Updated 1/23/2021

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

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp21)

https://northeastern-datalab.github.io/cs7240/sp21/

1/22/2021

Pre-class conversations

- Last class recapitulation
- Any questions on class procedures?
  - You will see some "minimum examples" today in class

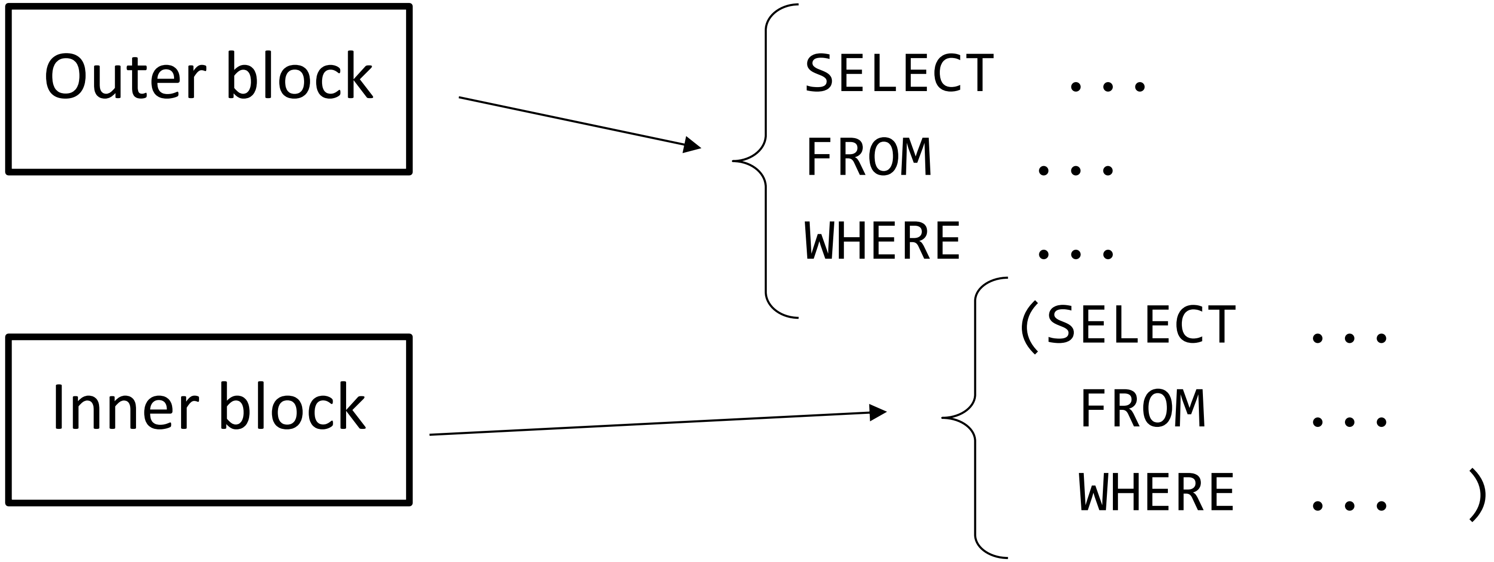
- today:
  - SQL continued (with connection to table integration)
  - perhaps start of calculus

Outline: SQL (a refresher)

- SQL
  - Schema and keys
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Outer joins
  - Top-k

Subqueries = Nested queries

We mostly focus on nestings in the WHERE clause, which is the most expressive type of nesting



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

- We can nest queries because SQL is compositional:
  - Everything (inputs / outputs) is represented as multisets
  - the output of one query can thus be used as the input to another (nesting)
  - Subqueries return relations
- This is extremely powerful!
- It gets more complicated with correlated nested queries

3. Subqueries in WHERE

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries compute?

```
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)
```

⇨ ?

3. Subqueries in WHERE

305

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries compute?

```
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)
```

⇨ ?

3. Subqueries in WHERE

What do these queries compute?

R

a
1
2

W

a	b
2	0
3	0
4	0

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

a
2

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

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

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)
```

?

3. Subqueries in WHERE

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries compute?

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

a
2

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

{(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.
  - 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.

Correlated subquery (existential)

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

Existential quantifiers ∃

Q: Find all companies that make some product~~s~~ with price < 25!

Using IN:

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

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery (existential)

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

Existential quantifiers ∃

Q: Find all companies that make some products with price < 25!

Using IN:

"Set membership"

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

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery (existential)

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

Existential quantifiers ∃

Q: Find all companies that make some products with price < 25!

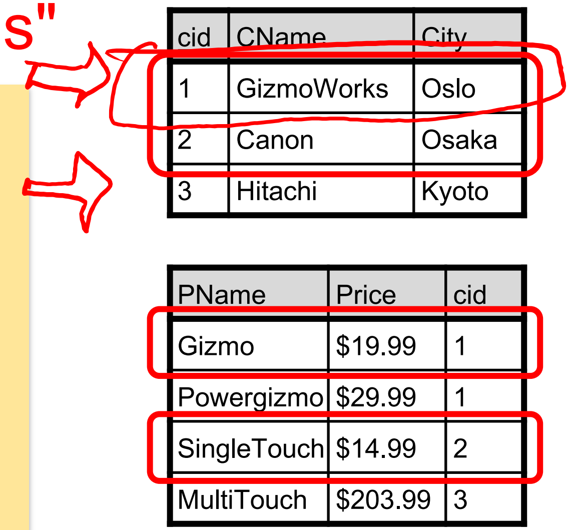
EXISTS is true iff the subquery's result is not empty

Using EXISTS:

"Test for empty relations"

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

Correlated subquery



cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery (existential)

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

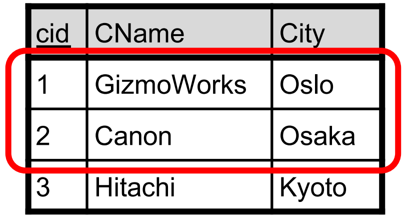
Existential quantifiers ∃

Q: Find all companies that make some products with price < 25!

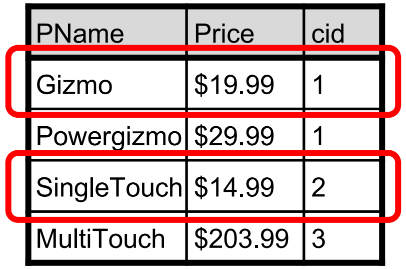
Using ANY (also some):

"Set comparison"

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



cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto



PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery

SQLlite does not support "ANY" ☹

Correlated subquery (existential)

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

Existential quantifiers ∃

Q: Find all companies that make some products with price < 25!

Now, let's unnest:

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

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Existential quantifiers are easy ! ☺

Correlated subquery (universal)

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

Universal quantifiers ∀

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

same as:

Q: Find all companies for which all products have price < 25!

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

Correlated subquery (exist not -> universal)

315

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

1. Find the other companies: i.e. they have some product ≥ 25!

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

2. Find all companies s.t. all their products have 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 (exist not -> universal)

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

Universal quantifiers ∀

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

Using NOT EXISTS:

```
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 (exist not -> universal)

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

Universal quantifiers ∀

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

Using ALL:

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

SQLlite does not support "ALL" ☹

A natural question

- How can we unnest the universal quantifier query ?

?

