Updated 9/28/2022

Topic 1: SQL
L07: SQL advanced

Wolfgang Gatterbauer

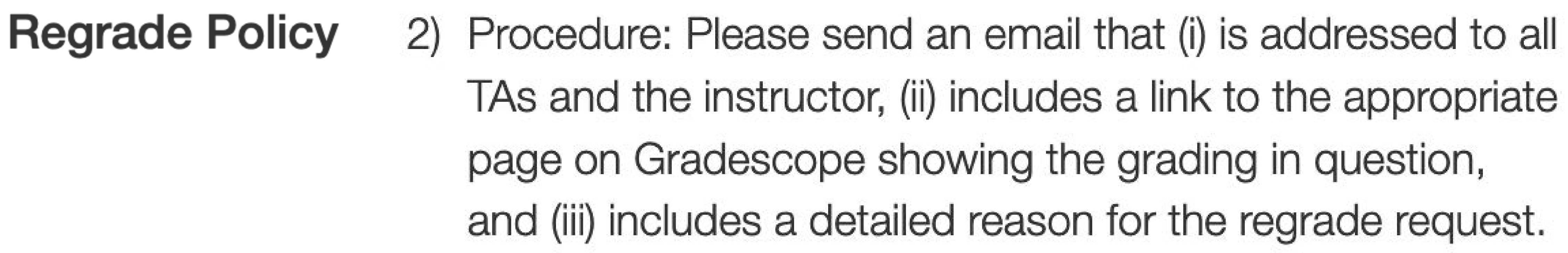
CS3200 Database design (fa22)

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

9/28/2022

Class warm-up

- Last class summary
- Grading philosophy: full points if correct over any database (unless something explicitly specified); if question ambiguous, we will fix



Regrade Policy

2) Procedure: Please send an email that (i) is addressed to all TAs and the instructor, (ii) includes a link to the appropriate page on Gradescope showing the grading in question, and (iii) includes a detailed reason for the regrade request.

- SQL today: Nulls, outer joins

Sorting Strings

Side topic: sorting of strings

true or false?

ASCII encoding

ASCII #	char
48	0
49	1
...	...
57	9
65	A
...	...
90	Z
97	a
...	...
122	z

```
SELECT 'A' < 'a' as eval
```
⇨ ?

```
SELECT '1' < 'A' as eval
```
⇨ ?

```
SELECT 'a' < 'ab' as eval
```
⇨ ?

```
SELECT 'a' < 'B' as eval
```
⇨ ?

Side topic: sorting of strings

true or false?

ASCII encoding

ASCII #	char
48	0
49	1
...	...
57	9
65	A
...	...
90	Z
97	a
...	...
122	z

```
SELECT 'A' < 'a' as eval
```

$\Rightarrow$

eval
true

```
SELECT '1' < 'A' as eval
```

$\Rightarrow$?

```
SELECT 'a' < 'ab' as eval
```

$\Rightarrow$?

```
SELECT 'a' < 'B' as eval
```

$\Rightarrow$?

Side topic: sorting of strings

true or false?

ASCII encoding

ASCII #	char
48	0
49	1
...	...
57	9
65	A
...	...
90	Z
97	a
...	...
122	z

```
SELECT 'A' < 'a' as eval
```
⇨

eval
true

```
SELECT '1' < 'A' as eval
```
⇨

eval
true

```
SELECT 'a' < 'ab' as eval
```
⇨ ?

```
SELECT 'a' < 'B' as eval
```
⇨ ?

Side topic: sorting of strings

true or false?

ASCII encoding

ASCII #	char
48	0
49	1
...	...
57	9
65	A
...	...
90	Z
97	a
...	...
122	z

```
SELECT 'A' < 'a' as eval
```

eval
true

```
SELECT '1' < 'A' as eval
```

eval
true

```
SELECT 'a' < 'ab' as eval
```

(lexicographical order)

eval
true

```
SELECT 'a' < 'B' as eval
```

?

Side topic: sorting of strings

true or false?

ASCII encoding

ASCII #	char
48	0
49	1
…	…
57	9
65	A
…	…
90	Z
97	a
…	…
122	z

```
SELECT 'A' < 'a' as eval
```

eval
true

```
SELECT '1' < 'A' as eval
```

eval
true

```
SELECT 'a' < 'ab' as eval
```

(lexicographical order)

eval
true

```
SELECT 'a' < 'B' as eval
```

eval
false

Null Values

3-valued logic example

- Three logicians walk into a bar. The bartender asks: "Do all of you want a drink?"
- The 1st logician says: "I don't know."
- The 2nd logician says: "I don't know."
- The 3rd logician says: "Yes!"

What is going on here ?

Nulls in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things, e.g.:

?

Nulls in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things, e.g.:
  - Value exists but is unknown
  - Value not applicable

A new student without GPA

sid	Name	GPA
101	Alice	3.2
123	Bob	null

- The schema specifies for each attribute if it can be NULL (nullable attribute) or not ("NOT NULL")
- Lots of ongoing research on NULLs
- Next: How does SQL cope with tables that have NULLs ?

Null Values

- In SQL there are three Boolean values ("ternary logic")
  - FALSE, TRUE, UNKNOWN

- If x= NULL then
  - Boolean conditions are also NULL. E.g: x='Joe'
  - Arithmetic operations produce NULL. E.g: 4*(3-x)/7
  - But aggregates ignore NULL values (exception: count(*))

we will practice in a moment!

- Logical reasoning:
  - FALSE = 0
  - TRUE = 1
  - UNKNOWN = 0.5

x AND y = min(x,y)

x OR y = max(x,y)

NOT x = (1 – x)

Null Values: example

```
SELECT  *
FROM    Person
WHERE  (age < 25)
   and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
NULL	6.5	170

?

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

Rule in SQL: include only tuples that yield TRUE

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

Rule in SQL: include only tuples that yield TRUE

```
SELECT  *
FROM    Person
WHERE   age < 25  or age >= 25
```

?

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

Rule in SQL:
include only tuples that
yield TRUE

```
SELECT  *
FROM    Person
WHERE   age < 25  or age >= 25
```

$\leftarrow$ Unexpected behavior

$\Downarrow$

```
SELECT  *
FROM    Person
WHERE   age < 25  or age >= 25 or age IS NULL
```

Test NULL
explicitly

Null Values and Aggregates

T

gid	val
1	NULL
1	NULL
2	a
2	B
2	z
2	z
2	NULL
3	A
3	A
3	Z

```
SELECT gid,
         MAX(val) maxv,
         MIN(val) minv,
         COUNT(*) ctr,
         COUNT(val) ctv,
         COUNT(DISTINCT val) ctdv
FROM     T
GROUP BY gid
ORDER BY gid
```

Key rule: NULL is ignored by aggregate functions if you reference the column specifically.
Exception: COUNT(*)

?

Null Values and Aggregates

T

gid	val
1	NULL
1	NULL
2	a
2	B
2	z
2	z
2	NULL
3	A
3	A
3	Z

```
SELECT gid,
       MAX(val) maxv,
       MIN(val) minv,
       COUNT(*) ctr,
       COUNT(val) ctv,
       COUNT(DISTINCT val) ctdv
FROM   T
GROUP BY gid
ORDER BY gid
```

Key rule: NULL is ignored by aggregate functions if you reference the column specifically: count(col) starts with 0, sum(col) starts wtih null. Exception: COUNT(*)

gid	maxv	minv	ctr	ctv	ctdv
1	NULL	NULL	2	0	0
2	z	B	5	4	3
3	Z	A	3	3	2

Null Values and Aggregates

T

gid	val
1	NULL
1	NULL
2	a
2	B
2	z
2	z
2	NULL
3	A
3	A
3	Z

```
SELECT val,
       COUNT(*) ctr
FROM   T
GROUP BY val
```

?

NULL is included by "GROUP BY".
Relative sorting of NULL by "ORDER BY" is DBMS-specific

Null Values and Aggregates

T

gid	val
1	NULL
1	NULL
2	a
2	B
2	z
2	z
2	NULL
3	A
3	A
3	Z

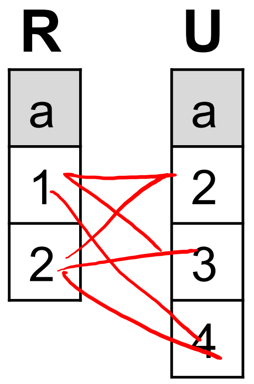
```
SELECT val,
       COUNT(*) ctr
FROM   T
GROUP BY val
```

val	ctr
A	2
B	1
Z	1
a	1
z	2
NULL	3

NULL is included by "GROUP BY".
Relative sorting of NULL by "ORDER BY" is DBMS-specific

Theta joins (θ)

Theta joins



R

a
1
2

U

a
2
3
4

What do these queries compute?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a < U.a
```

?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a >= U.a
```

?

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

Theta joins

R

a
1
2

U

a
2
3
4

What do these queries compute?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a < U.a
```

⇒

a	b
1	2
1	3
1	4
2	3
2	4

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a >= U.a
```

⇒ ?

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

Theta joins

R

a
1
2

U

a
2
3
4

What do these queries compute?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a < U.a
```

a	b
1	2
1	3
1	4
2	3
2	4

$\cup$

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a >= U.a
```

a	b
2	2

$\times$

Think about these two queries as a partition of the Cartesian product

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

Processing Multiple Tables–Joins

- **Join**: a relational operation that causes two or more tables with a common domain to be combined into a single table or view
- **Equi-join**: a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table
- **Natural join**: an equi-join in which one of the duplicate columns is eliminated in the result table
- A **Theta-join** allows for arbitrary comparison relationships (e.g., ≥). An equijoin is a theta join using the equality operator.

The common columns in joined tables are usually the primary key of the dominant table and the foreign key of the dependent table in 1:M relationships

Inner Joins vs. Outer Joins

Join Illustration

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

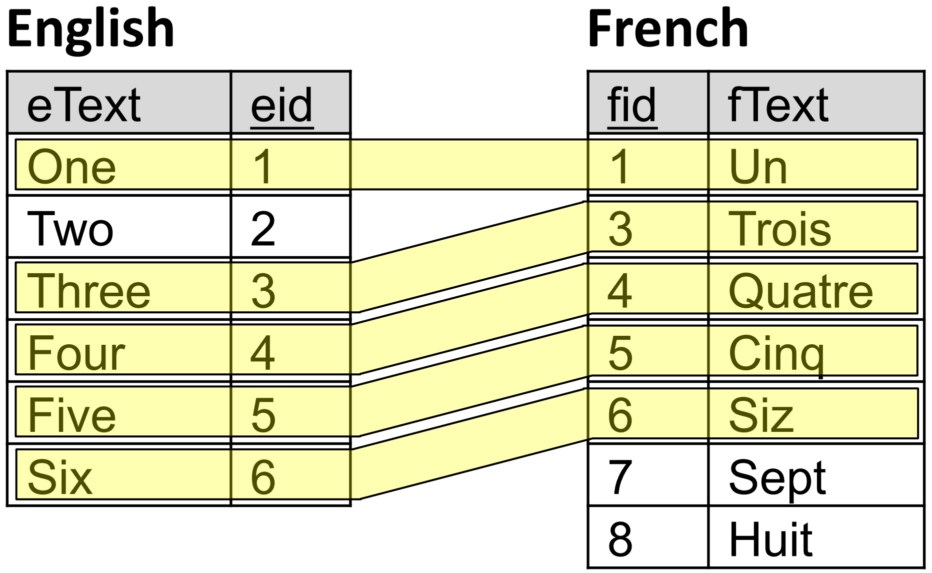
An "inner join":

```
SELECT  *
FROM    English, French
WHERE   eid = fid
```

?

Join Illustration

361



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

An "inner join":

```
SELECT  *
FROM    English, French
WHERE   eid = fid
```

Same as:

```
SELECT  *
FROM    English JOIN French
ON      eid = fid
```

etext	eid	fid	ftext
One	1	1	Un
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz

"JOIN"
same as
"INNER JOIN"

Join Illustration

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Null also sometimes just shown as empty

How do we get a join with the full data ?

~~SELECT *~~
~~FROM English JOIN French~~
~~ON eid = fid~~

etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz
NULL	NULL	7	Sept
NULL	NULL	8	Huit

Join Illustration

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

"FULL JOIN"
same as
"FULL OUTER JOIN"

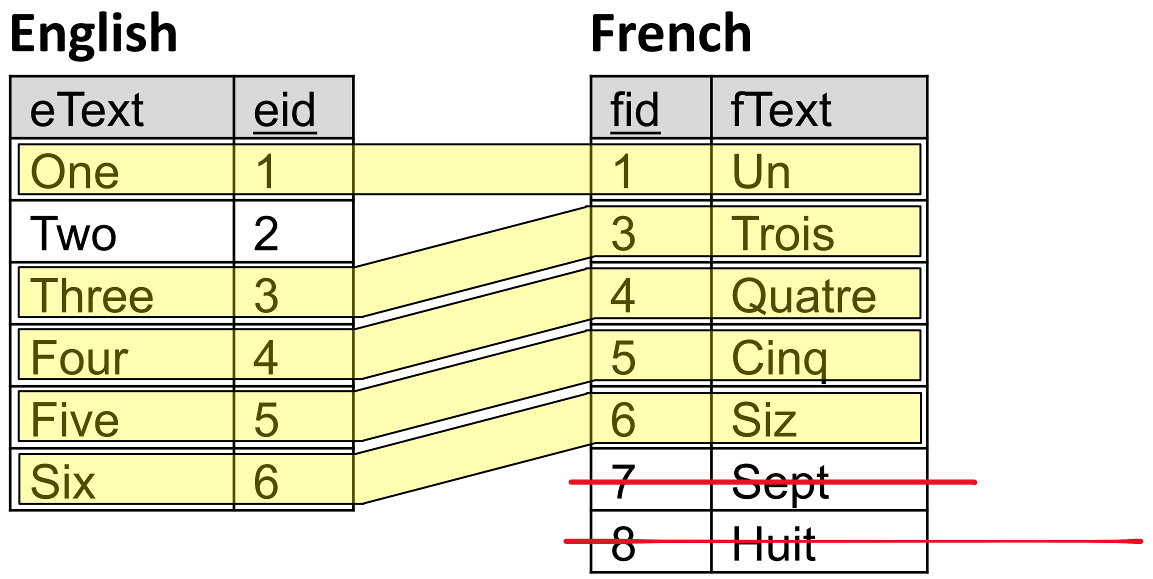
```
SELECT  *
FROM    English FULL JOIN French
ON      English.eid = French.fid
```

```
~~SELECT  *~~
~~FROM    English JOIN French~~
~~ON      eid = fid~~
```

Null also sometimes just shown as empty

etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz
NULL	NULL	7	Sept
NULL	NULL	8	Huit

Join Illustration



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
~~7~~	~~Sept~~
~~8~~	~~Huit~~

