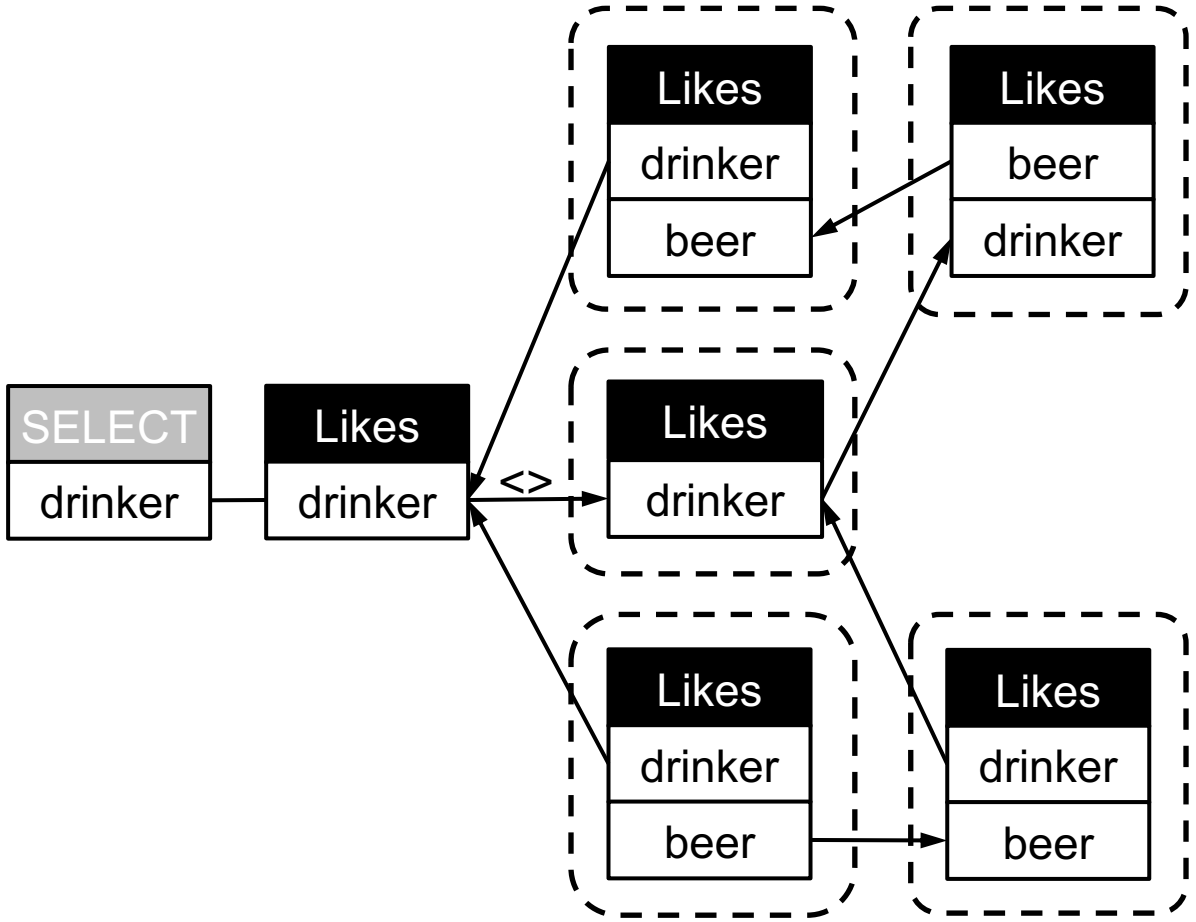
```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```



Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

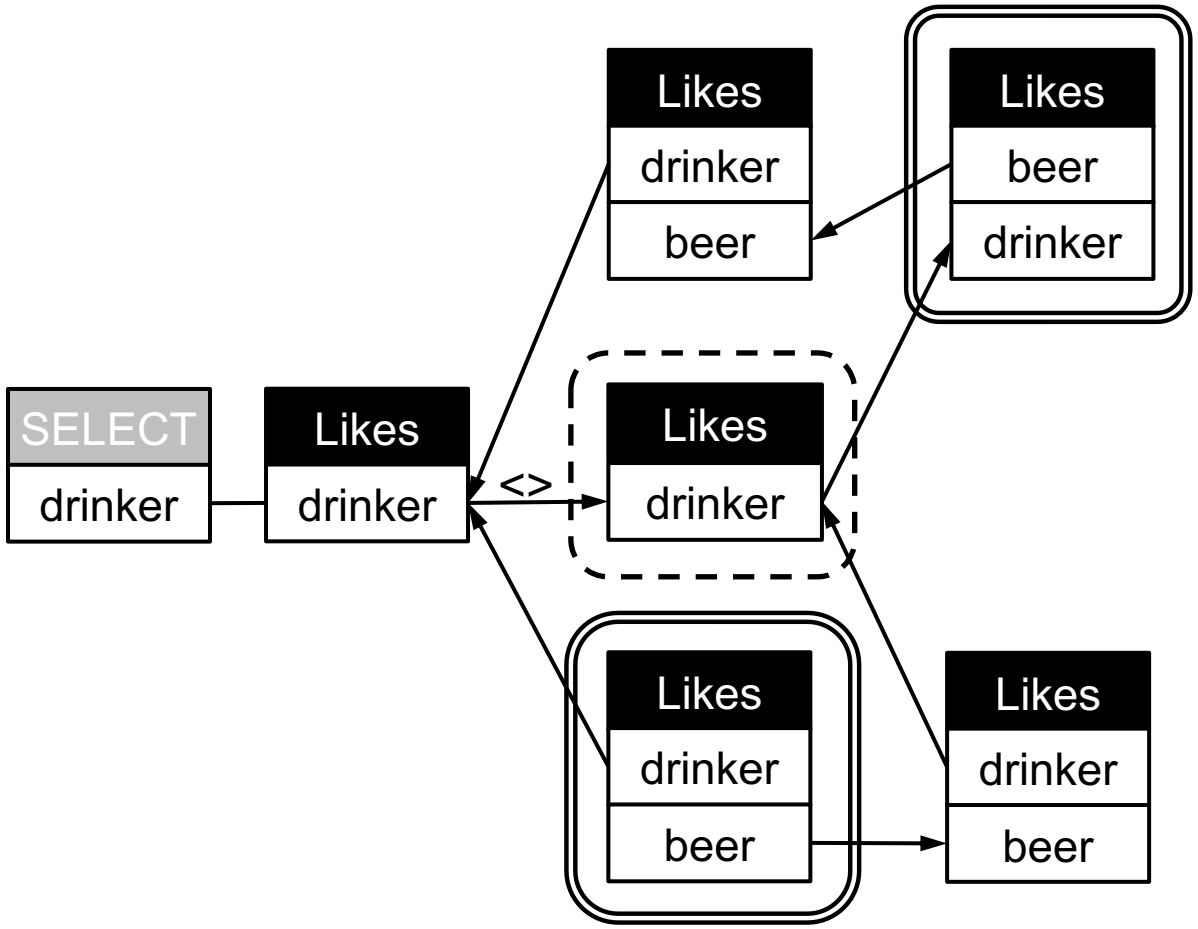