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

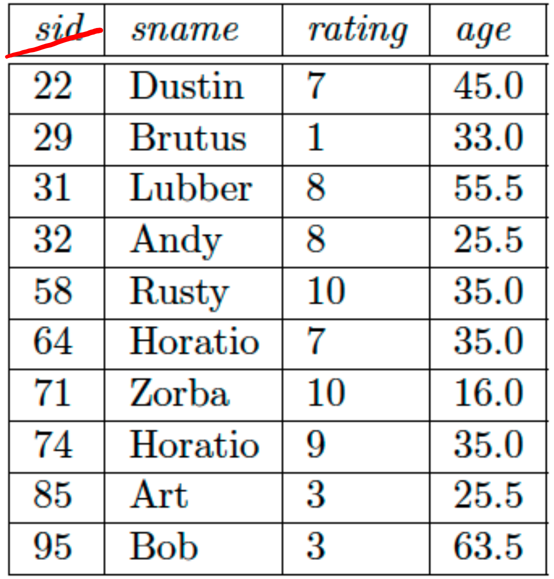
Understanding nested queries

The sailors database

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

340

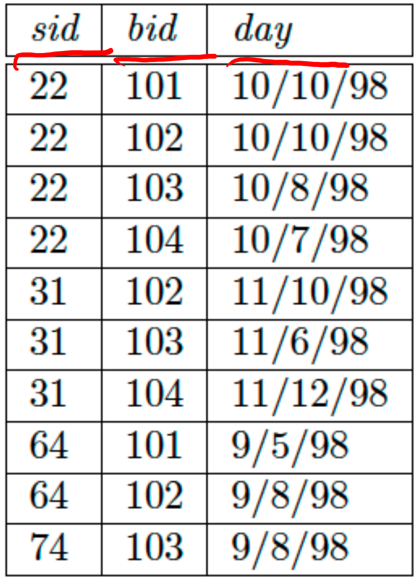
Sailor



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

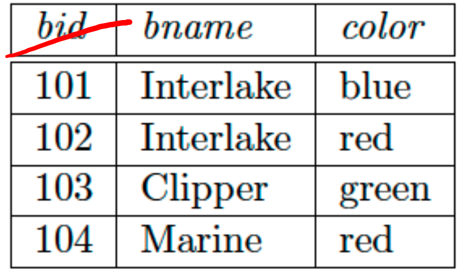
Reserves



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

22	101	10/11/98
22	102	10/11/98

Schema from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

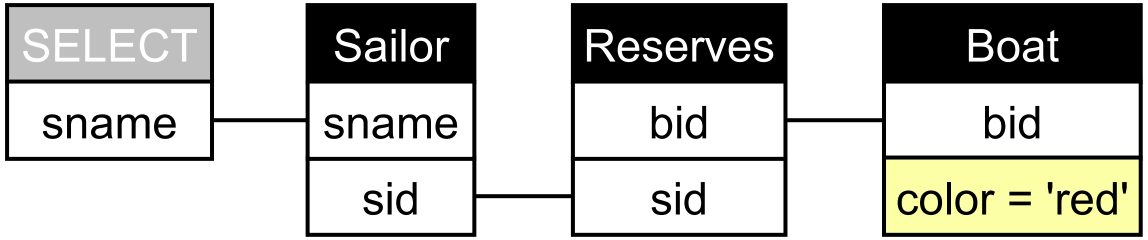
Nested query 1

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

340

Q: ?

```
SELECT 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'))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

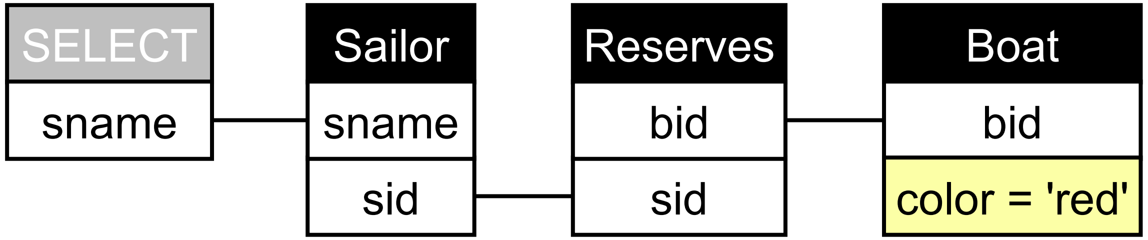
Nested query 1

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 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')))}

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

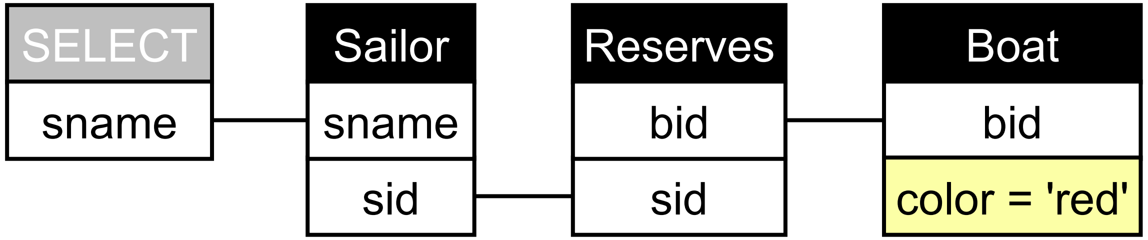
Nested query 1

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 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.

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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

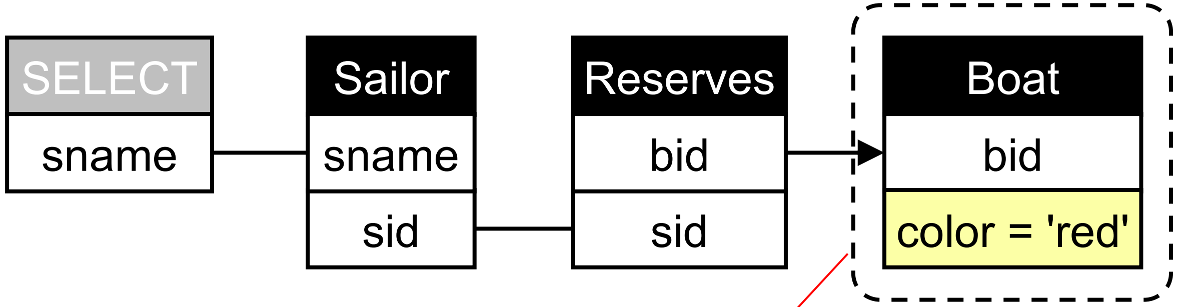
Nested query 2

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

340

Q: ?

```
SELECT 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 ∄

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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

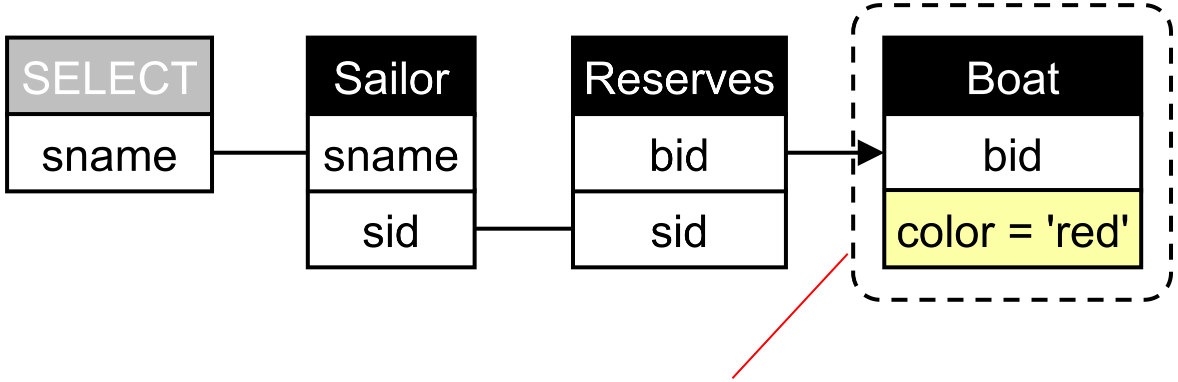
Nested query 2

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 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.

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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

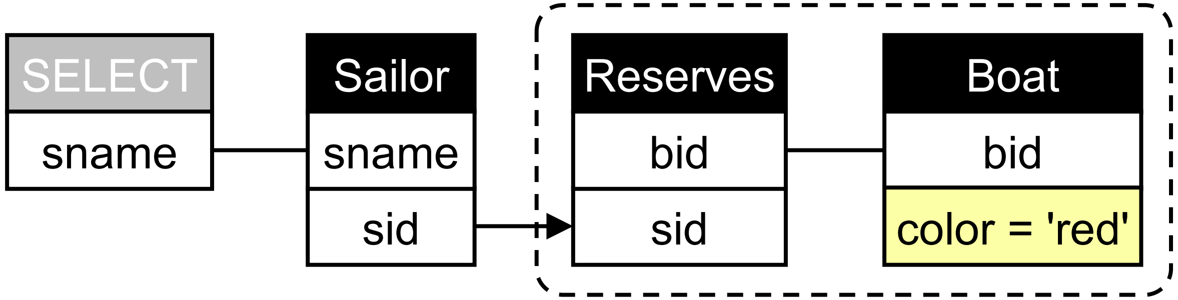
Nested query 3

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

340

Q: ?

```
SELECT 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'))
```



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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

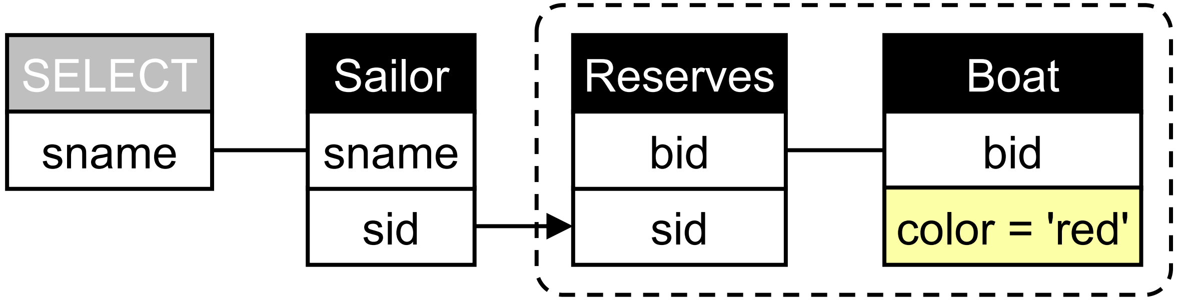
Nested query 3

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 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.