```
SELECT  *
FROM    English LEFT JOIN French
ON      English.eid = French.fid
```

etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz

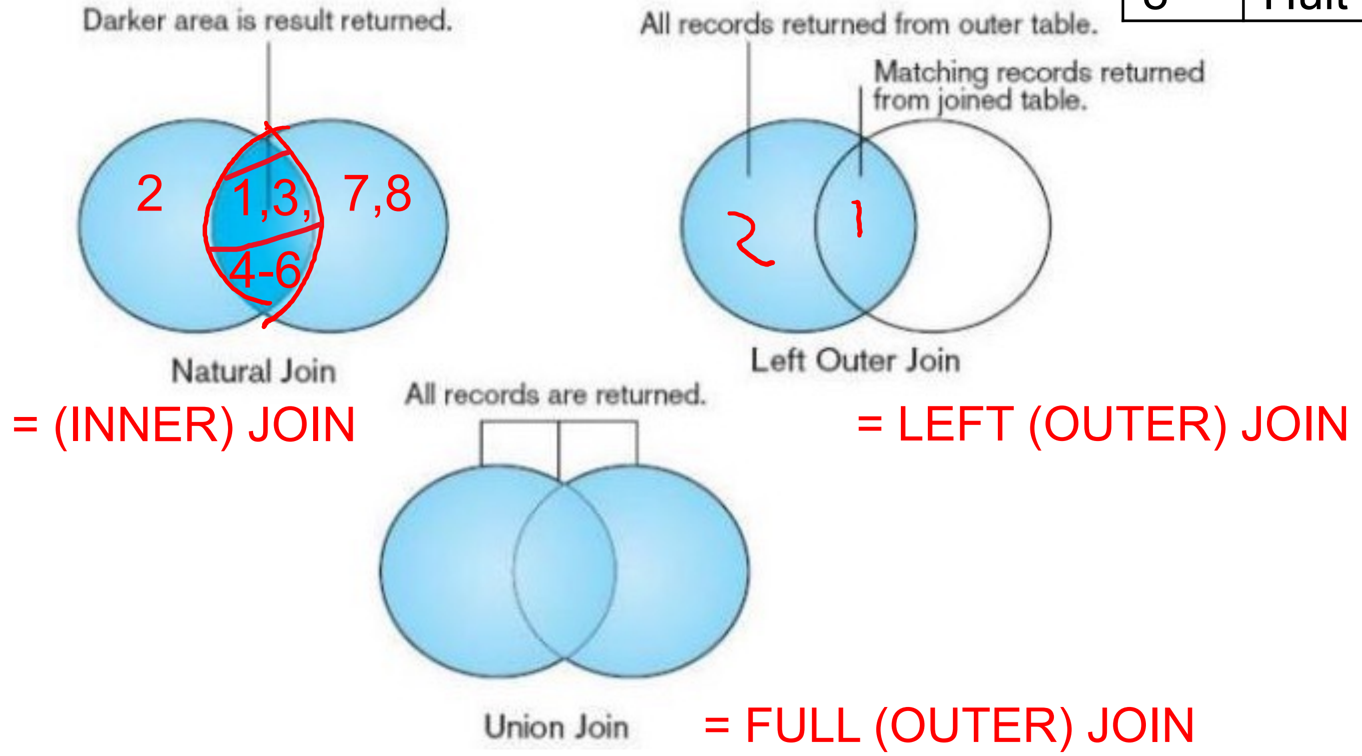
Join Illustration

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

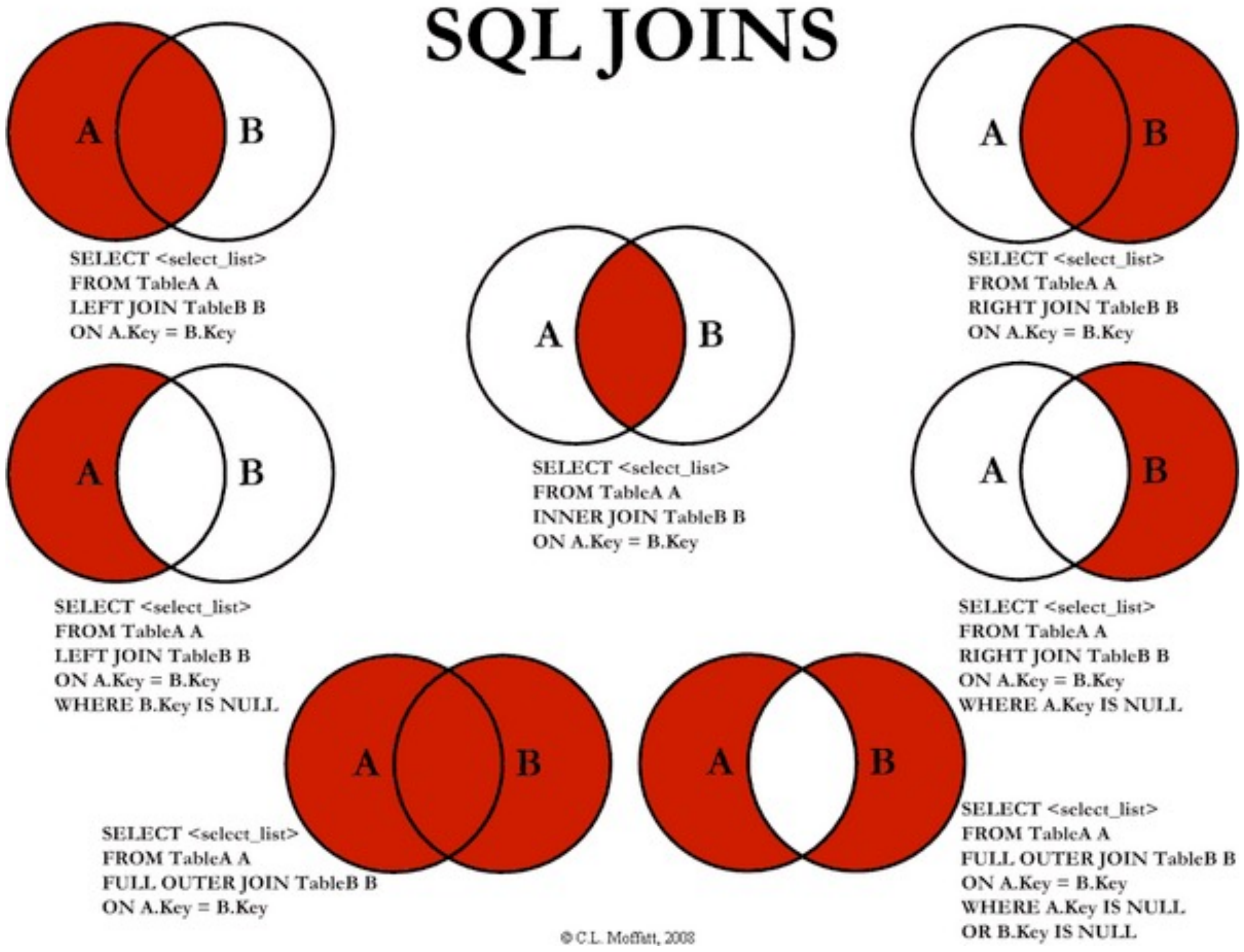
fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit


Darker area is result returned.
2
1,3,
4-6
7,8
Natural Join
= (INNER) JOIN
All records returned from outer table.
Matching records returned from joined table.
2
1
Left Outer Join
= LEFT (OUTER) JOIN
All records are returned.
Union Join
= FULL (OUTER) JOIN

Source: Fig. 7-2, Hoffer et al., Modern Database Management, 10ed ed, 2011.

Detailed Illustration with Examples (follow the link)

also called "anti-join"



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

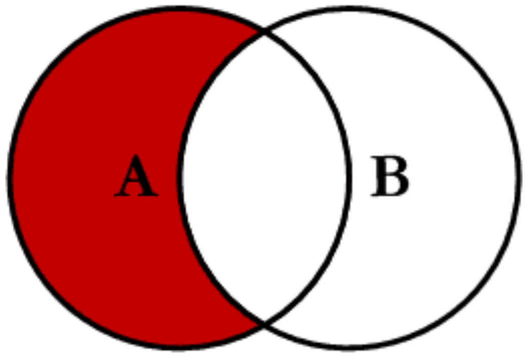
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

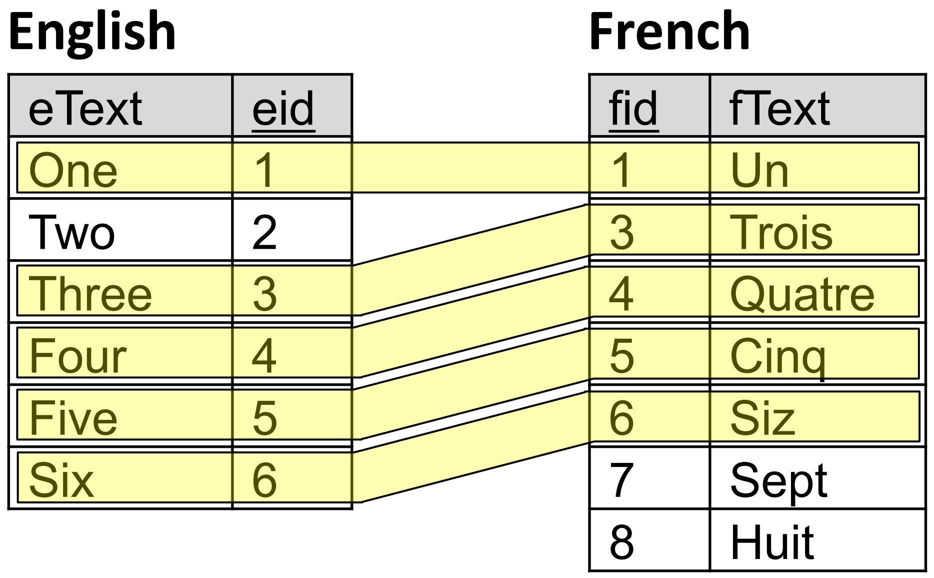
Check this web page for illustrating examples

Source: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

Let's practice anti-joins



```
SELECT <select_list>
FROM A
LEFT JOIN B
ON A.key = B.key
WHERE B.key IS NULL
```



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

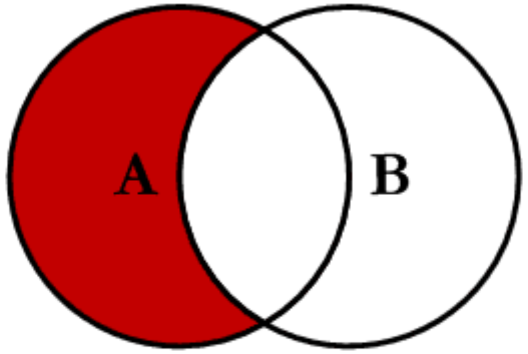
French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

?

Let's practice anti-joins



```
SELECT <select_list>
FROM A
LEFT JOIN B
ON A.key = B.key
WHERE B.key IS NULL
```

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

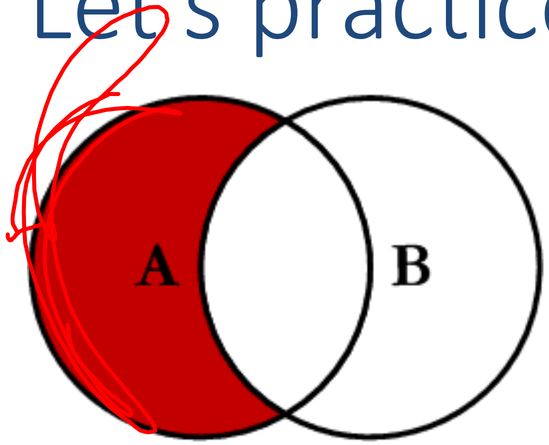
Results

eText	eid
Two	2

How to write in SQL?

?

Let's practice anti-joins



```
SELECT <select_list>
FROM A
LEFT JOIN B
ON A.key = B.key
WHERE B.key IS NULL
```

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
Two	2

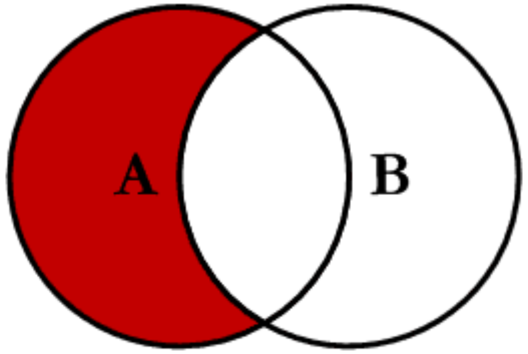
How to write in SQL?

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NULL
```

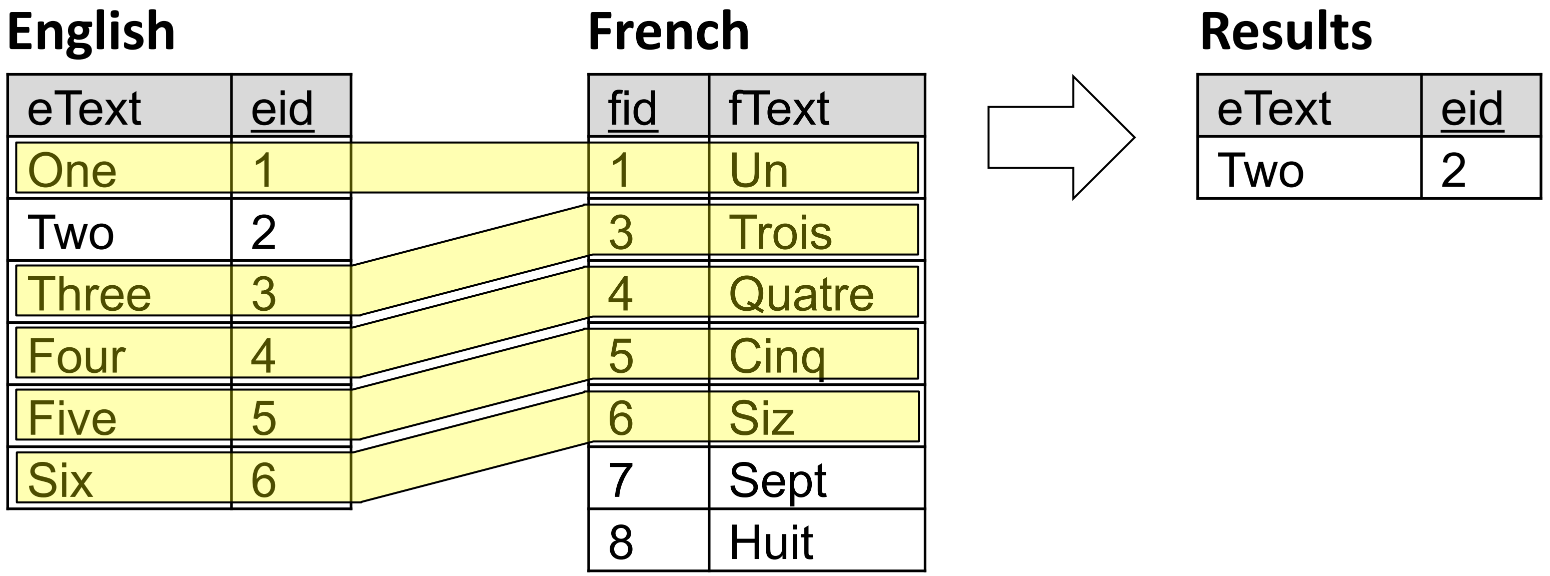
Any alternative?