```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```



Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

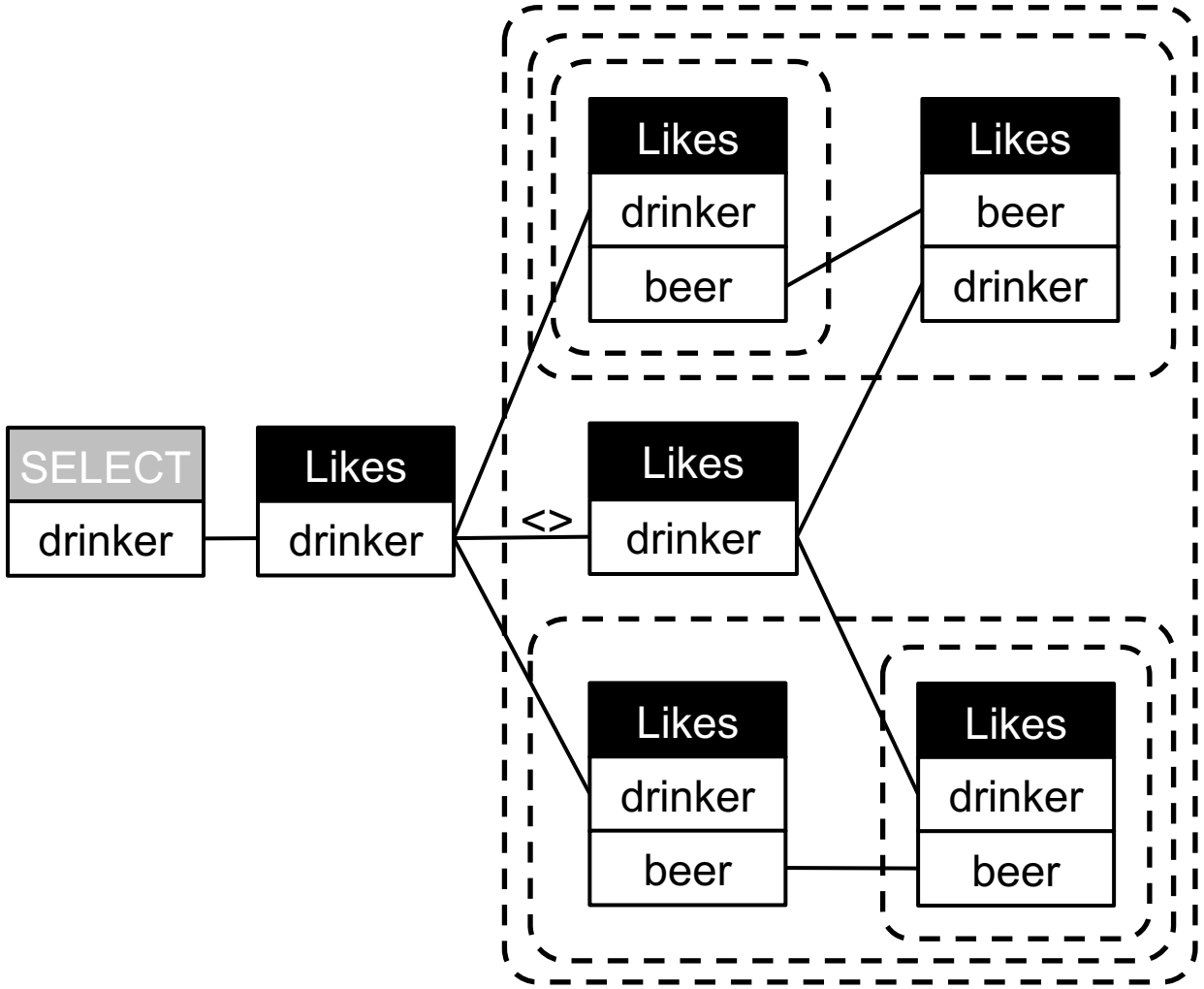
```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