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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

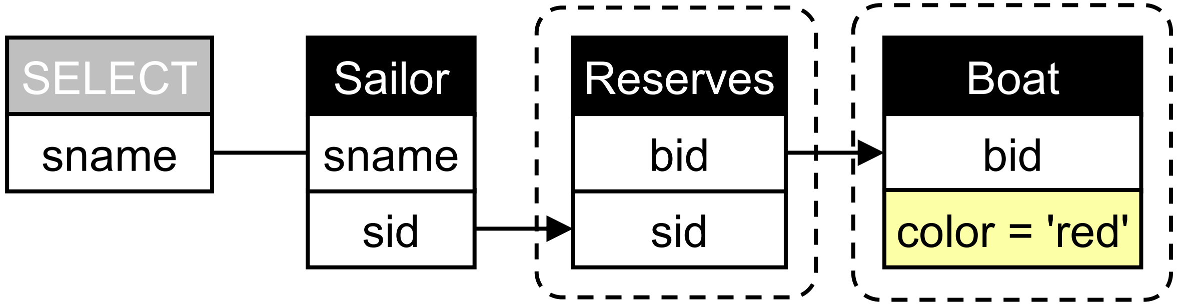
Nested query 4

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

340

Q: ?

```
SELECT 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'))
```



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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

Nested query 4

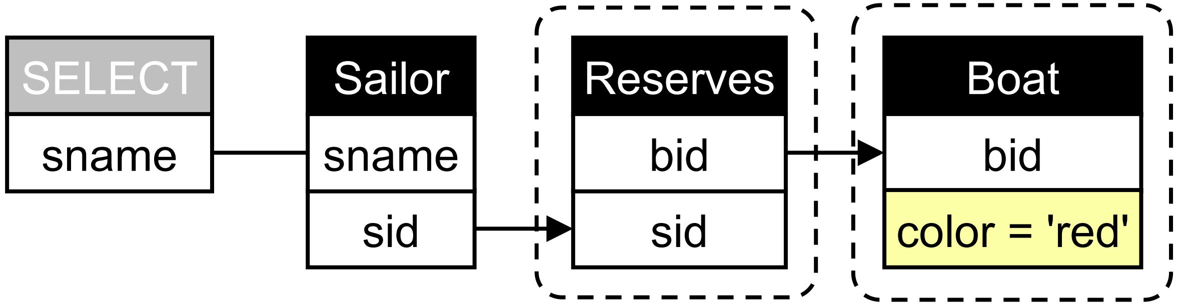
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

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 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.

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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

Nested query 4 (another variant)

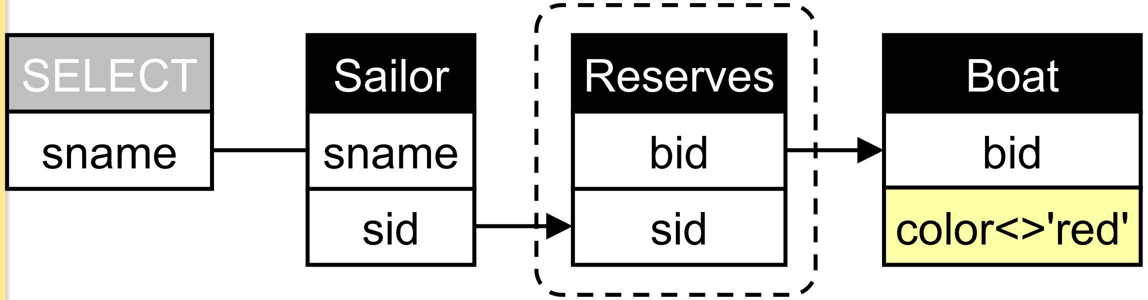
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

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 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.

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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

Nested query 4 (universal)

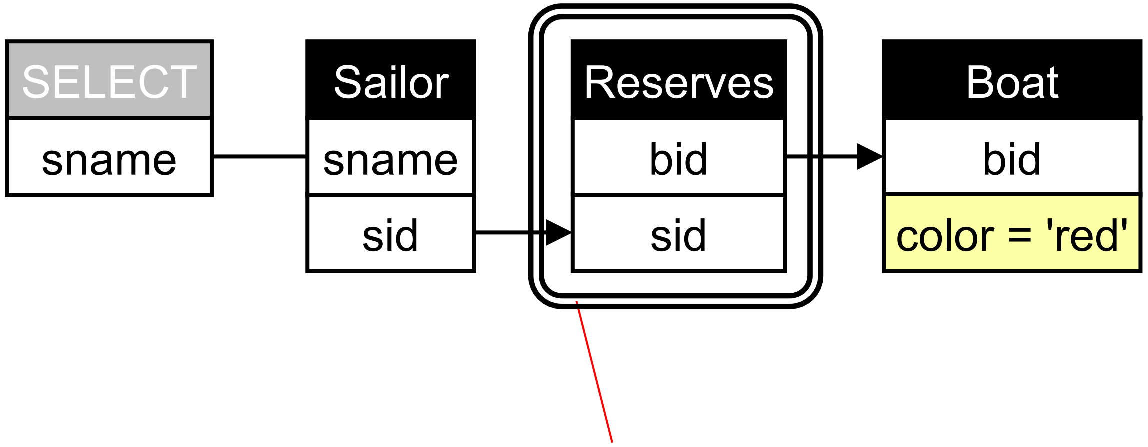
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

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 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 ∀

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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

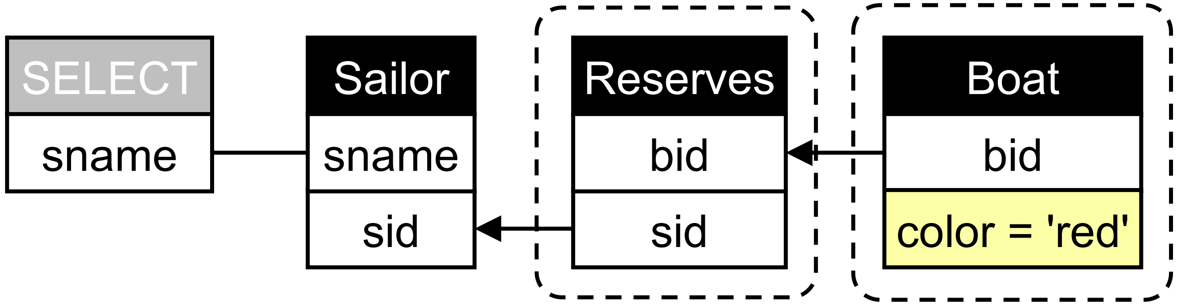
Nested query 5

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

340

Q: ?

```
SELECT 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))
```



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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

Nested query 5

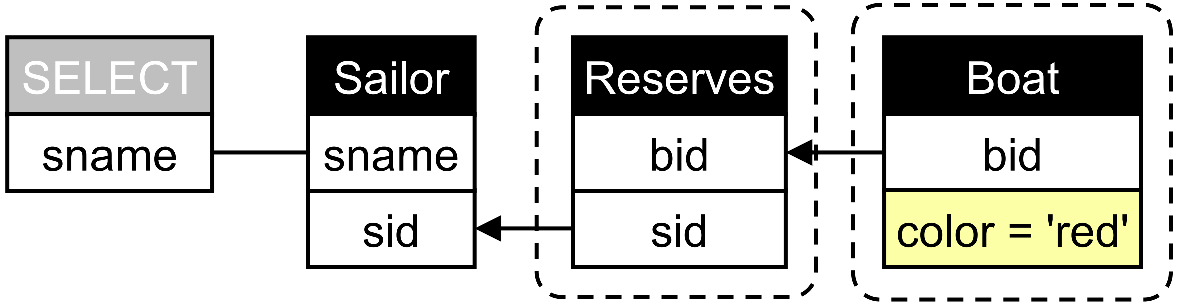
Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

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 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 later when we discuss this query in relational algebra.