?

Let's practice anti-joins



```
SELECT <select_list>
FROM A
LEFT JOIN B
ON A.key = B.key
WHERE B.key IS NULL
```



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
Two	2

How to write in SQL?

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NULL
```

Any alternative?

```
SELECT *
FROM English
WHERE eid NOT IN
    (SELECT fid
    FROM French)
```

"Semi-joins:" kind of the anti-anti-joins...

361

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
One	1
Three	3
Four	4
Five	5
Six	6

What do we have to change to these queries to get the tuples in English that have a partner in French?

?

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NULL
```

```
SELECT *
FROM English
WHERE eid NOT IN
    (SELECT fid
    FROM French)
```

"Semi-joins:" kind of the anti-anti-joins...

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
One	1
Three	3
Four	4
Five	5
Six	6

What do we have to change to these queries to get the tuples in English that have a partner in French?

What if fid is not a key?

?

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NOT NULL
```

```
SELECT *
FROM English
WHERE eid IN
    (SELECT fid
     FROM French)
```

"Semi-joins:" kind of the anti-anti-joins...

361

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
One	1
Three	3
Four	4
Five	5
Six	6

What do we have to change to these queries to get the tuples in English that have a partner in French?

What if fid is not a key?

```
DISTINCT
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NOT NULL
```

```
SELECT *
FROM English
WHERE eid IN
    (SELECT fid
    FROM French)
```

Outer Joins with aggregates

Missing sales

Item(<u>name</u>, category)
Purchase(iName, store, month)

An "inner join":

```
SELECT  Item.name, Purchase.store
FROM    Item, Purchase
WHERE   Item.name = Purchase.iName
```

Same as:

```
SELECT  Item.name, Purchase.store
FROM    Item JOIN Purchase
ON      Item.name = Purchase.iName
```

We will have a group exercise in a few slides. Please ask questions if things are not clear, or make screenshots to discuss later also in your group

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

⇨

Missing sales

Item(name, category)
Purchase(iName, store, month)

We will have a group exercise in a few slides. Please ask questions if things are not clear, or make screenshots to discuss later also in your group

An "inner join":

```
SELECT  Item.name, Purchase.store
FROM    Item, Purchase
WHERE   Item.name = Purchase.iName
```

Same as:

```
SELECT  Item.name, Purchase.store
FROM    Item JOIN Purchase
ON      Item.name = Purchase.iName
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

$\Rightarrow$

Result

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Products that never sold will be lost ☹

Missing sales

Item(name, category)
Purchase(iName, store, month)

An "inner join":

```
SELECT  Item.name, Purchase.store
FROM    Item, Purchase
WHERE   Item.name = Purchase.iName
```

Same as:

```
SELECT  Item.name, Purchase.store
FROM    Item INNER JOIN Purchase
ON      Item.name = Purchase.iName
```

"INNER JOIN" same as "JOIN"

What if you want to include never-sold products?

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Products that never sold will be lost ☹

Outer Joins

Item(name, category)
Purchase(iName, store, month)

"LEFT OUTER JOIN"
same as
"LEFT JOIN"

If we want to include the never-sold products, then we need an "outer join":

```
SELECT  Item.name, Purchase.store
FROM    Item LEFT JOIN Purchase
ON      Item.name = Purchase.iName
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

Now we include those products ☺

Outer Joins

Item(<u>name</u>, category)
Purchase(iName, store, month)

Same question, but now only for sales in month = 9:

```
SELECT  Item.name, Purchase.store
FROM    Item LEFT JOIN Purchase
ON      Item.name = Purchase.iName
WHERE   month = 9
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store	
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9
OneClick	NULL	NULL

M ?

Outer Joins w/ selection

Item(name, category)
Purchase(iName, store, month)

Same question, but now only for sales in month = 9:

```
SELECT  Item.name, Purchase.store
FROM    Item LEFT JOIN Purchase
ON      Item.name = Purchase.iName
WHERE   month = 9
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Camera	Wiz

?

What just happened????

The products disappeared *despite* outer join ☹

Outer Joins w/ selection

Item(name, category)
Purchase(iName, store, month)

Explanation: the filter ("month = 9") applies to the result of the outer join. Any tuple that has NULL as month, does not pass the filter

Same question, but now only for sales in month = 9:

```
SELECT  Item.name, Purchase.store
FROM    Item LEFT JOIN Purchase
ON      Item.name = Purchase.iName
WHERE   month = 9
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
~~Gizmo~~	~~Wiz~~	~~8~~
~~Camera~~	~~Ritz~~	~~8~~
Camera	Wiz	9

Result

Name	Store
Camera	Wiz

?

What just happened????

The products disappeared *despite* outer join ☹

Outer Joins w/ selection

Item(name, category)
Purchase(iName, store, month)

Same question, but now only for those sold in month = 9:

```
SELECT  Item.name, Purchase.store
FROM    Item LEFT JOIN Purchase
ON      Item.name = Purchase.iName
WHERE   month = 9
```

?

What do we need to do to get back all names?

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
~~Gizmo~~	~~Wiz~~	~~8~~
~~Camera~~	~~Ritz~~	~~8~~
Camera	Wiz	9

Result

Name	Store
Camera	Wiz
Gizmo	NULL
OneClick	NULL

Outer Joins w/ selection

Item(name, category)
Purchase(iName, store, month)

Explanation: now the filter ("month = 9") applies to the right side of the left join *before* joining. NULLs are appended only after filter, during join

Same question, but now only for those sold in month = 9:

```
SELECT  Item.name, Purchase.store
FROM    Item LEFT JOIN Purchase
ON      (Item.name = Purchase.iName
AND     month = 9)
```

parenthesis not required, and just for illustration

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Camera	Wiz
Gizmo	NULL
OneClick	NULL

Now they are back again ☺

Outer Joins w/ selection

Item(<u>name</u>, category)
Purchase(iName, store, month)

334

Explanation: now the filter ("month = 9") applies to the right side of the left join *before* joining. NULLs are appended only after filter, during join

Same question, but now only for those sold in month = 9:

```
SELECT  Item.name, X.store
FROM    Item LEFT JOIN
        (SELECT iName, store FROM Purchase WHERE month = 9) X
ON      Item.name = X.iName
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
~~Gizmo~~	~~Wiz~~	~~8~~
~~Camera~~	~~Ritz~~	~~8~~
Camera	Wiz	9

Result

Name	Store
Camera	Wiz
Gizmo	NULL
OneClick	NULL

Now they are back again ☺

Empty Group Problem

Item(name, category)
Purchase(iName, store, month)

Q: Compute, for each product, the total number of sales in Sept (= month 9)

```
SELECT      name, count(*) ct
FROM        Item, Purchase
WHERE       name = iName
AND         month = 9
GROUP BY    name
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

$\Rightarrow$?

Empty Group Problem

Item(<u>name</u>, category)
Purchase(iName, store, month)

Q: Compute, for each product, the total number of sales in Sept (= month 9)

```
SELECT    name, count(*) ct
FROM      Item, Purchase
WHERE     name = iName
AND       month = 9
GROUP BY  name
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

$\Rightarrow$

Result

Name	ct
Camera	1

Whats wrong ?

Empty Group Problem

Item(name, category)
Purchase(iName, store, month)

Q: Compute, for each product, the total number of sales in Sept (= month 9)

```
SELECT     name, count(*) ct
FROM       Item, Purchase
WHERE      name = iName
AND        month = 9
GROUP BY   name
```

?

That's what we want: the count for *all* products. How do we get this anwer?

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

$\Rightarrow$

Result

Name	ct
Camera	1
Gizmo	0
OneClick	0

Empty Group Problem

Item(name, category)
Purchase(iName, store, month)

334

Q: Compute, for each product, the total number of sales in Sept (= month 9)

```
SELECT    name, count(store) ct
FROM      Item LEFT JOIN Purchase
ON        name = iName
AND       month = 9
GROUP BY  name
```

We need to use any attribute from "Purchase" to get the correct 0 count.
→ Try "iname" from "Purchase". Then try "name" from "Item".

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

⇒

Result

Name	ct
Camera	1
Gizmo	0
OneClick	0

Now we also get the products with 0 sales ☺

Empty Group Problem

Item(name, category)
Purchase(iName, store, month)

Q: Compute, for each product, the total number of sales in Sept (= month 9)

```
SELECT    name, count(store) ct, sum(month)),
FROM      Item LEFT JOIN Purchase
ON        name = iName
AND       month = 9
GROUP BY  name
```

SUM(COALESCE(month, 0))

What happens if you add "sum(month)" to the SELECT clause?

?

Tip: "COALESCE" function (comes later)

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

$\Rightarrow$

Result

Name	ct	s
Camera	1	
Gizmo	0	NULL
OneClick	0	NULL

Empty Group Problem

Item(name, category)
Purchase(iName, store, month)

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

```
SELECT  *
FROM  Item LEFT JOIN Purchase
ON  name = iName
AND  month = 9
```

Name	Category	iName	Store	Month
Gizmo	Gadget	null	null	null
Camera	Photo	Camera	Wiz	9
OneClick	Photo	null	null	null

```
SELECT  name,
   count(iname) c,
   sum(month) s,
   sum(coalesce(month,0)) sc
FROM  Item LEFT JOIN Purchase
ON  name = iName
AND  month = 9
GROUP BY  name
```

Result

Name	ct	s	sc
Camera	1	9	9
Gizmo	0	null	0
OneClick	0	null	0

Repeated use of WITH

Witnesses: with aggregates per group (8/10)

Find the product that is sold with max sales?

Purchase

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

⇨

Product	sales
Banana	70

```
SELECT      product, sum(quantity) as sales
FROM        Purchase
GROUP BY product
HAVING      sum(quantity) = (
            SELECT max (Q)
            FROM  ( SELECT   sum(quantity) Q
                    FROM     Purchase
                    GROUP BY product           ) X  )
```

WITH clause

```
WITH X AS
```

?

```
SELECT    product, sum(quantity) as sales
FROM      Purchase
GROUP BY product
HAVING    sum(quantity) = (
              SELECT max (Q)
              FROM  ( SELECT   sum(quantity) Q
                      FROM     Purchase
                      GROUP BY product            ) X   )
```

WITH clause

```
WITH X AS
        (SELECT   product, SUM(quantity) sales
         FROM     Purchase
         GROUP BY product)
```

```
SELECT     product, sum(quantity) as sales
FROM       Purchase
GROUP BY product
HAVING     sum(quantity) = (
           SELECT max (Q)
           FROM  ( SELECT  sum(quantity) Q
                   FROM    Purchase
                   GROUP BY product             ) X   )
```

WITH clause

```
WITH X AS
              (SELECT    product, SUM(quantity) sales
              FROM       Purchase
              GROUP BY product)
SELECT        *
FROM          X
WHERE
```

```
SELECT        product, sum(quantity) as sales
FROM          Purchase
GROUP BY product
HAVING        sum(quantity) = (
              SELECT max (Q)
              FROM  ( SELECT  sum(quantity) Q
                      FROM     Purchase
                      GROUP BY product          ) X    )
```

WITH clause

```
WITH X AS
            (SELECT    product, SUM(quantity) sales
            FROM       Purchase
            GROUP BY product)
SELECT      *
FROM        X
WHERE       sales =
            (SELECT    MAX (sales)
            FROM       X)
```

```
SELECT      product, sum(quantity) as sales
FROM        Purchase
GROUP BY product
HAVING      sum(quantity) = (
            SELECT max (Q)
            FROM  (  SELECT   sum(quantity) Q
                     FROM     Purchase
                     GROUP BY product         ) X   )
```

WITH clause

```
WITH X AS
            (SELECT   product, SUM(quantity) sales
             FROM     Purchase
             GROUP BY product),
Y AS
            (SELECT   MAX (sales) ms
             FROM     X)
SELECT      *
FROM        X
WHERE       sales = (SELECT ms FROM Y))
```

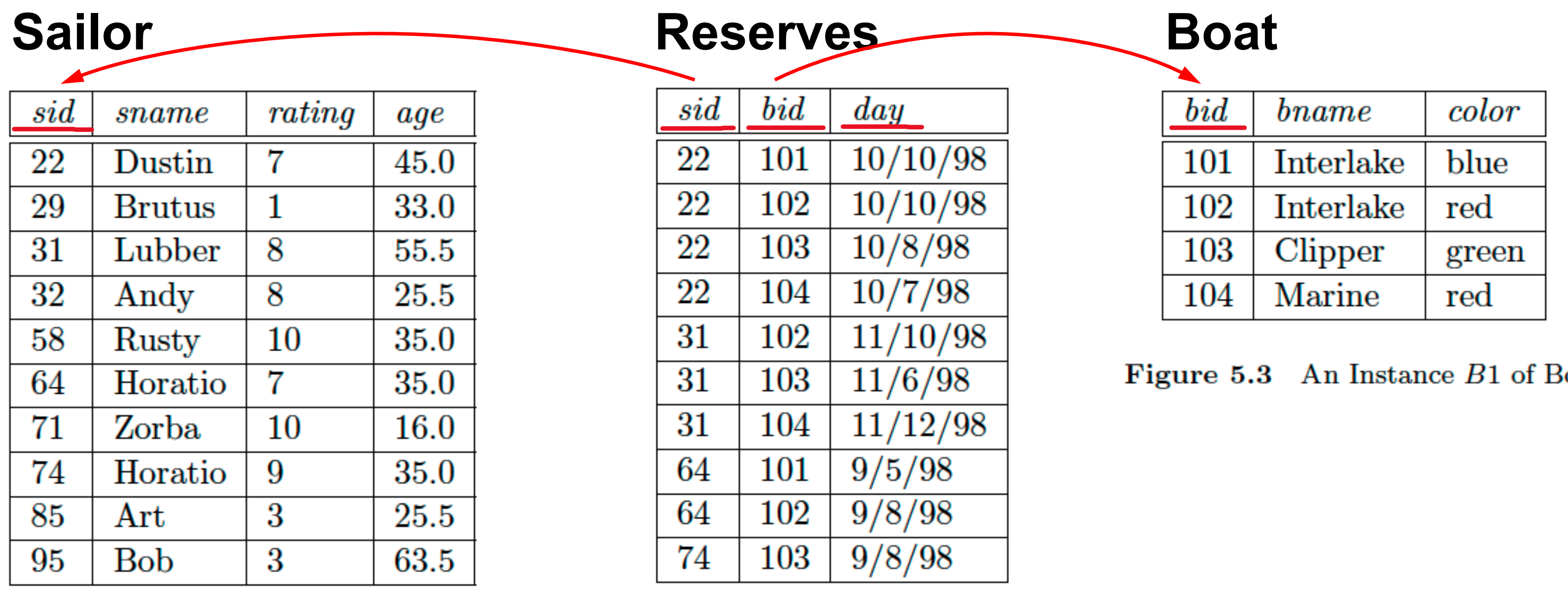
```
SELECT      product, sum(quantity) as sales
FROM        Purchase
GROUP BY product
HAVING      sum(quantity) = (
            SELECT max (Q)
            FROM  (  SELECT   sum(quantity) Q
                     FROM     Purchase
                     GROUP BY product           ) X    )
```

Understanding nested queries

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 *S*3 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 *R*2 of Reserves

Boat

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

Figure 5.3 An Instance *B*1 of Boats

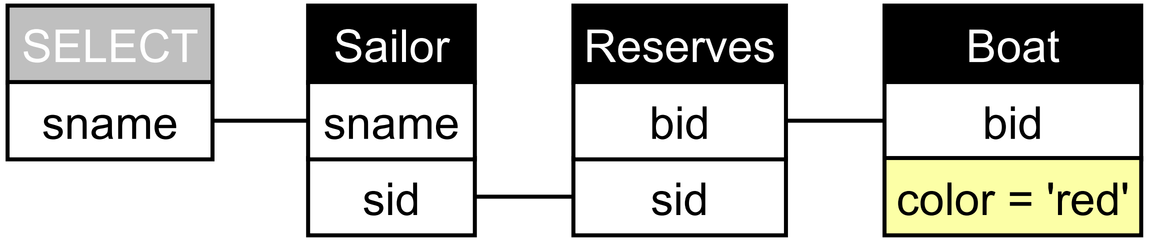
Nested query 1

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'))
```