```



Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

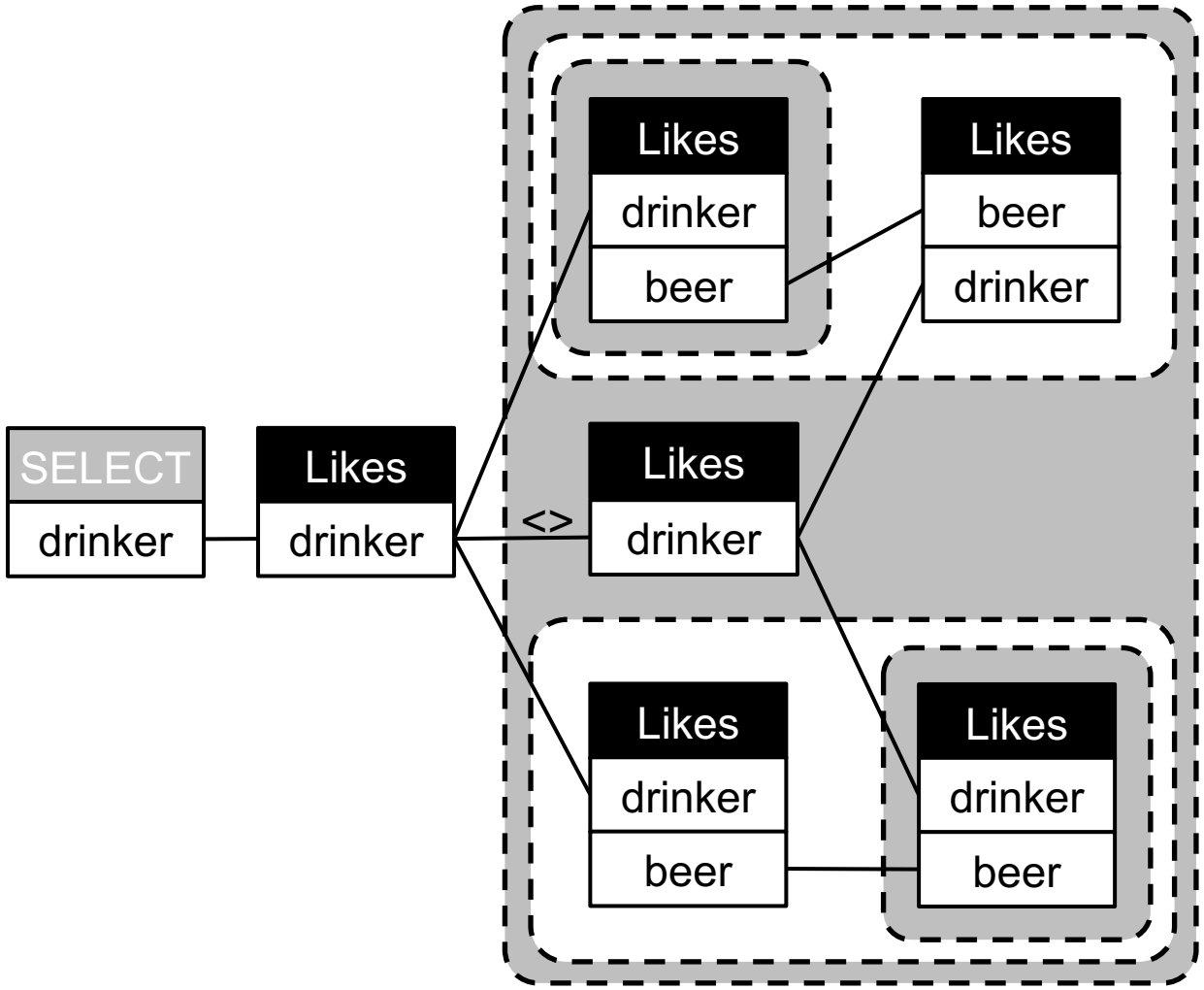
```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```



Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

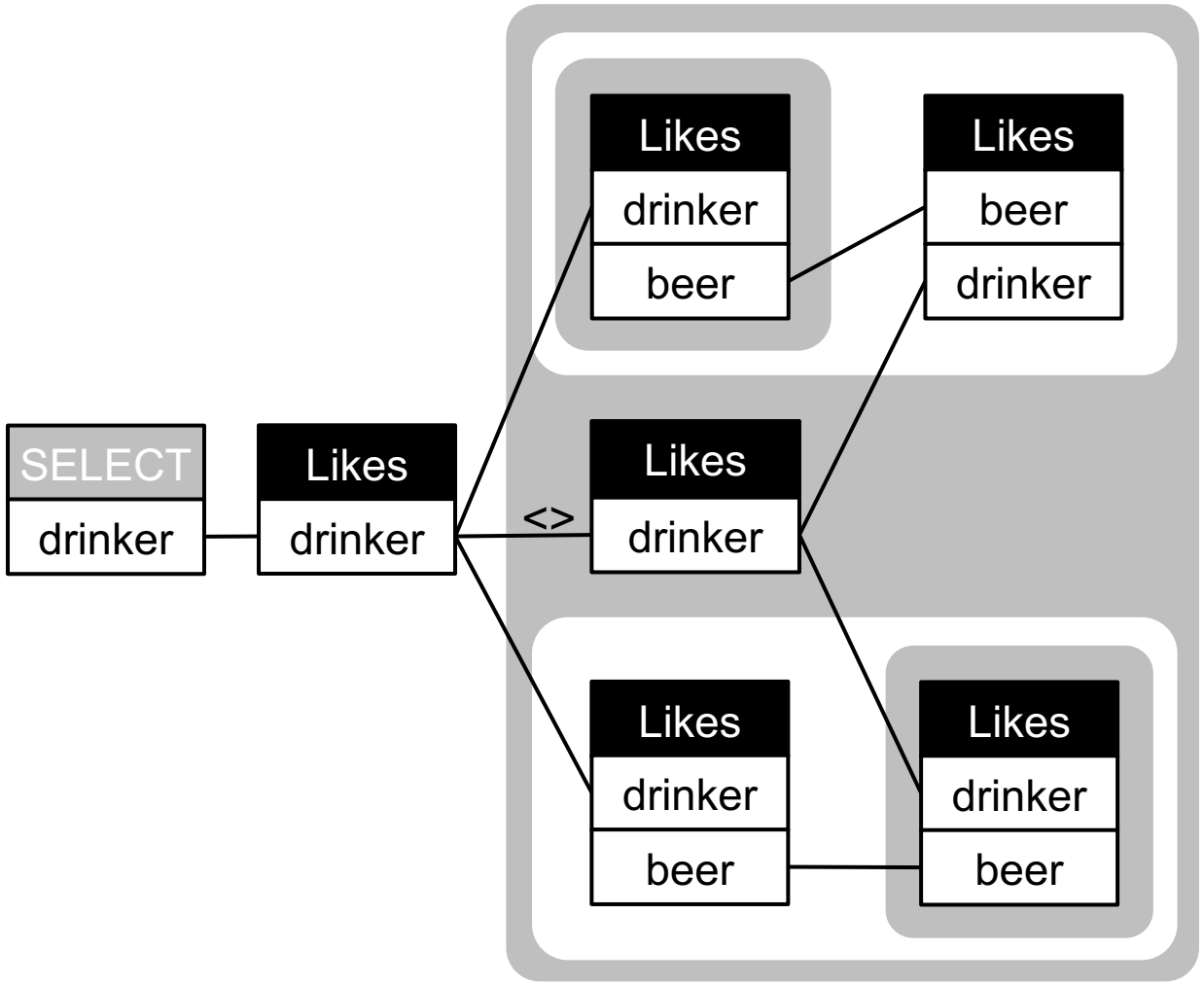
```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```



Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```