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

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

Nested query 5 (universal)

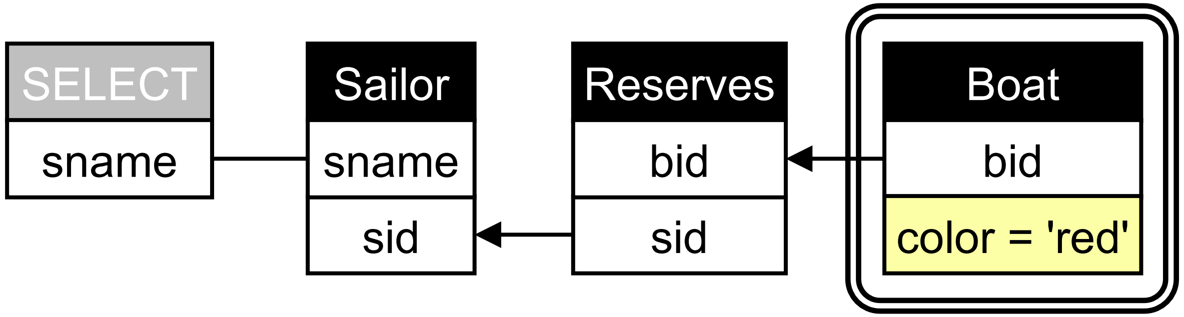
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

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 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))))}

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

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

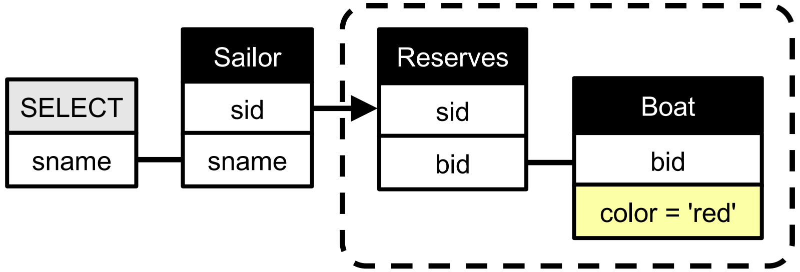
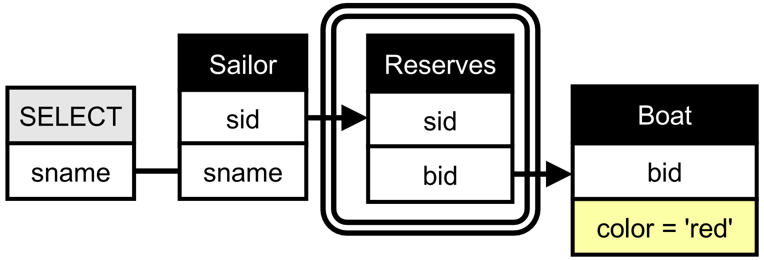
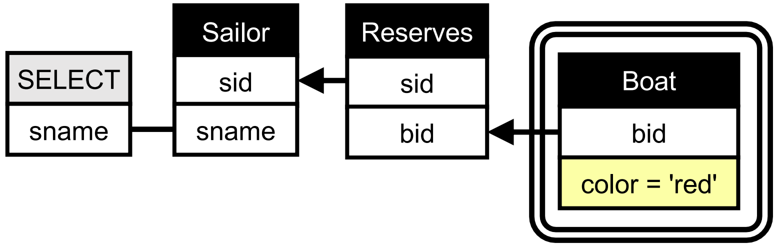
SQL	SELECT S.sname
	FROM Sailor S
	WHERE NOT EXISTS(
	SELECT *
	FROM Reserves R, Boat B
	WHERE R.sid = S.sid
	AND R.bid = B.bid
	AND B.color = 'red')
		SELECT S.sname
		FROM Sailor S
		WHERE NOT EXISTS(
		SELECT *
		FROM Reserves R
		WHERE R.sid = S.sid
		AND NOT EXISTS(
		SELECT *
		FROM Boat B
		WHERE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

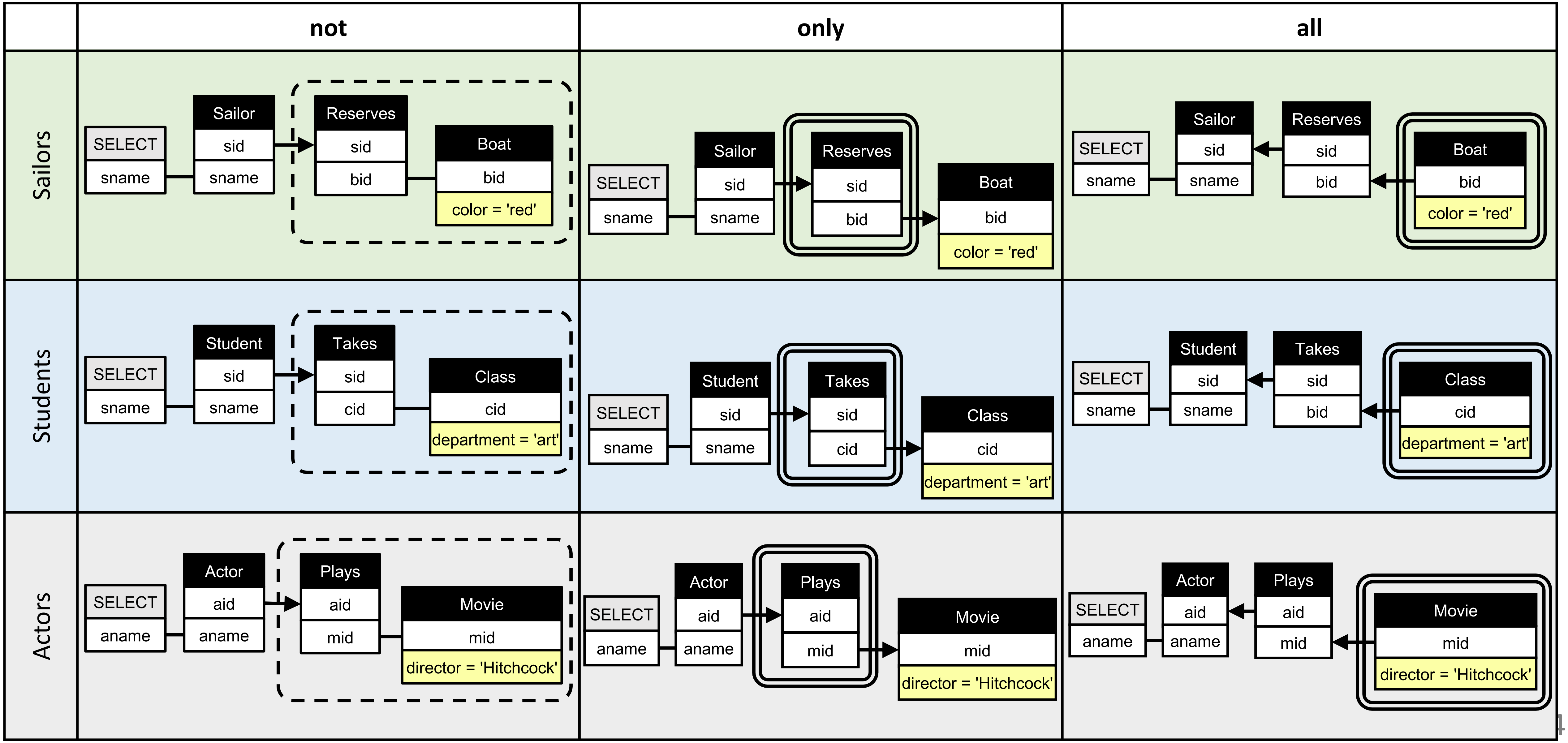
Students	SELECT S.sname
	FROM Student S
	WHERE NOT EXISTS(
	SELECT *
	FROM Takes T, Class C
	WHERE T.sid = S.sid
	AND C.cid = T.cid
	AND C.department ='art')
		SELECT S.sname
		FROM Student S
		WHERE NOT EXISTS(
		SELECT *
		FROM Takes T
		WHERE T.sid = S.sid
		AND NOT EXISTS(
		SELECT *
		FROM Class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