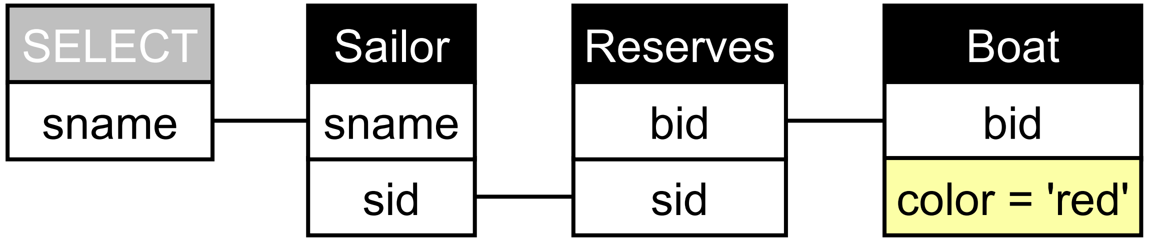
Nested query 1

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'))
```



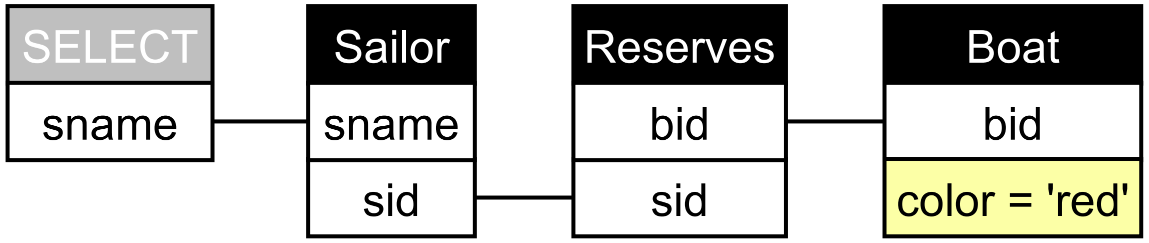
{S.sname | ∃S∈Sailor.(∃R∈Reserves.(R.sid=S.sid ∧ ∃B∈Boat.(B.bid=R.bid ∧ B.color='red')))}

Nested query 1

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

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

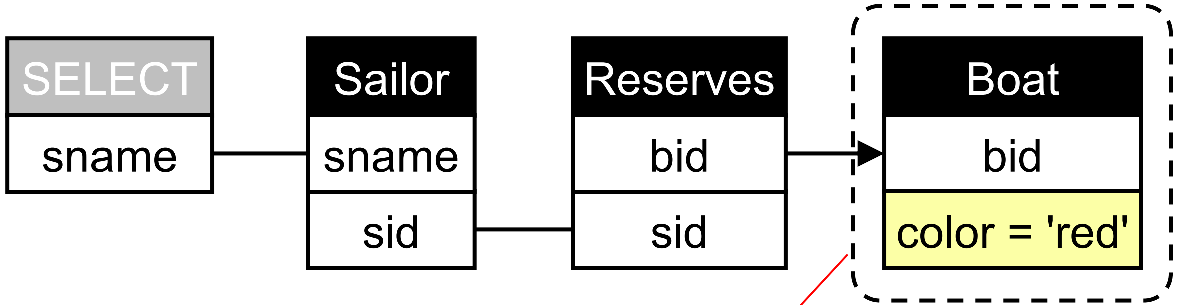
Nested query 2

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

340

Q: ?

```
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 ∄

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

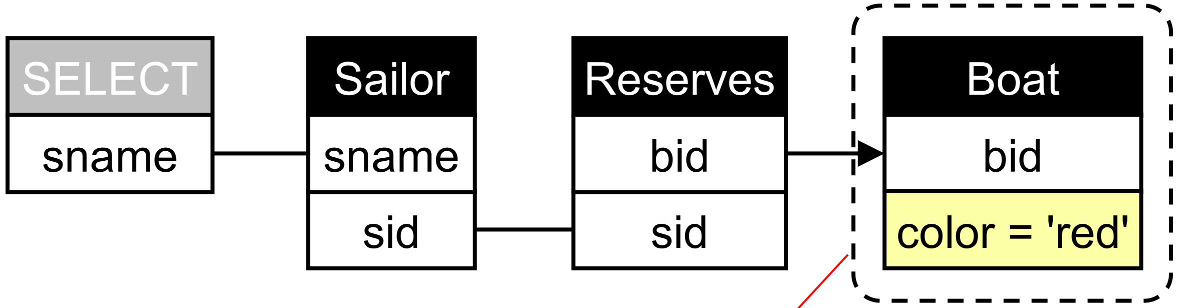
Nested query 2

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')))}

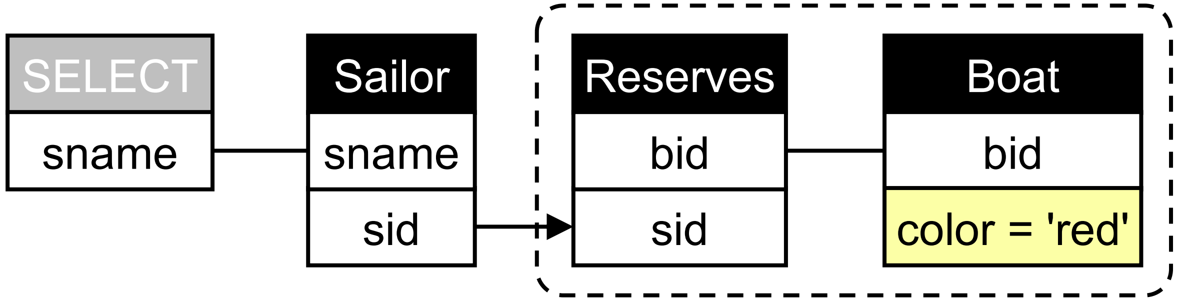
Nested query 3

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')))}

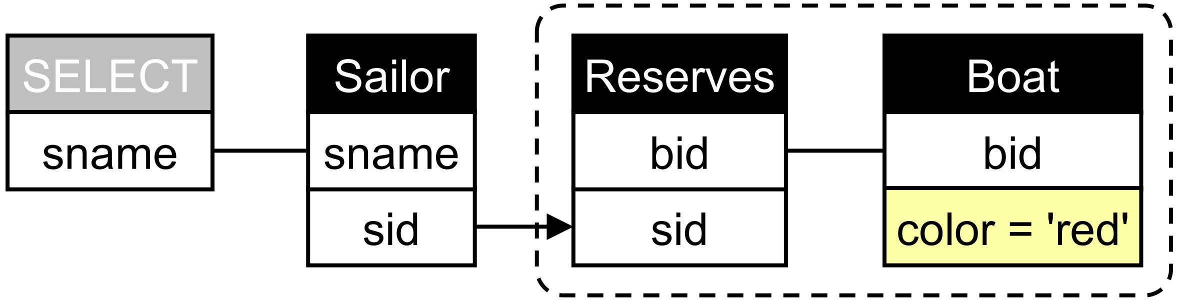
Nested query 3

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 | ∃S∈Sailor.(∄R∈Reserves.(R.sid=S.sid ∧ ∃B∈Boat.(B.bid=R.bid ∧ 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 *S*3 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 *R*2 of Reserves

Boat

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

Figure 5.3 An Instance *B*1 of Boats

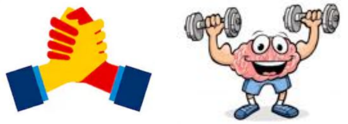
Should Dustin be in the output of either of the two queries?

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 either 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!

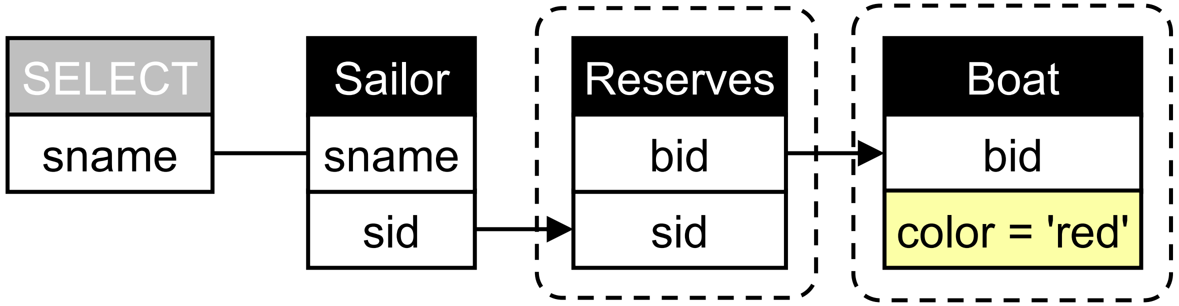
Nested query 4

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

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

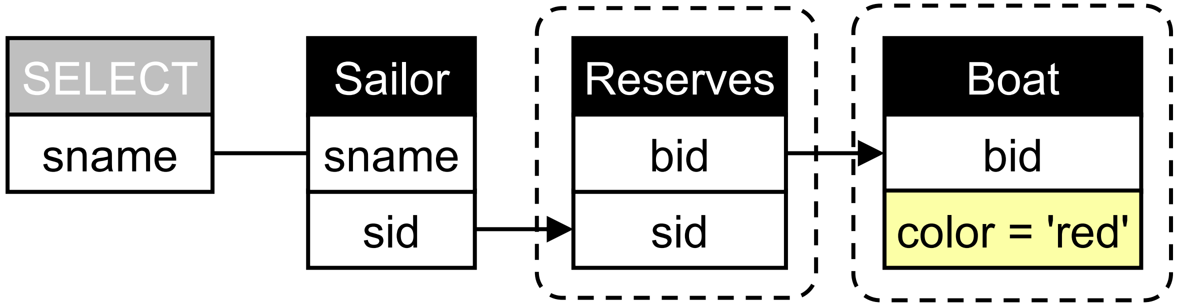
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