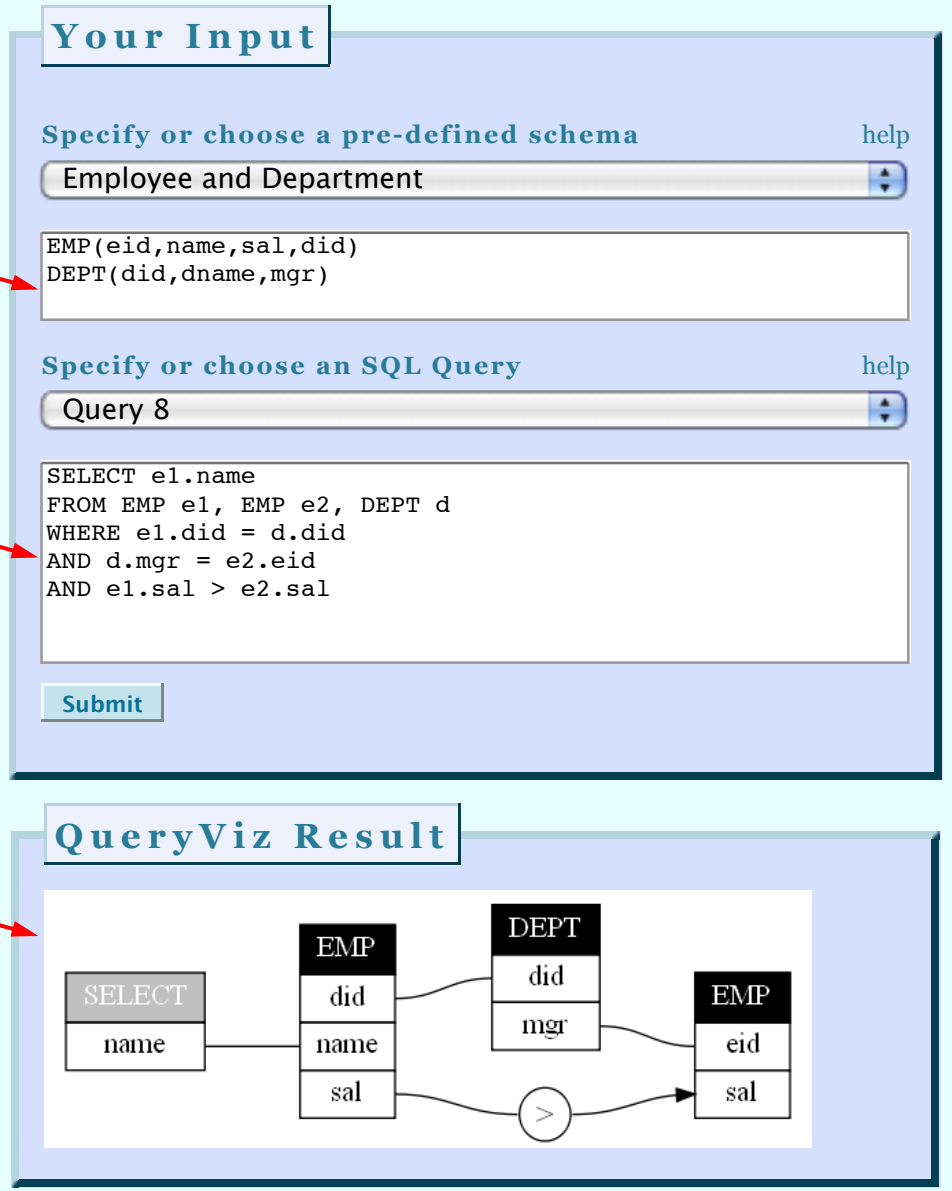


https://demo.queryvis.com

Input: Schema

Input Query

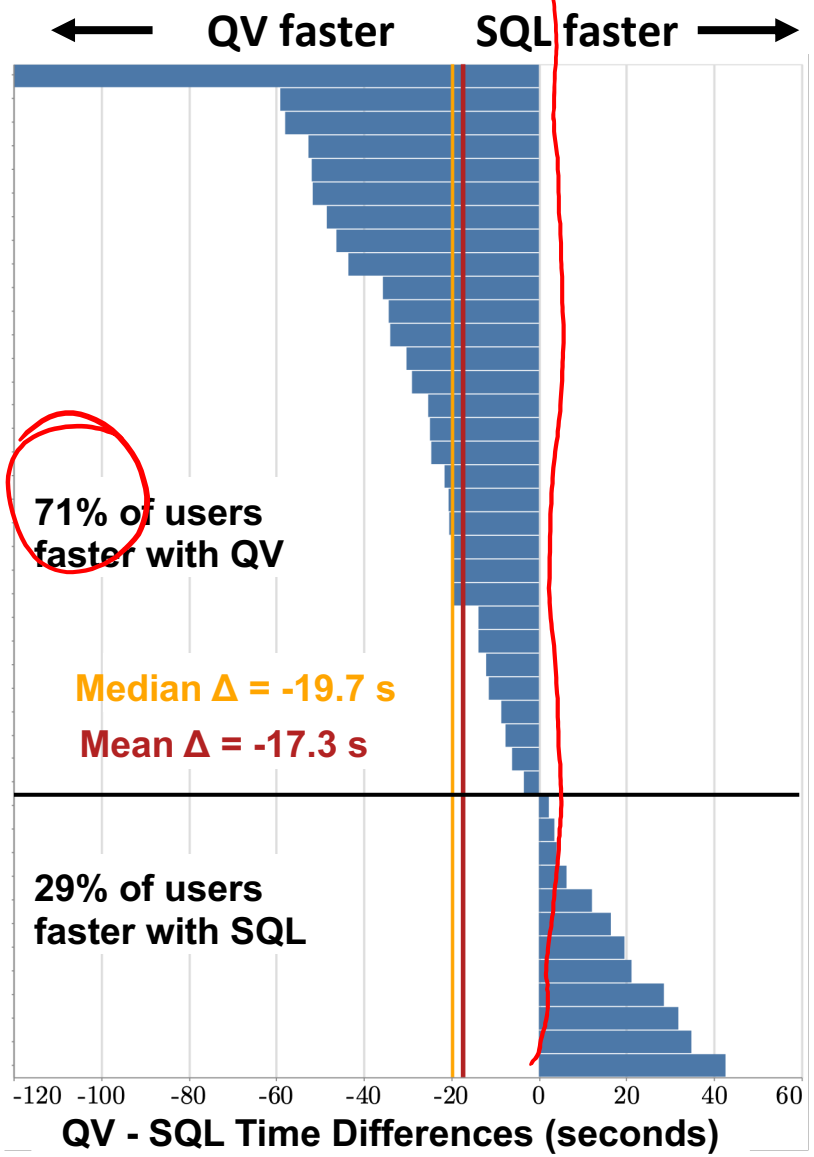
Output: Visualization