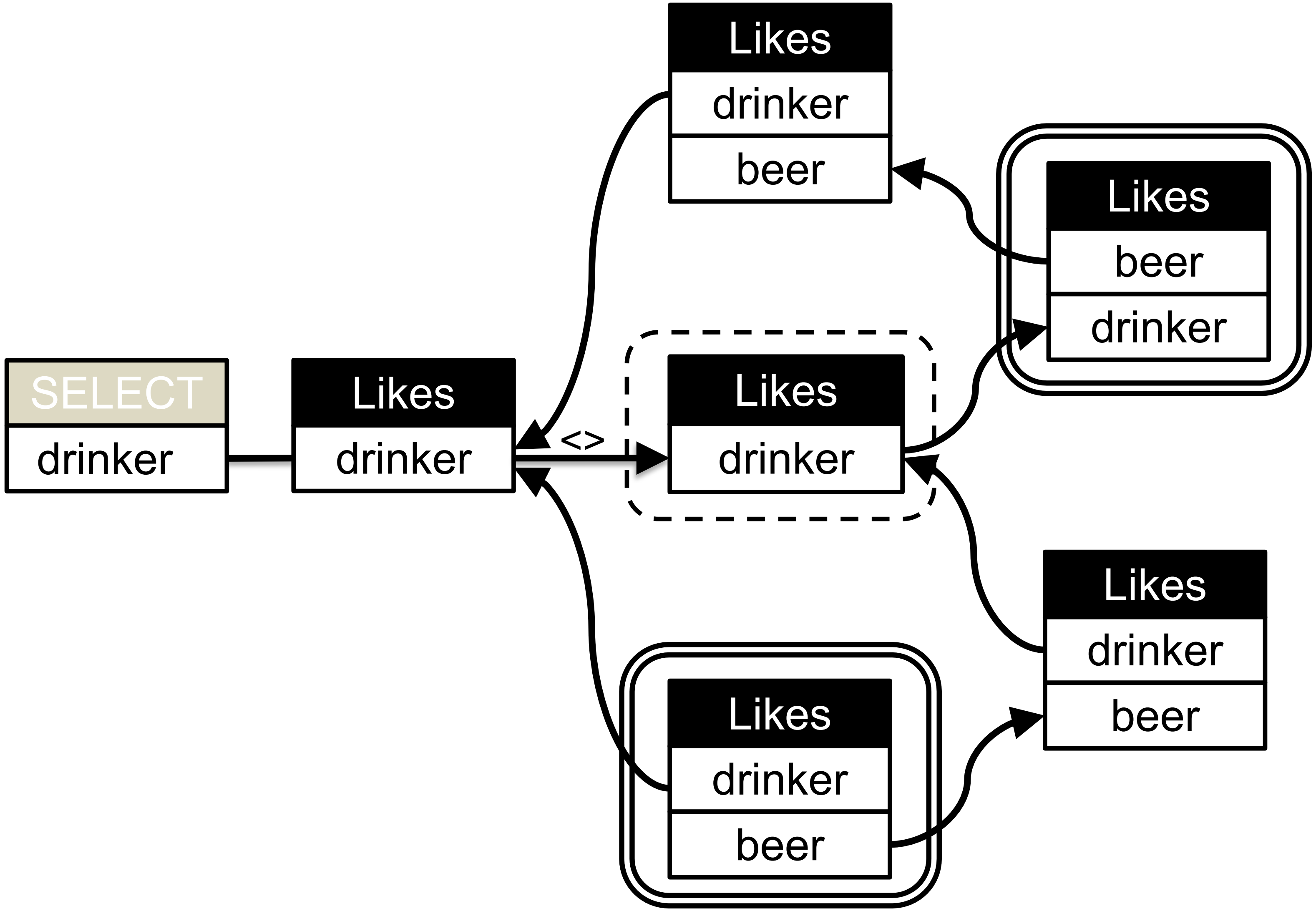
Likes (drinker, beer)

Nesting Depth

```
    SELECT  L1.drinker
0   FROM    Likes L1
    WHERE   NOT EXISTS(
            SELECT  *
1           FROM    Likes L2
            WHERE   L1.drinker <> L2.drinker
            AND     NOT EXISTS(
                    SELECT  *
2                   FROM    Likes L3
                    WHERE   L3.drinker = L2.drinker
                    AND     NOT EXISTS(
                            SELECT  *
3                           FROM    Likes L4
                            WHERE   L4.drinker = L1.drinker
                            AND     L4.beer = L3.beer))
            AND     NOT EXISTS(
                    SELECT  *
2                   FROM    Likes L5
                    WHERE   L5. drinker = L1. drinker
                    AND     NOT EXISTS(
                            SELECT  *
                            FROM    Likes L6
3                           WHERE   L6.drinker = L2.drinker
                            AND     L6.beer= L5.beer)))
```

Q: Finder drinkers with a unique beer taste

Likes (drinker, beer)