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'))
```



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

Nested query 4 (universal)

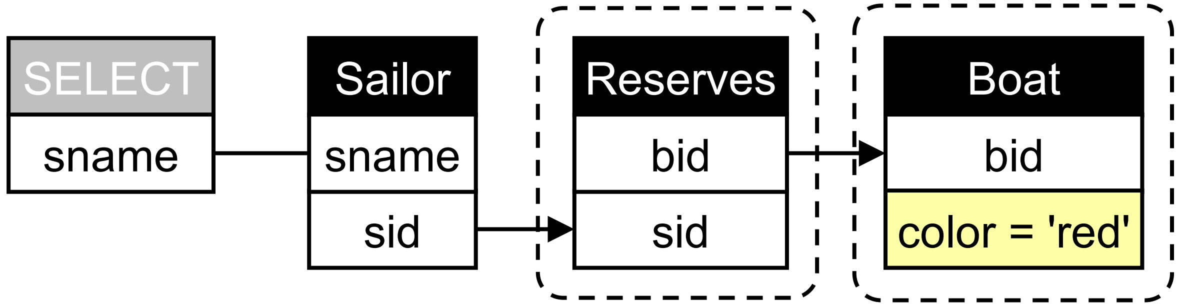
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

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

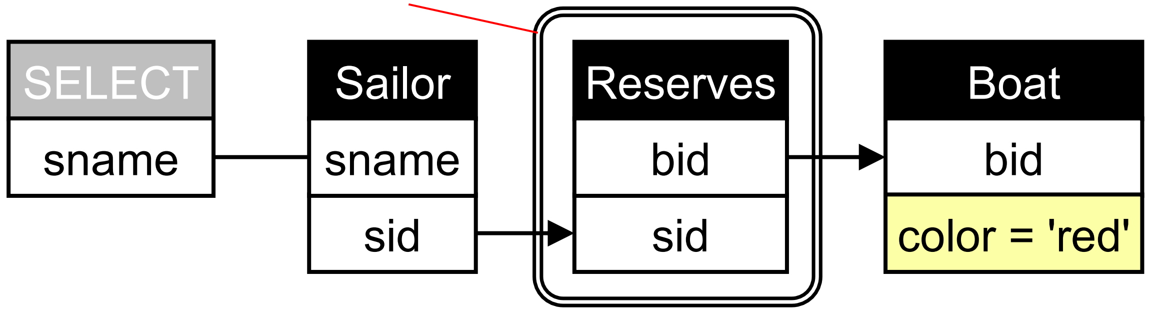
= Find the names of sailors who have reserved only red boats

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')))}

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

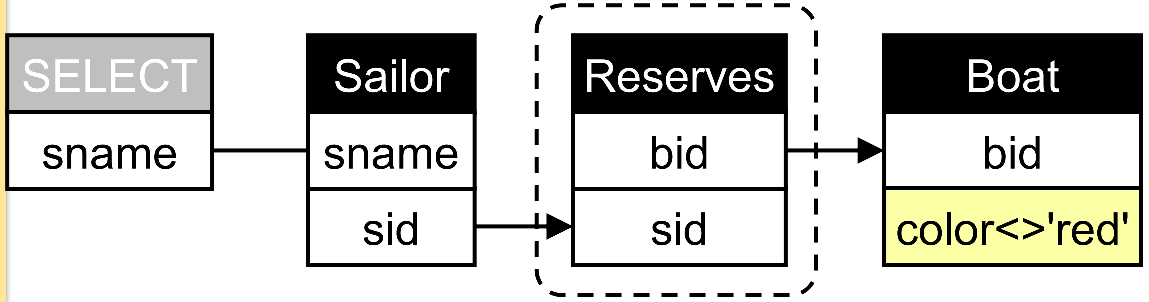
Nested query 4 (another variant)

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

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

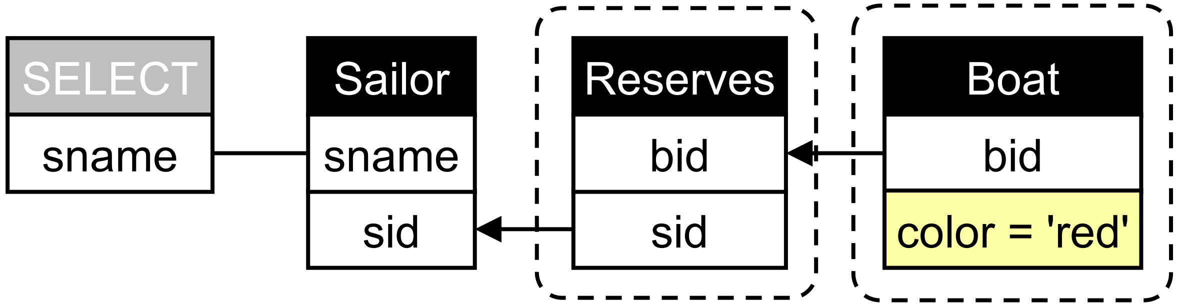
Nested query 5

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

340

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

Nested query 5

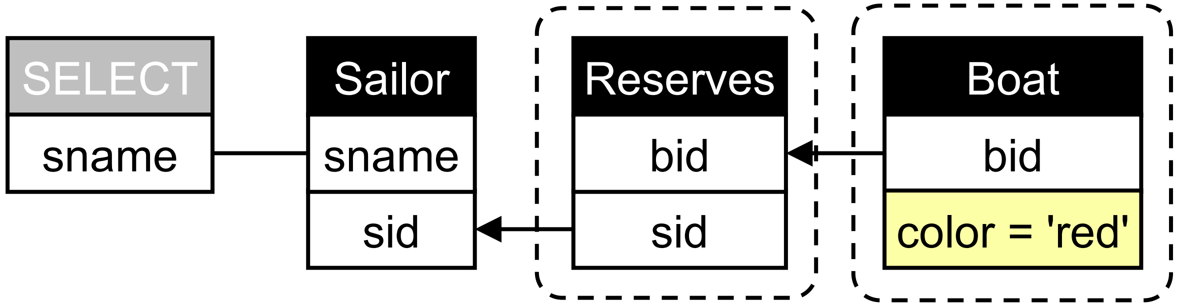
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

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 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)

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

Nested query 5 (universal)

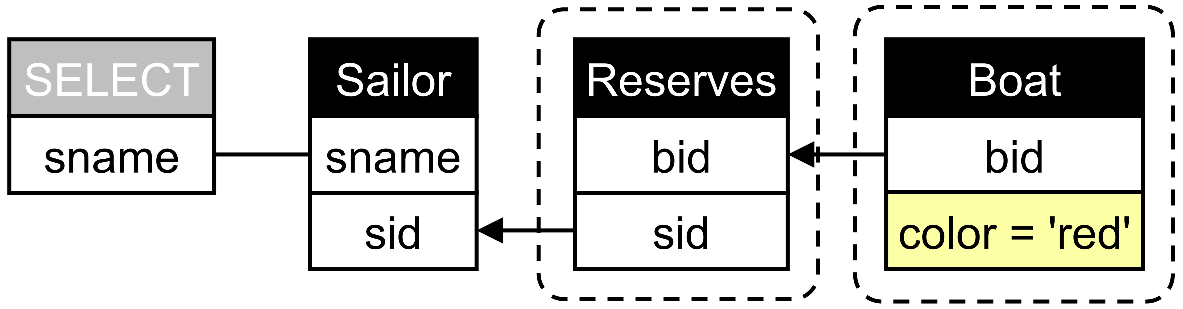
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

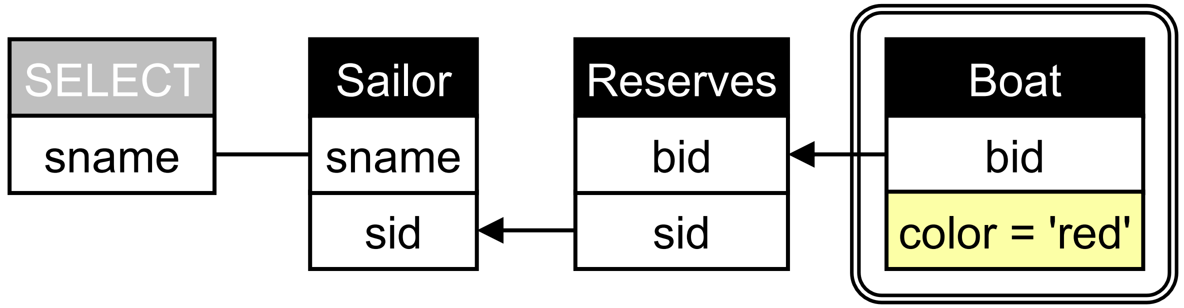
340

= 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 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)))}

{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)

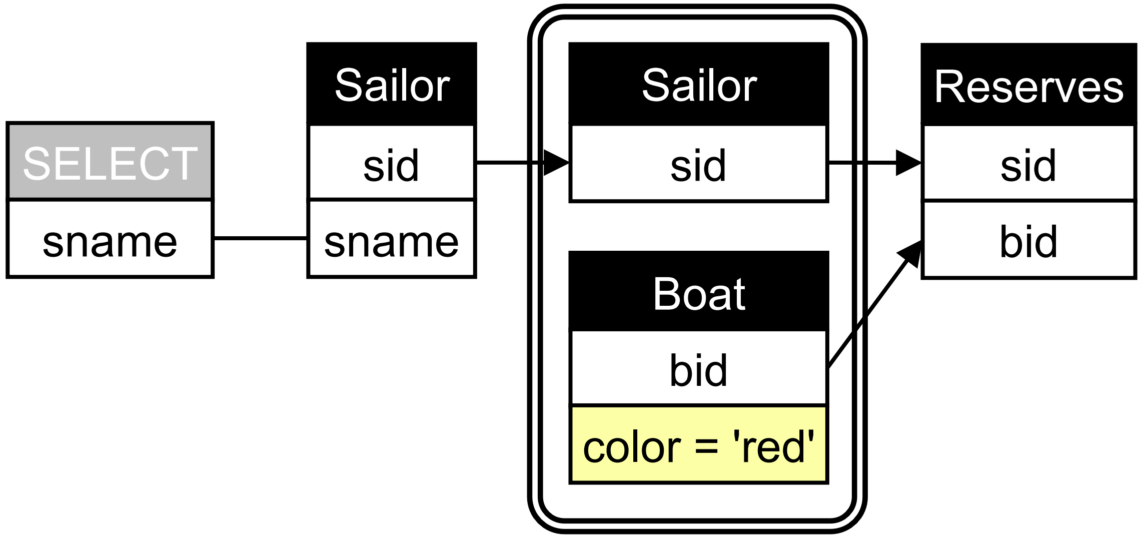
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

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)

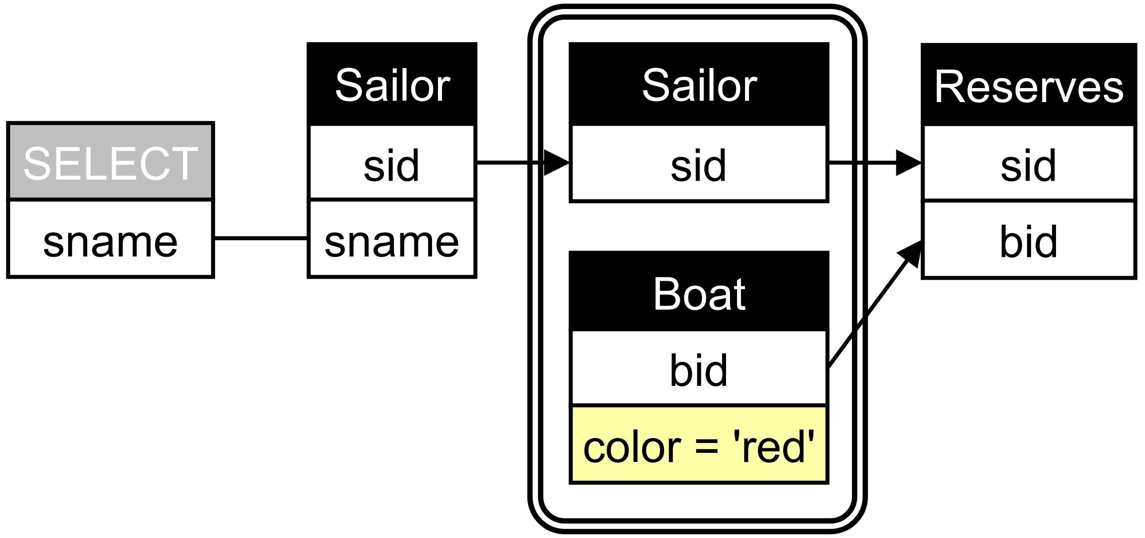
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

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

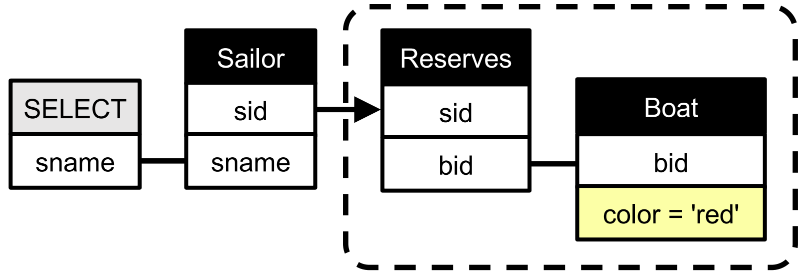
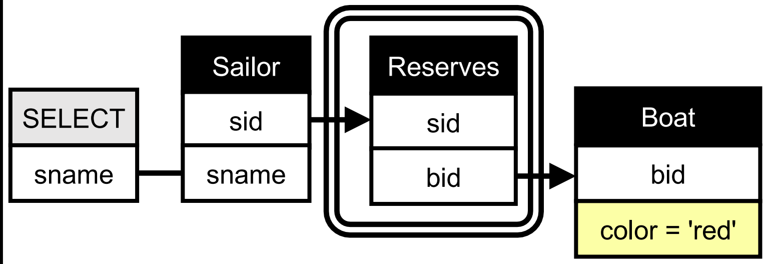
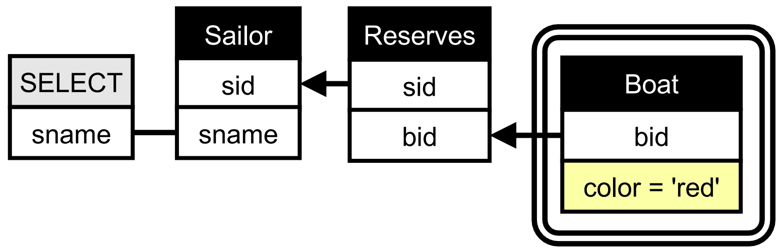
	Sailors who **have not** reserved a red boat	Sailors who reserved **only** red boats	Sailors who reserved **all** red boats
	SELECT DISTINCT S.sname	SELECT DISTINCT S.sname	SELECT DISTINCT S.sname
	FROM Sailor S	FROM Sailor S	FROM Sailor S
	WHERE NOT EXISTS(	WHERE NOT EXISTS(	WHERE NOT EXISTS(
	SELECT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

QV	 SELECT sname Sailor sid sname Reserves sid bid Boat bid color = 'red'	 SELECT sname Sailor sid sname Reserves sid bid Boat bid color = 'red'	 SELECT sname Sailor sid sname Reserves sid bid Boat bid color = 'red'

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)

	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 (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)

	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))

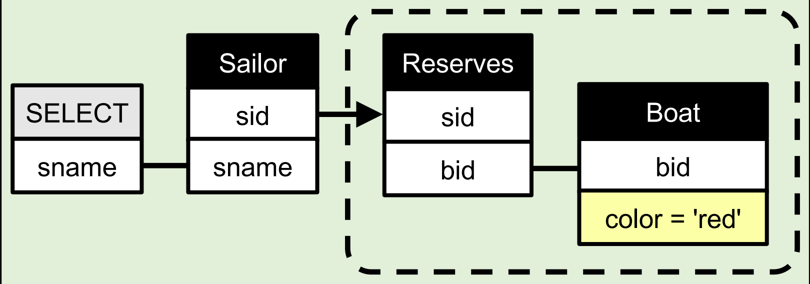
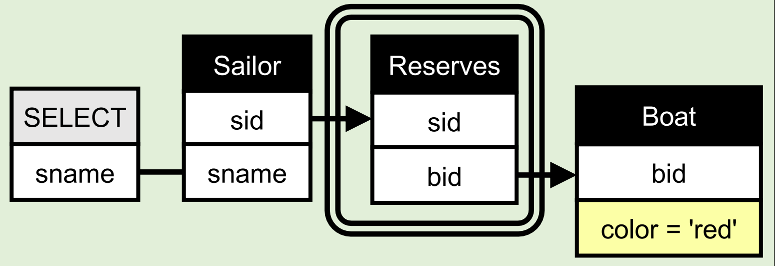
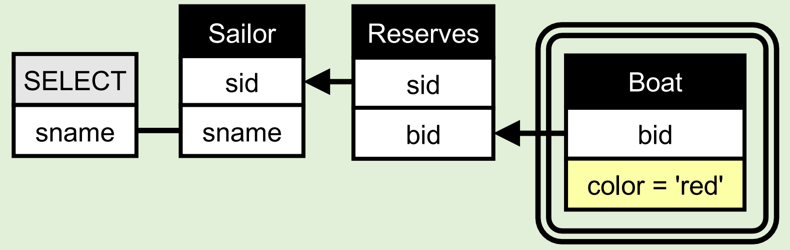
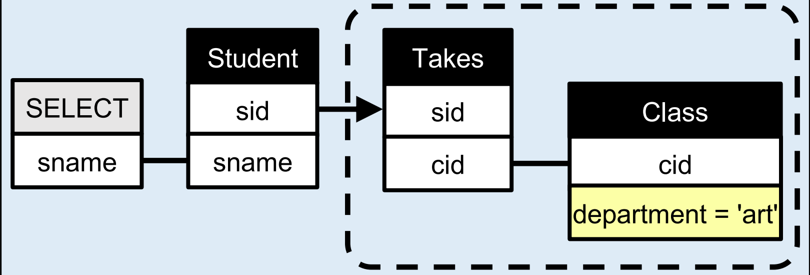
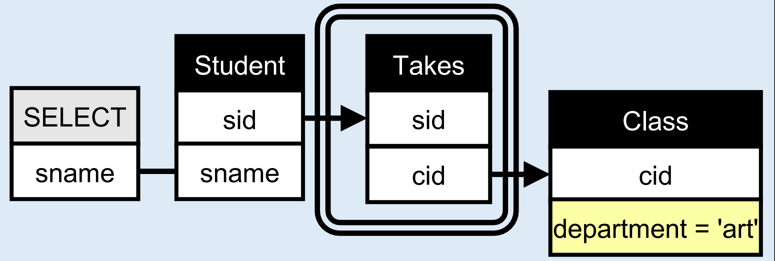
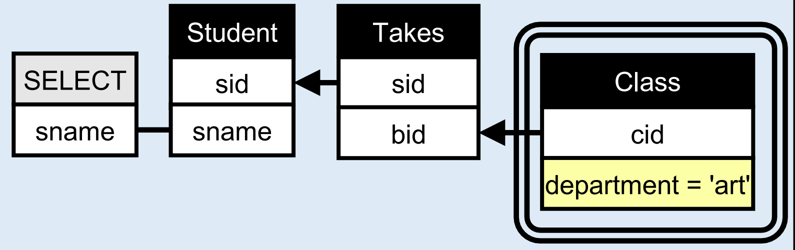
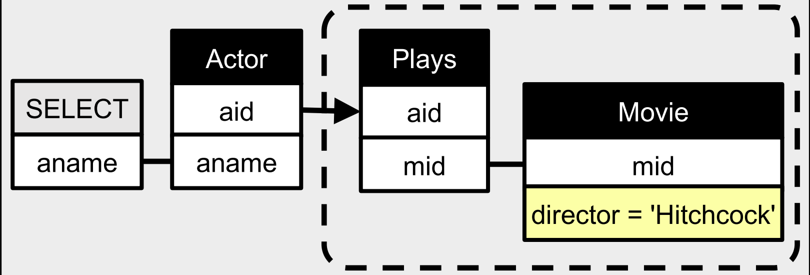
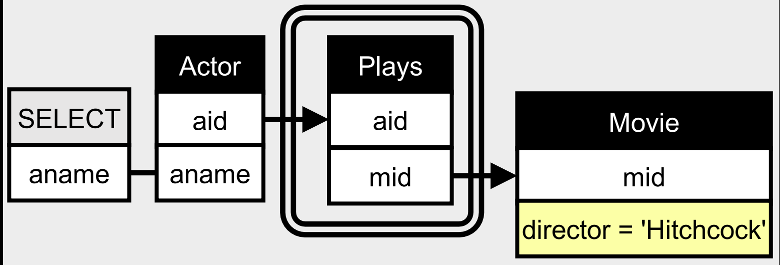
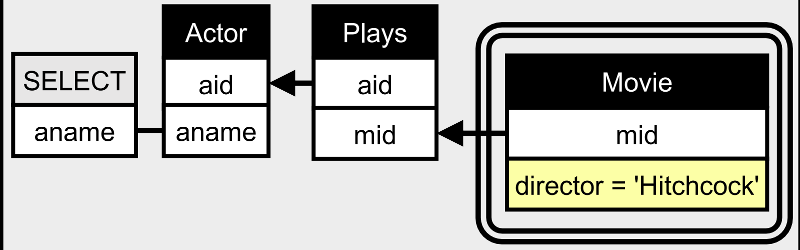
Students	SELECT DISTINCT S.sname
	FROM Student S
	WHERE NOT EXISTS(
	SELECT *
	FROM Takes T, Class C
	WHERE T.sid = S.sid
	AND C.cid = T.cid
	AND C.department ='art')
		SELECT DISTINCT S.sname
		FROM Student S
		WHERE NOT EXISTS(
		SELECT *
		FROM Takes T
		WHERE T.sid = S.sid
		AND NOT EXISTS(
		SELECT *
		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)

	not	**only**	**all**
Sailors	 SELECT sname Sailor sid sname Reserves sid bid Boat bid color = 'red'	 SELECT sname Sailor sid sname Reserves sid bid Boat bid color = 'red'	 SELECT sname Sailor sid sname Reserves sid bid Boat bid color = 'red'
Students	 SELECT sname Student sid sname Takes sid cid Class cid department = 'art'	 SELECT sname Student sid sname Takes sid cid Class cid department = 'art'	 SELECT sname Student sid sname Takes sid bid Class cid department = 'art'
Actors	 SELECT aname Actor aid aname Plays aid mid Movie mid director = 'Hitchcock'	 SELECT aname Actor aid aname Plays aid mid Movie mid director = 'Hitchcock'	 SELECT aname Actor aid aname Plays aid mid Movie mid director = 'Hitchcock'

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)

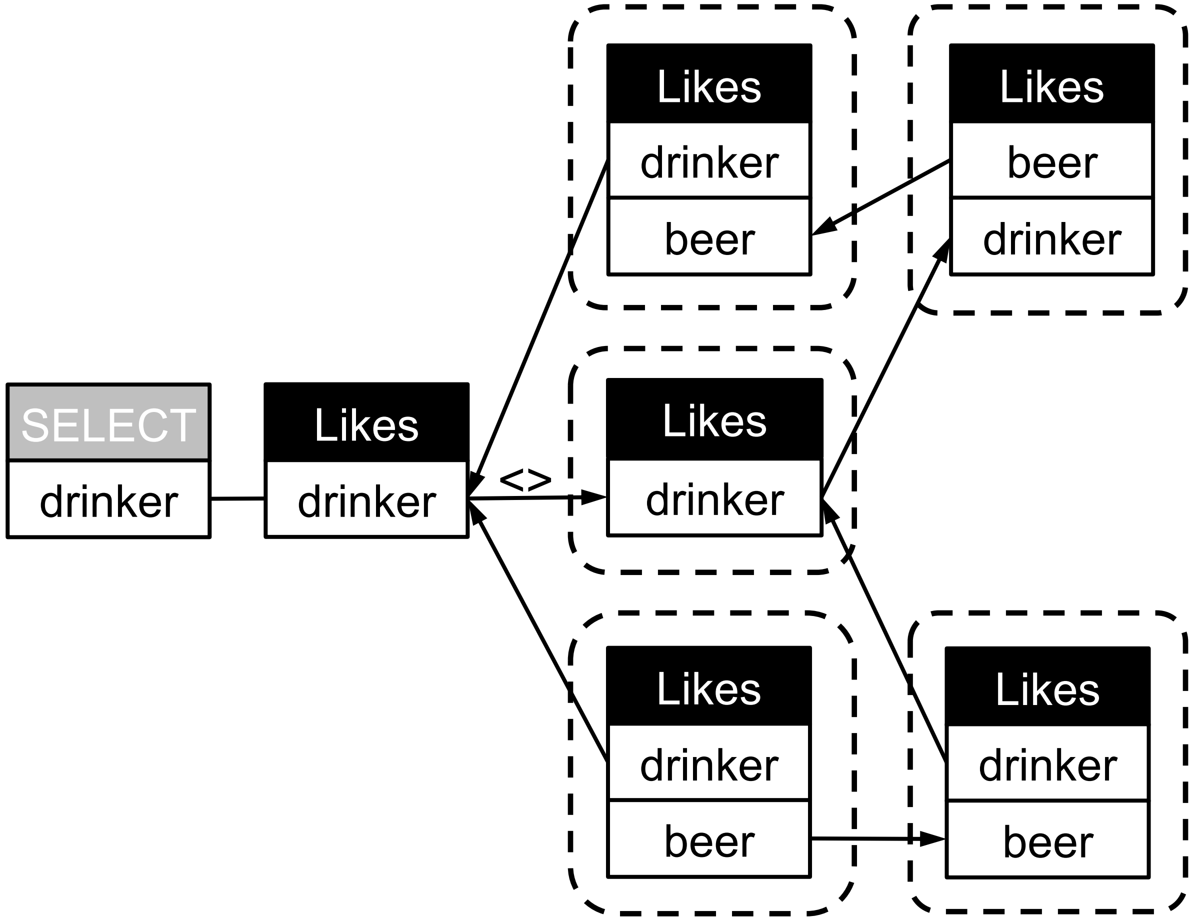
```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer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