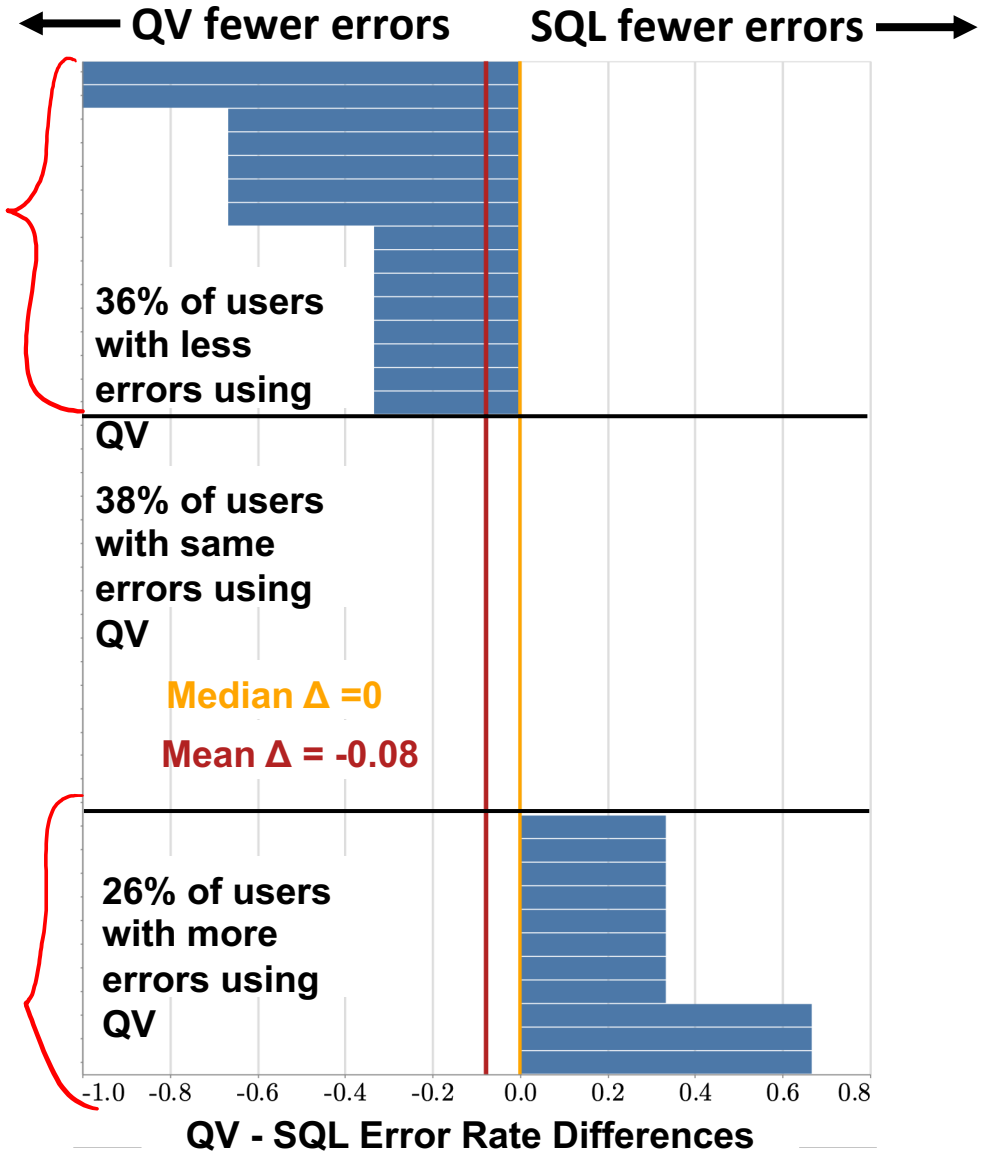


Amazon Turk user study with SQL users

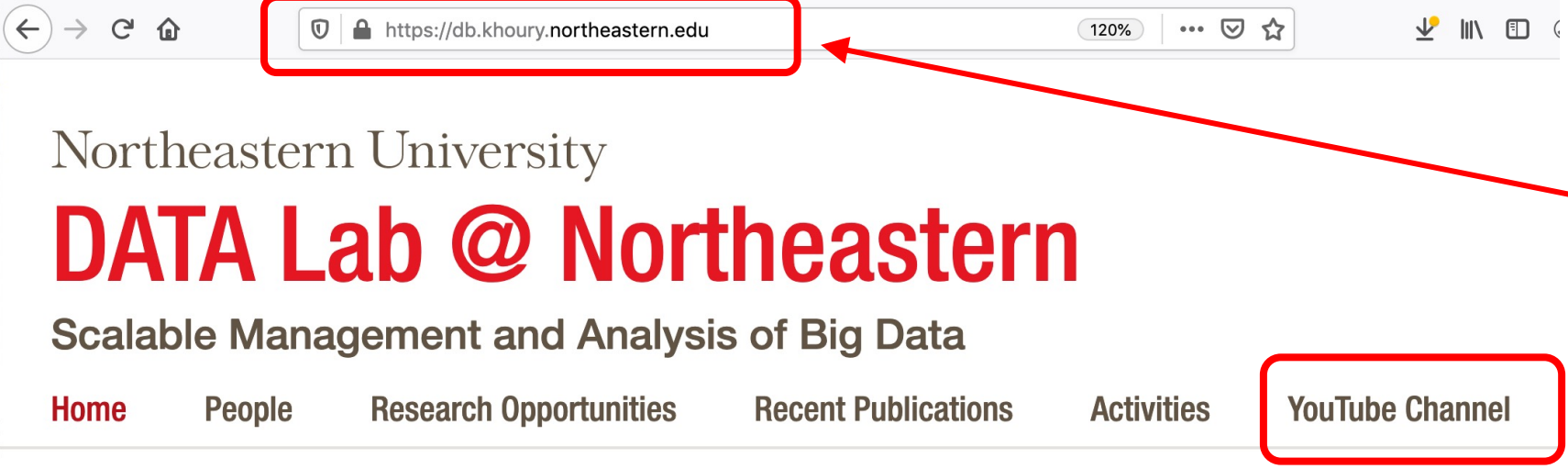
Leventidis+ [SIGMOD'20]

Each bar below corresponds to one participant (42 bars/participants in total)