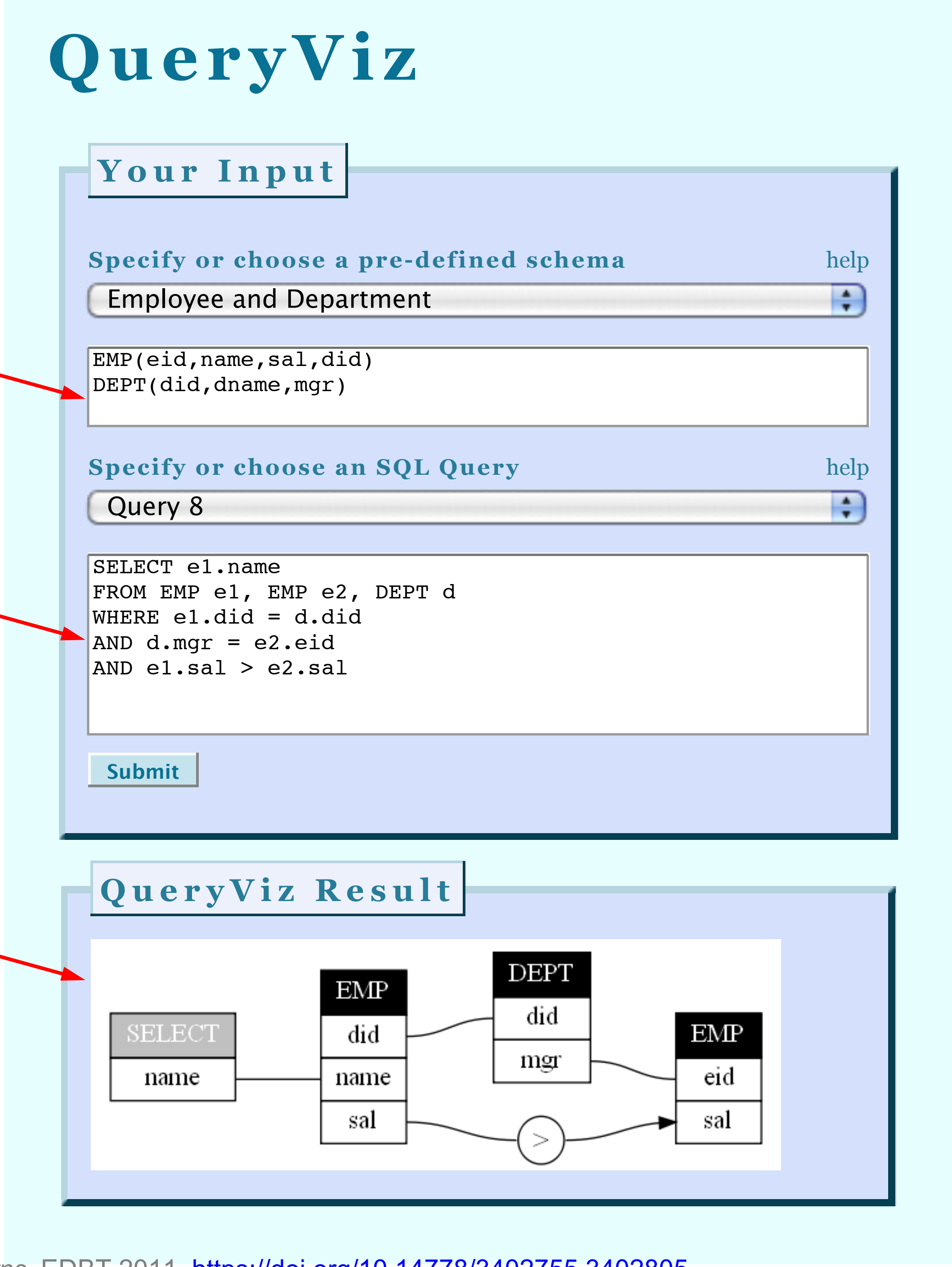


https://demo.queryvis.com

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/

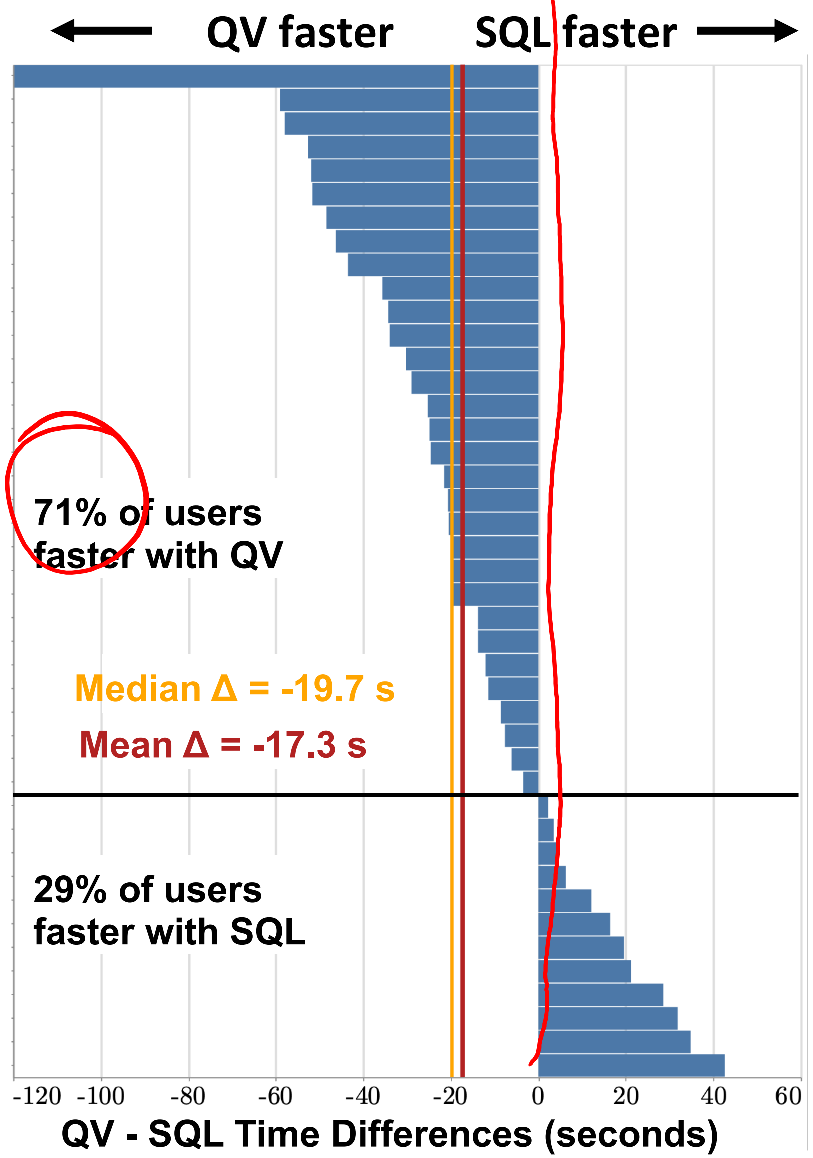
http://www.youtube.com/watch?v=kVFnQRGAQls

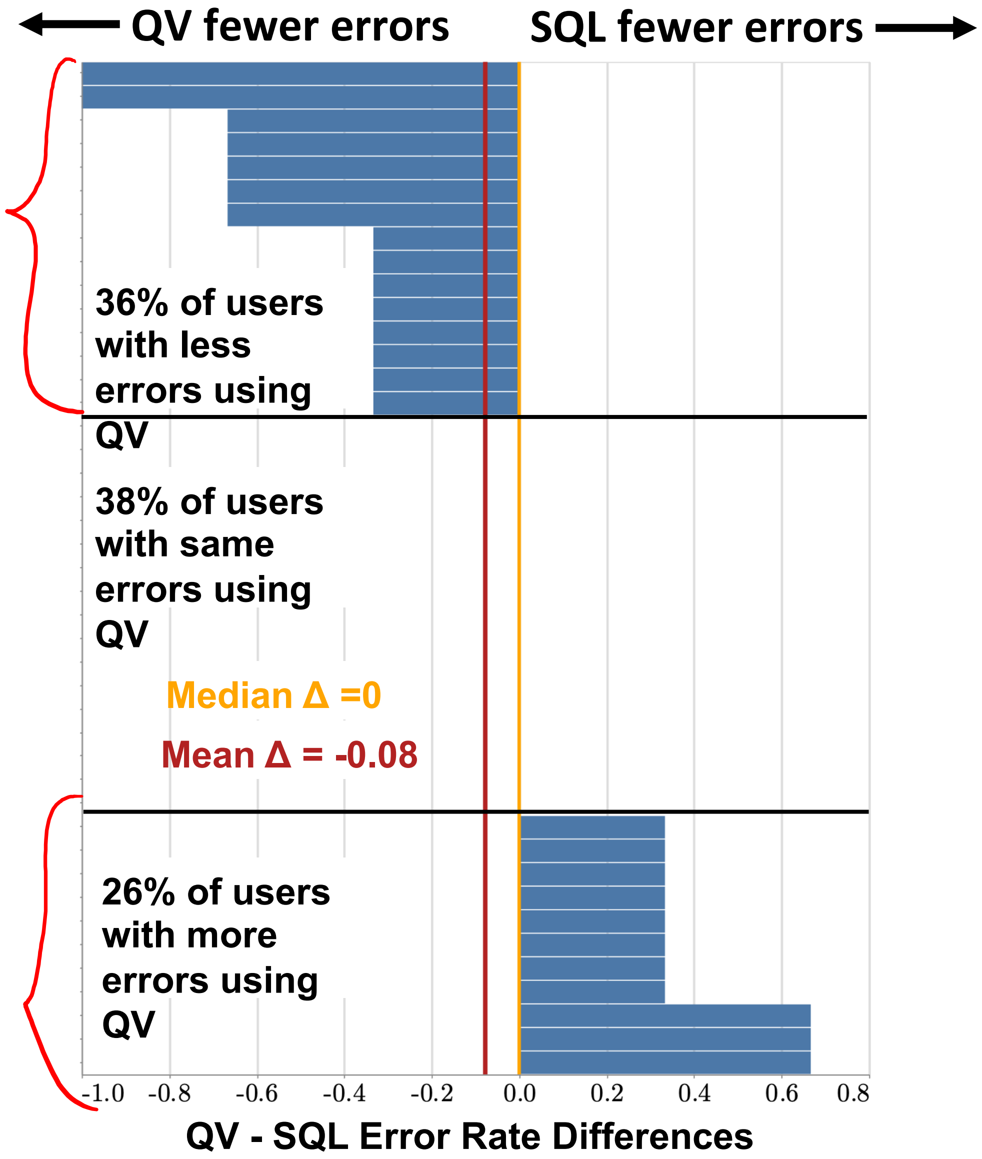
Source: Danaparamita, Gatterbauer: *QueryViz: Helping users understand SQL queries and their patterns*. EDBT 2011. https://doi.org/10.14778/3402755.3402805

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

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

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

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

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

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

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

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

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

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

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

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

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

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

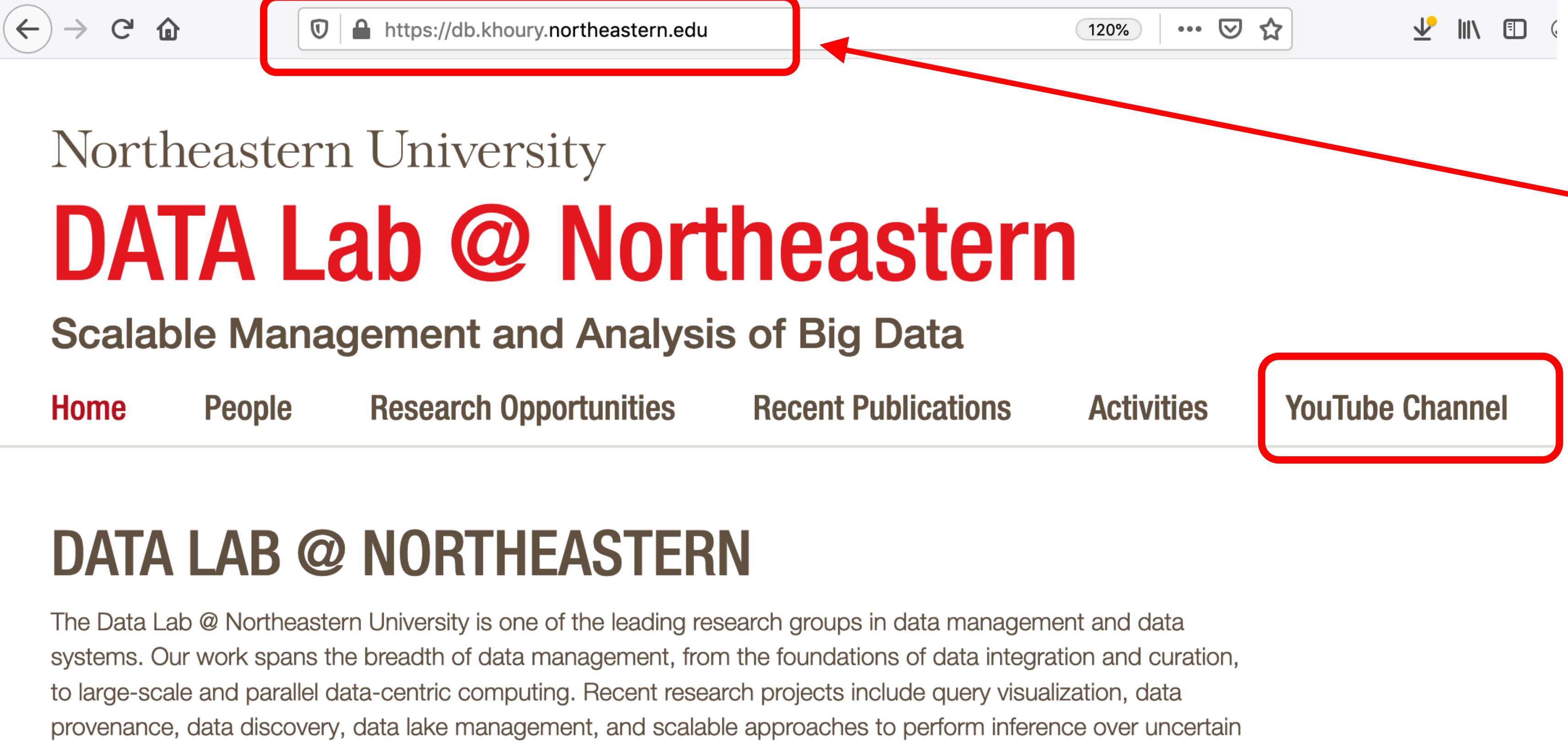
x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$