QueryVis scoping

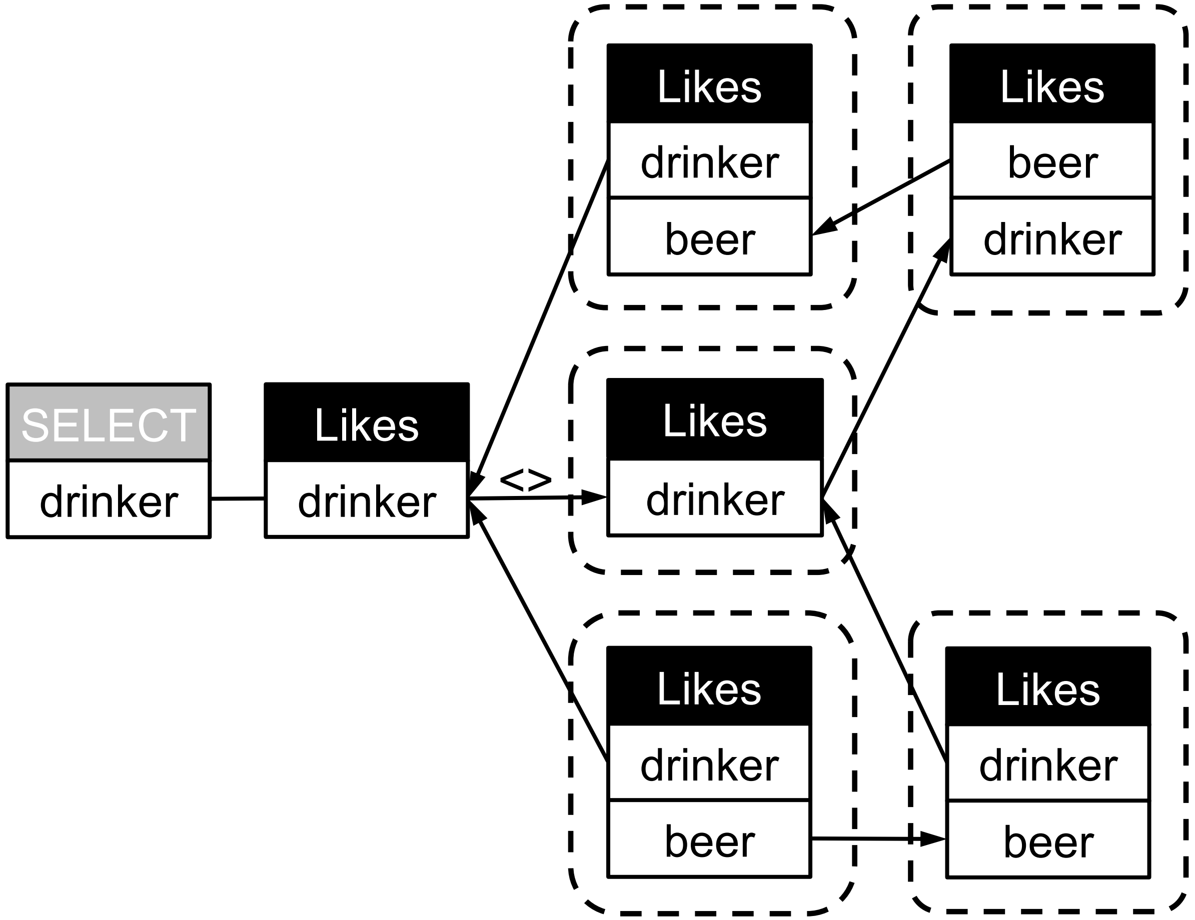


Q: Finder drinkers with a unique beer taste

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)))
```

QueryVis scoping

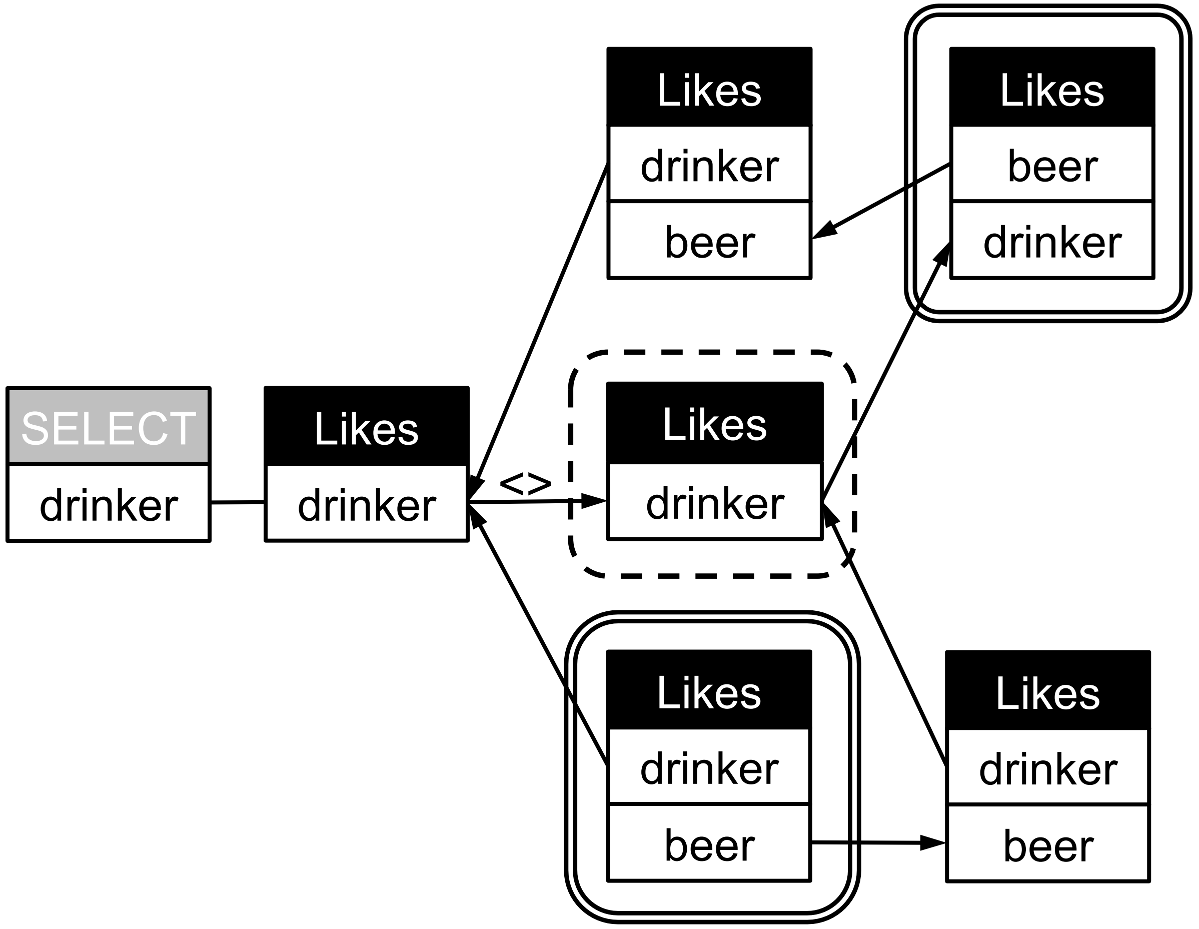


Q: Finder drinkers with a unique beer taste

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)))
```