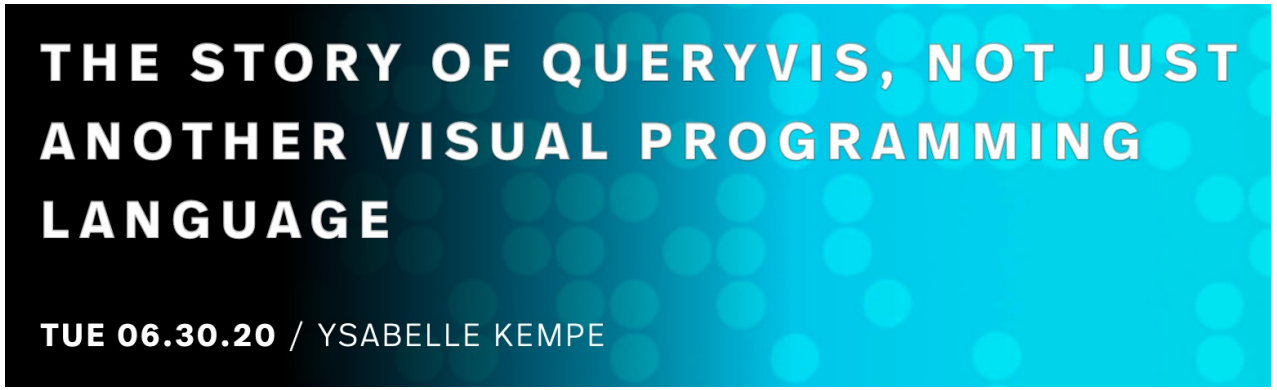




Source: Leventidis, Zhang, Dunne, Gatterbauer, Jagadish, Riedewald: *QueryVis: Logic-based Diagrams help Users Understand Complicated SQL Queries Faster*. SIGMOD 2020. https://doi.org/10.1145/3318464.3389767



https://queryvis.com



https://www.khoury.northeastern.edu/the-story-of-queryvis-not-just-another-visual-programming-language/