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

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

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

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



https://db.khoury.northeastern.edu

120%

Northeastern University

DATA Lab @ Northeastern

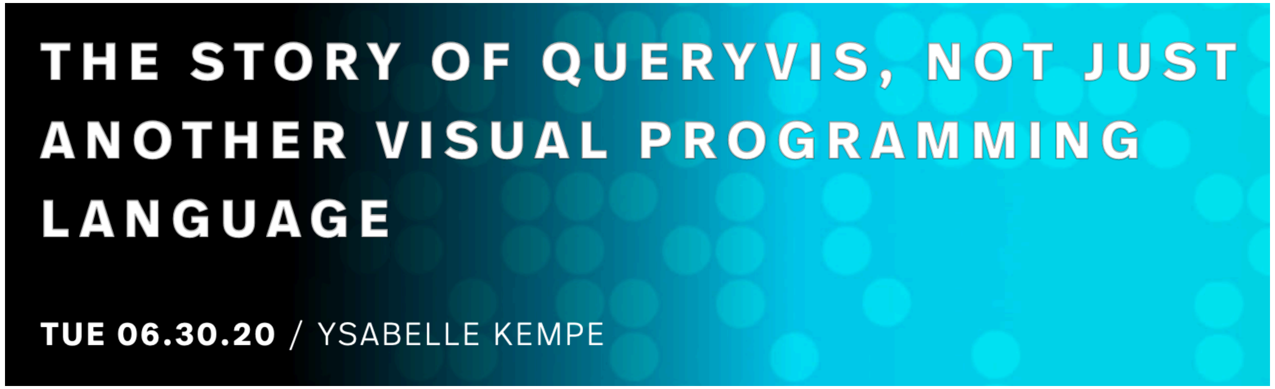
Scalable Management and Analysis of Big Data

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

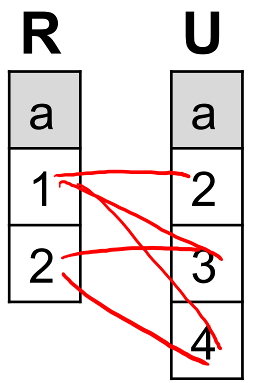
https://www.khoury.northeastern.edu/the-story-of-queryvis-not-just-another-visual-programming-language/

Outline: SQL (a refresher)

- SQL
  - Schema and keys
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Outer joins
  - Top-k

Theta joins

305



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

305

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

305

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	b
2	2

×

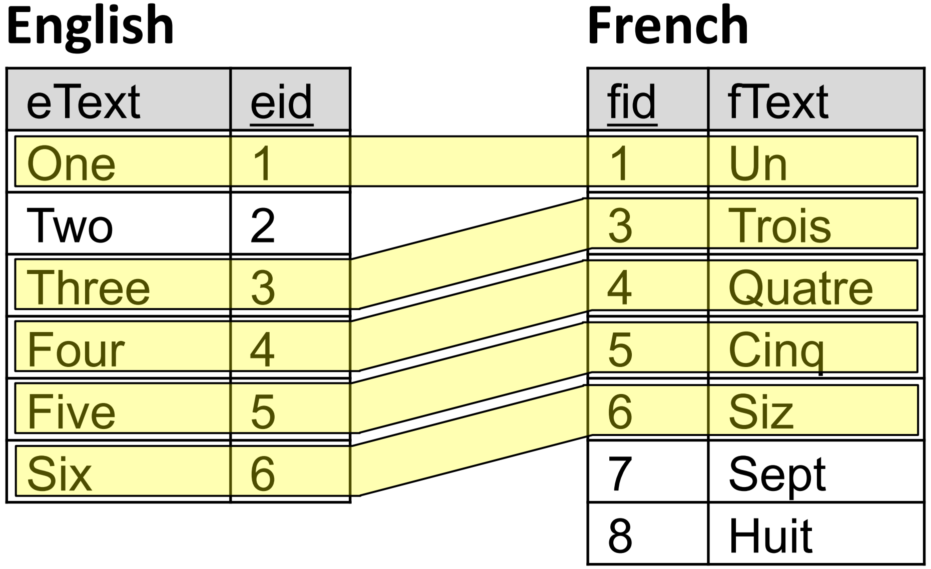
Think about these two queries as a partition of the Cartesian product

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

Outline: SQL (a refresher)

- SQL
  - Schema and keys
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Outer joins
  - Top-k

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
```

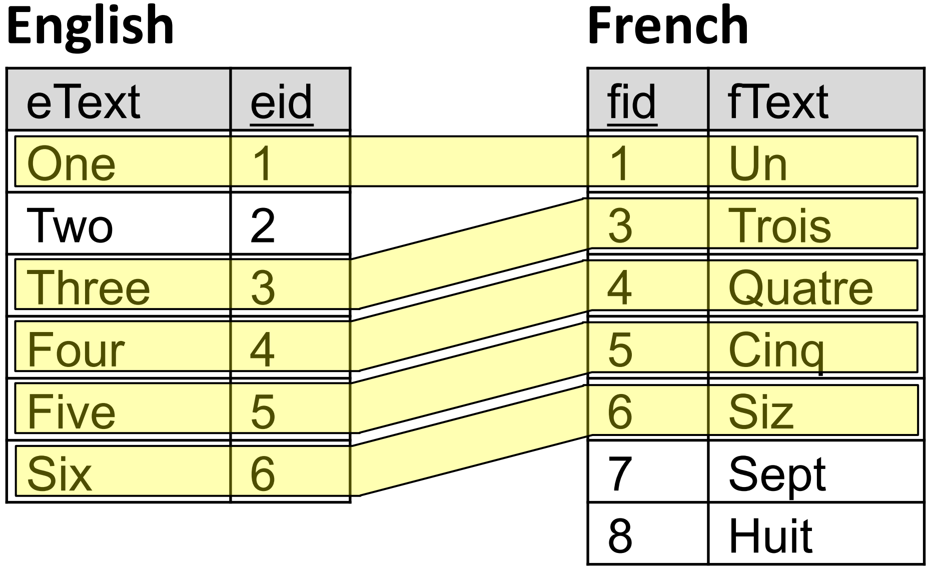
Same as:

```
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"