QueryVis scoping



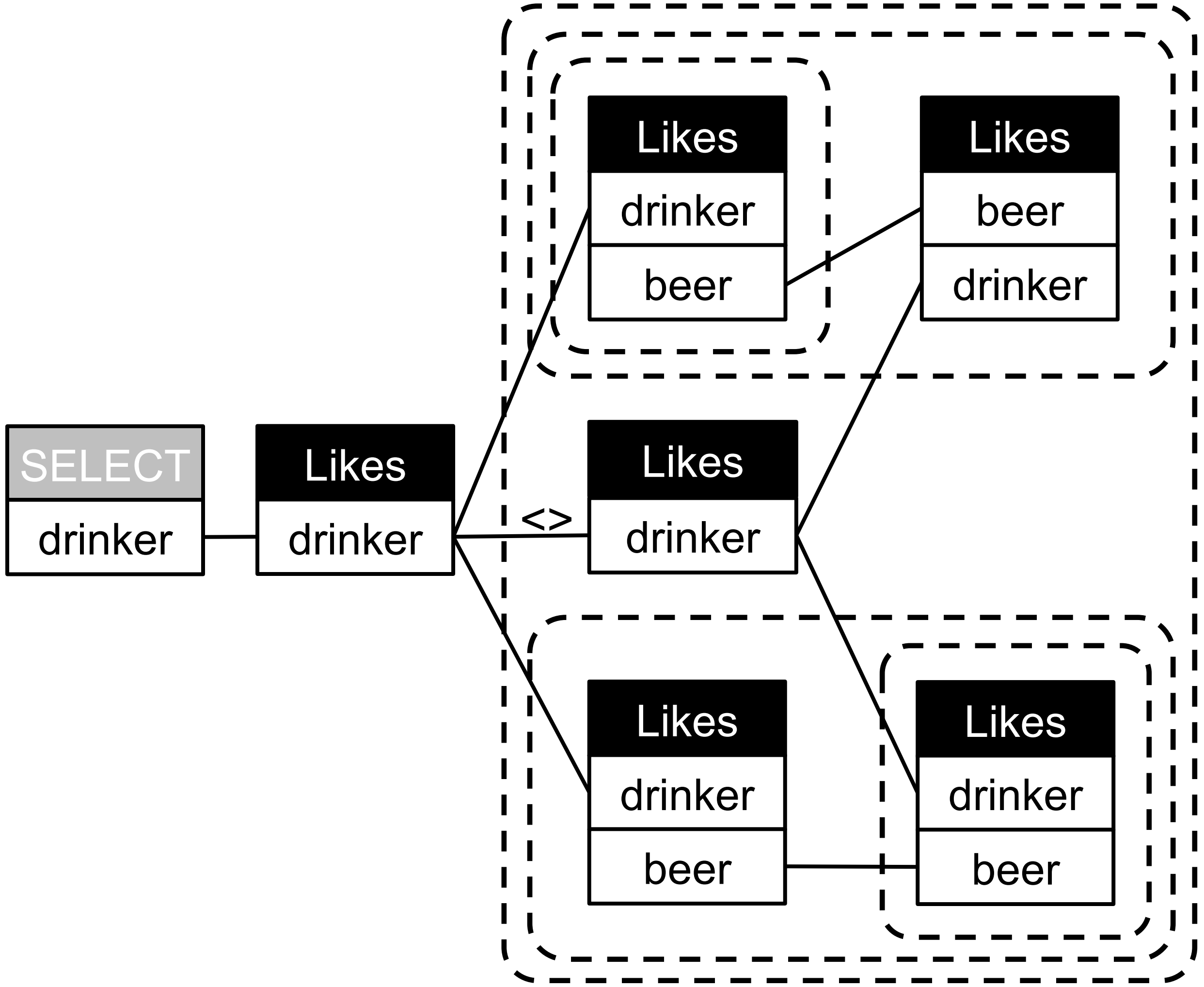
Q: Finder drinkers with a unique beer taste

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)))
```

QueryVis scoping

Relational Diagrams scoping



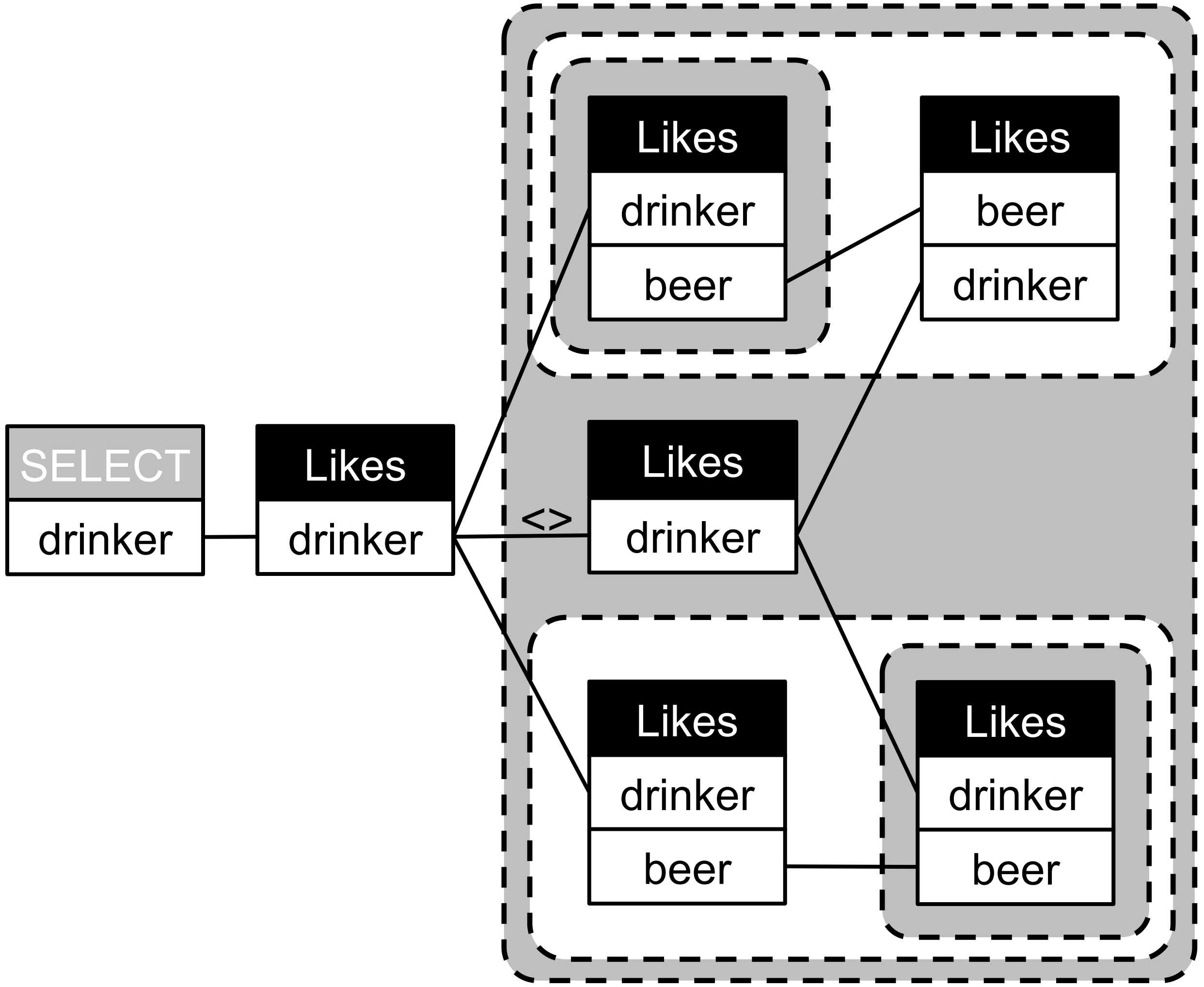
Q: Finder drinkers with a unique beer taste

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)))
```

QueryVis scoping

Relational Diagrams scoping



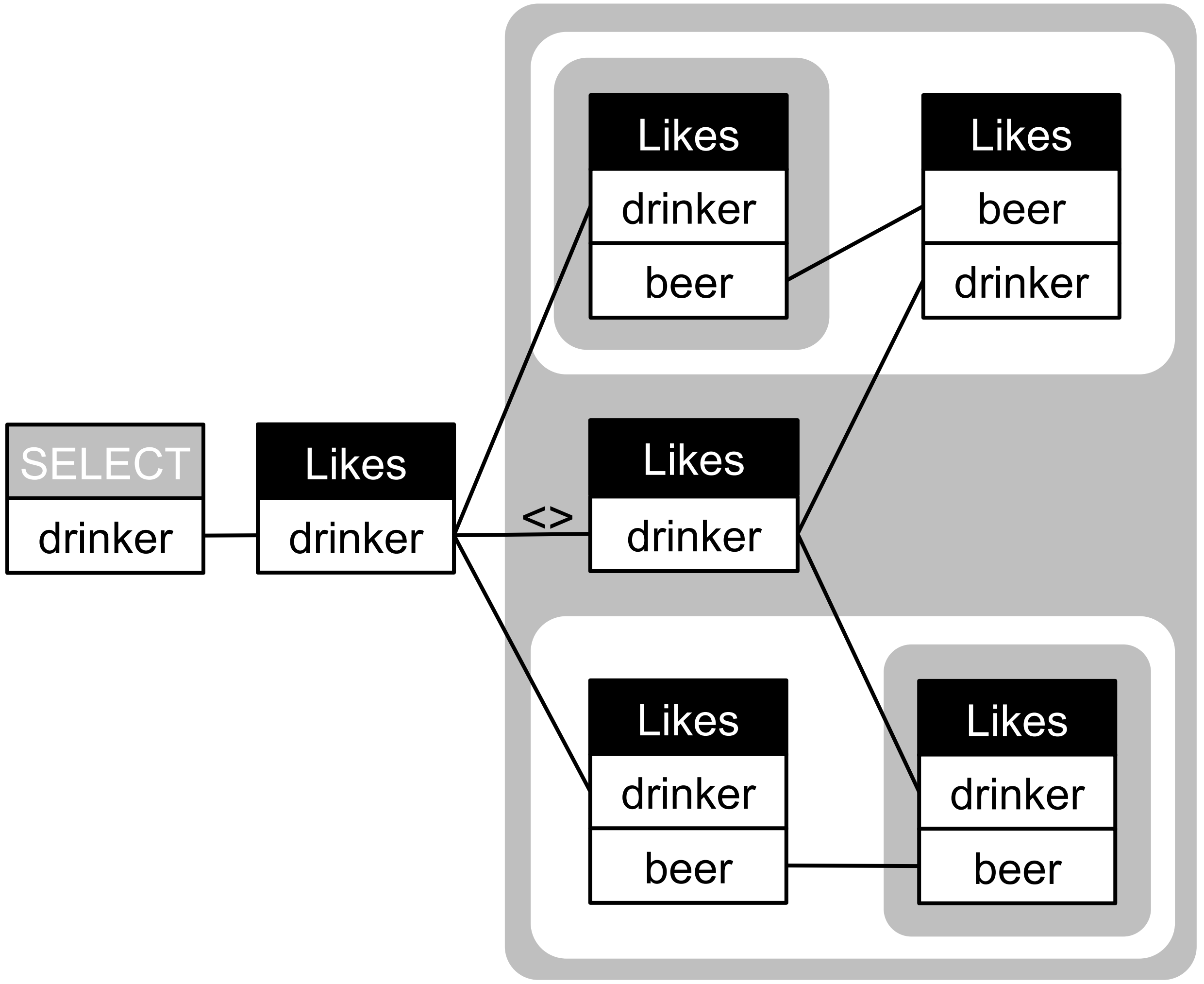
Q: Finder drinkers with a unique beer taste

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)))
```

QueryVis scoping

Relational Diagrams scoping

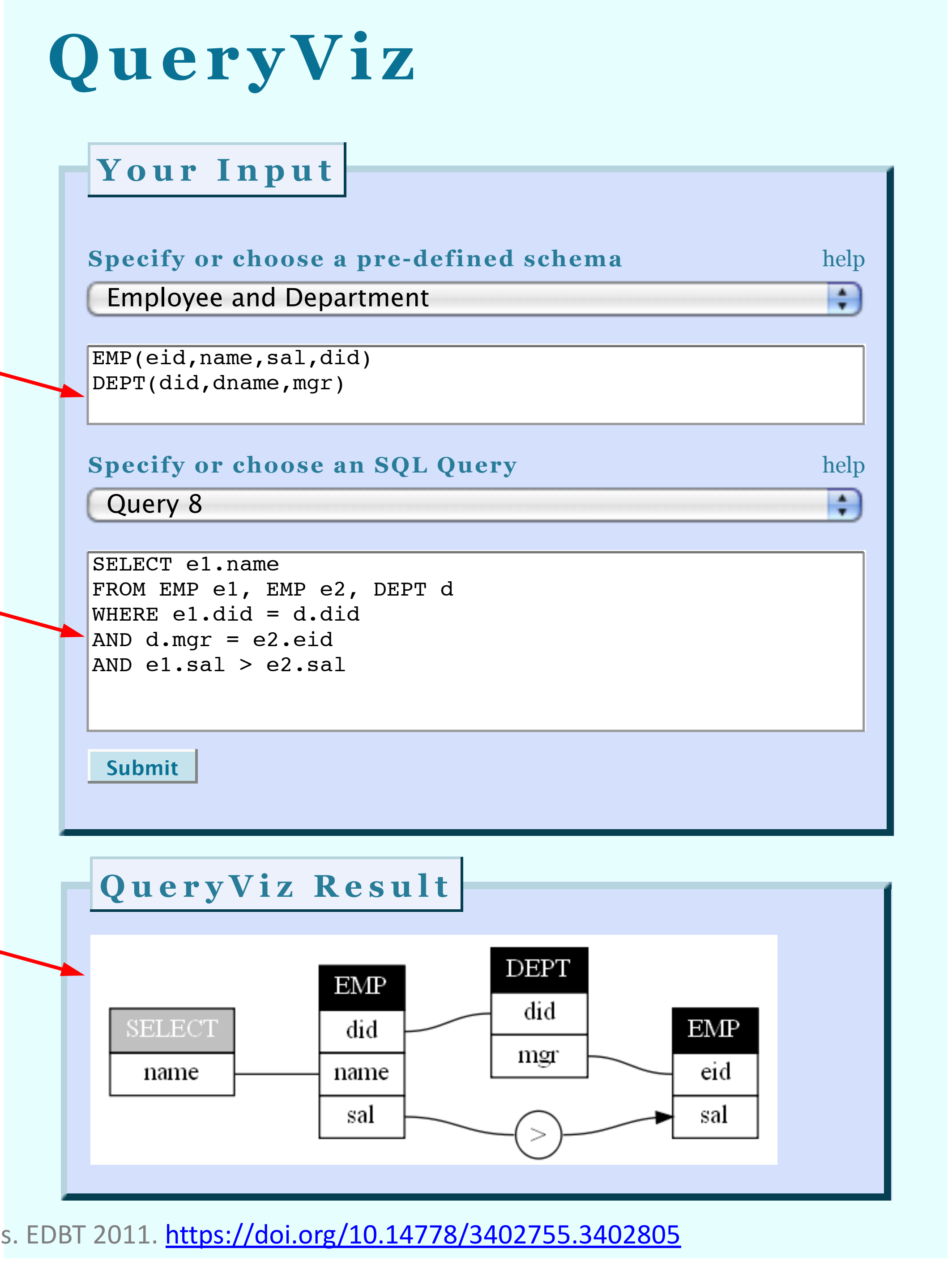


https://demo.queryvis.com

Input: Schema

Input Query

Output: Visualization



QueryViz

Your Input

Specify or choose a pre-defined schema help

Employee and Department

```
EMP(eid,name,sal,did)
DEPT(did,dname,mgr)
```

Specify or choose an SQL Query help

Query 8

```
SELECT e1.name
FROM EMP e1, EMP e2, DEPT d
WHERE e1.did = d.did
AND d.mgr = e2.eid
AND e1.sal > e2.sal
```

Submit

QueryViz Result

Danaparamita, G. [EDBT'11]

https://queryvis.com/

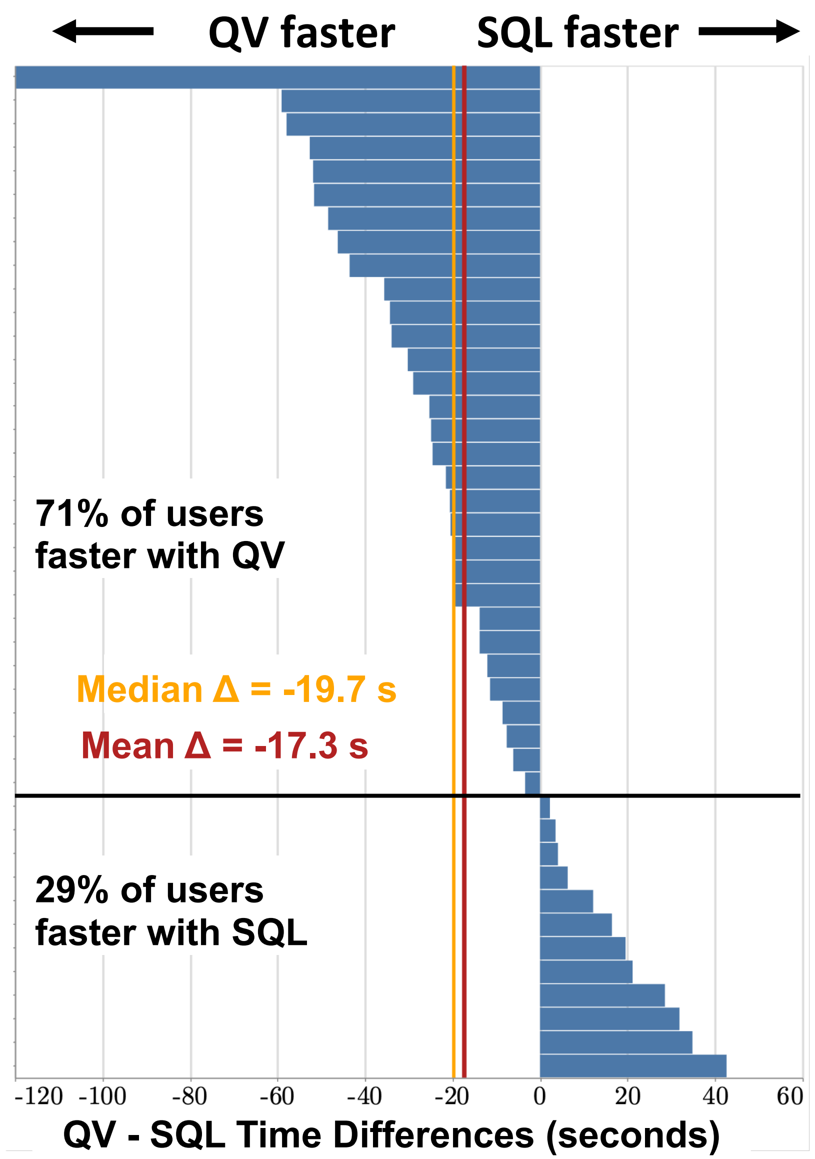
http://www.youtube.com/watch?v=kVFnQRGAQls

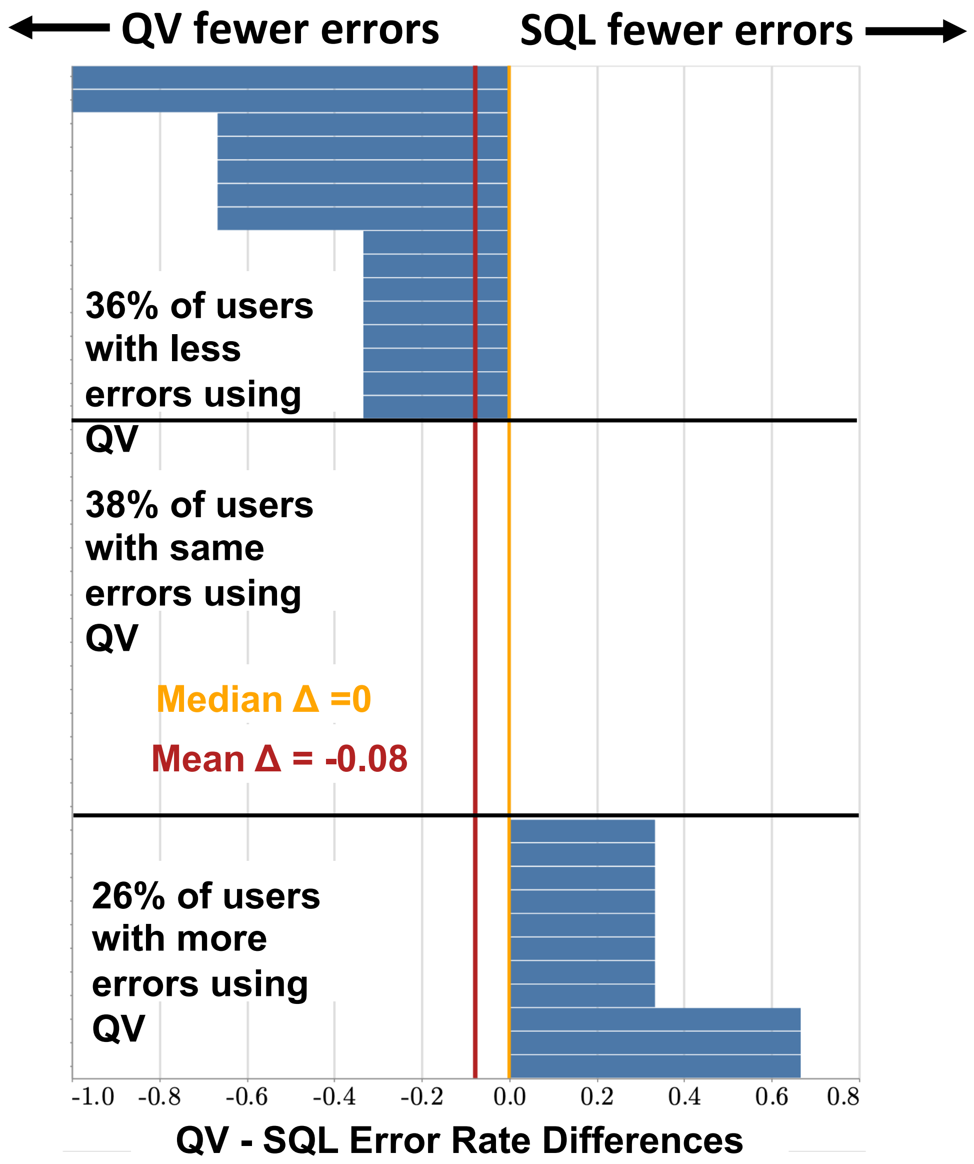
Source: Danaparamita, Gatterbauer: QueryViz: Helping users understand SQL queries and their patterns. EDBT 2011. https://doi.org/10.14778/3402755.3402805

Amazon Turk user study with SQL users

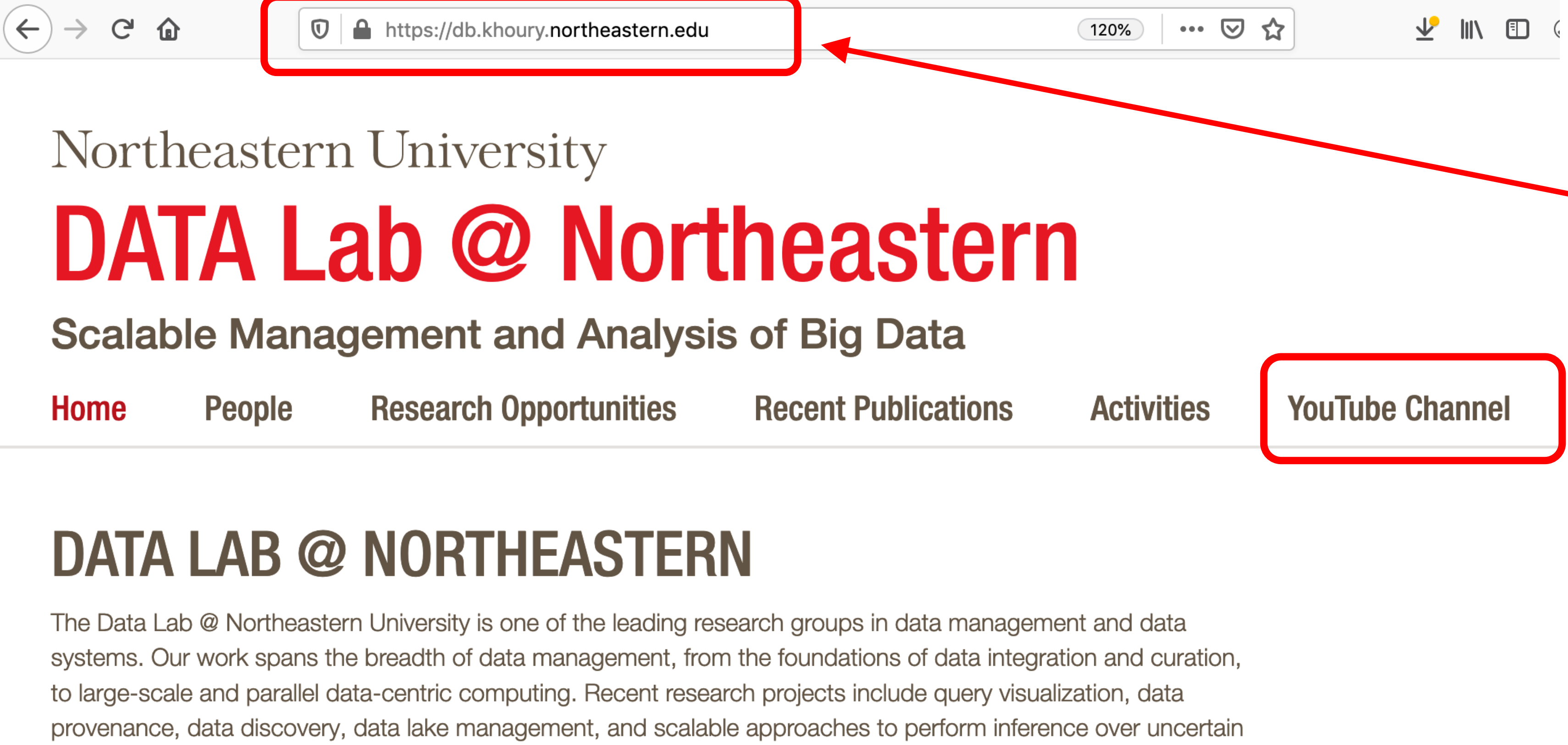
Leventidis+ [SIGMOD'20]

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://db.khoury.northeastern.edu

Northeastern University

DATA Lab @ Northeastern

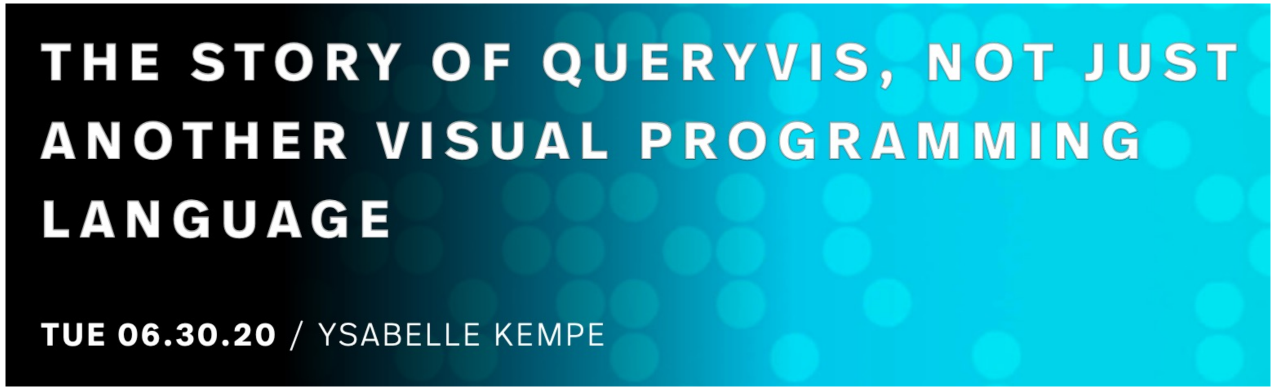
Scalable Management and Analysis of Big Data

Home People Research Opportunities Recent Publications Activities YouTube Channel

DATA LAB @ NORTHEASTERN

The Data Lab @ Northeastern University is one of the leading research groups in data management and data systems. Our work spans the breadth of data management, from the foundations of data integration and curation, to large-scale and parallel data-centric computing. Recent research projects include query visualization, data provenance, data discovery, data lake management, and scalable approaches to perform inference over uncertain

https://queryvis.com



THE STORY OF QUERYVIS, NOT JUST ANOTHER VISUAL PROGRAMMING LANGUAGE

TUE 06.30.20 / YSABELLE KEMPE

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

Practice with groupings

Grouping variants

Person

L	F	M	V
Smith	Alice	C.	1
Smith	Alice	NULL	2
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT   L, F, M, max(V) MV
FROM     Person
GROUP BY L, F, M
```

⇨ ?

```
SELECT   L, F, max(V) MV
FROM     Person
GROUP BY L, F
```

⇨ ?

```
SELECT   L, max(V) MV
FROM     Person
GROUP BY L
```

⇨ ?

```
SELECT max(V) MV
FROM   Person
```

⇨ ?