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

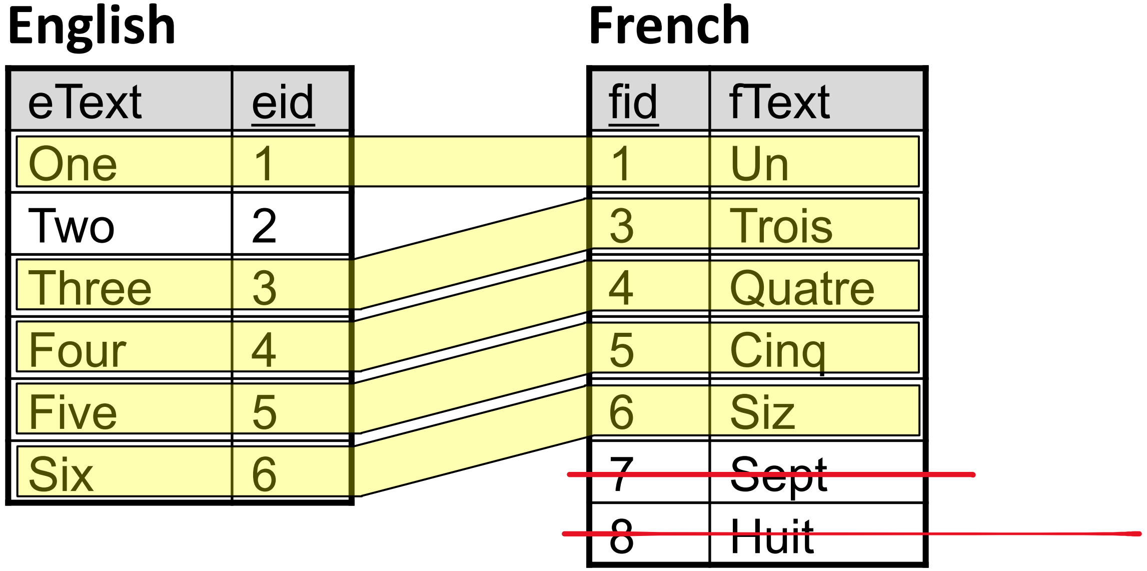
"FULL JOIN"
same as
"FULL OUTER JOIN"

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

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

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

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

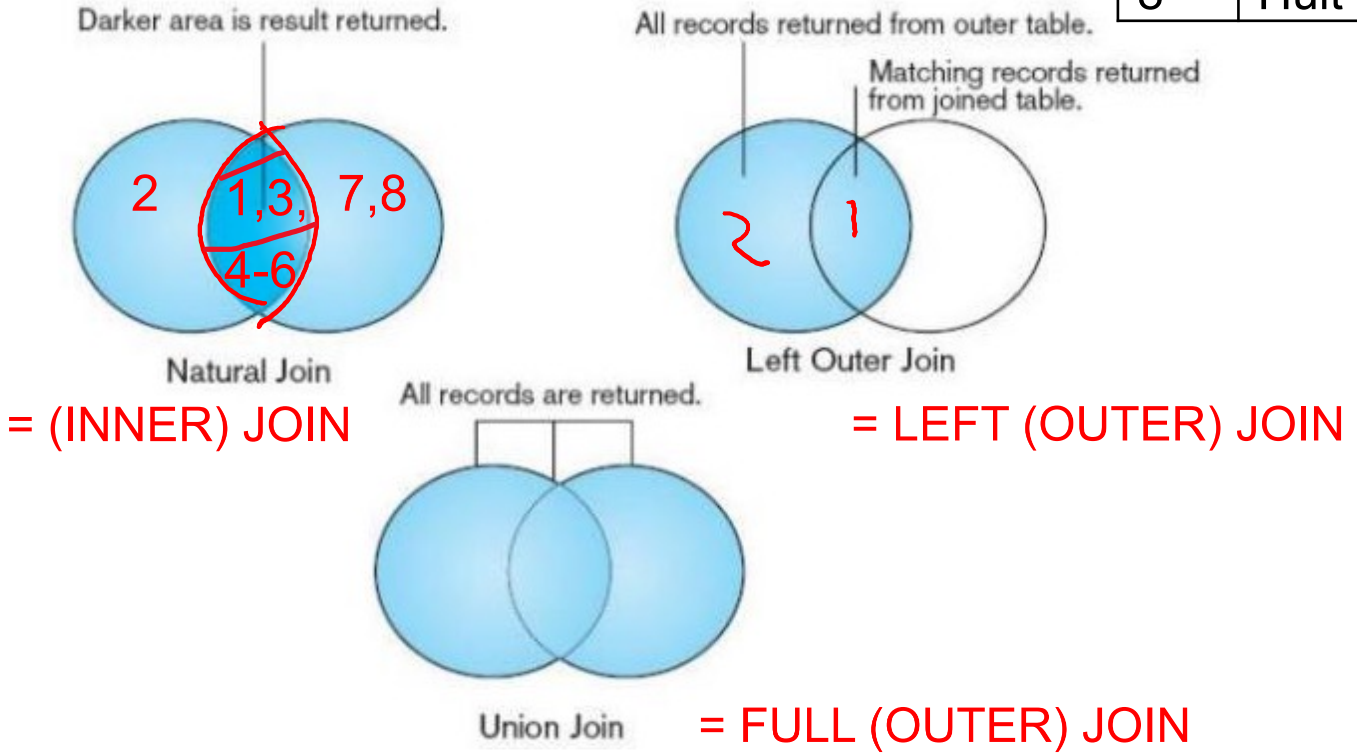
Illustration

English

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

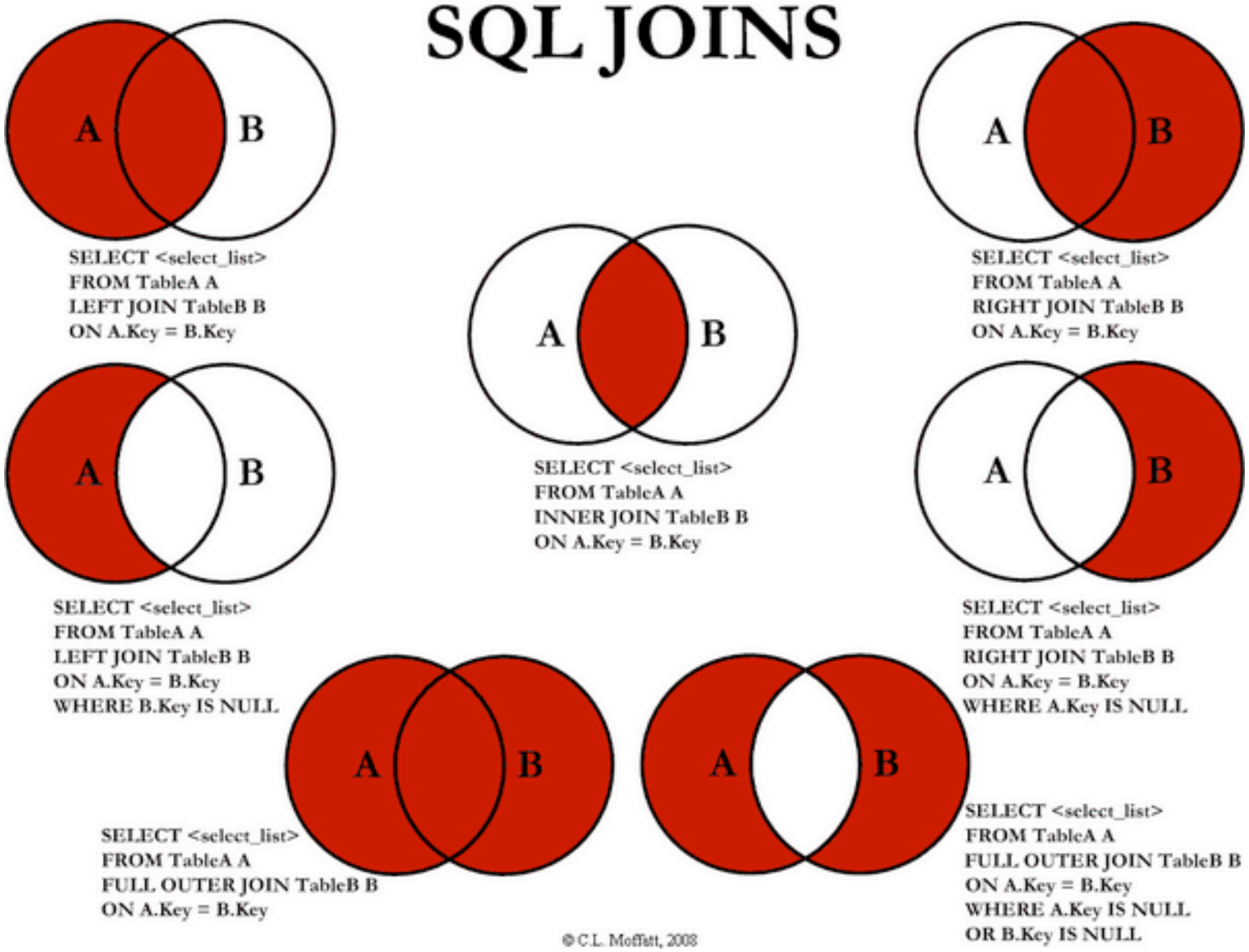
French

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)



SQL JOINS

```
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
```

also called "anti-join"

```
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
```

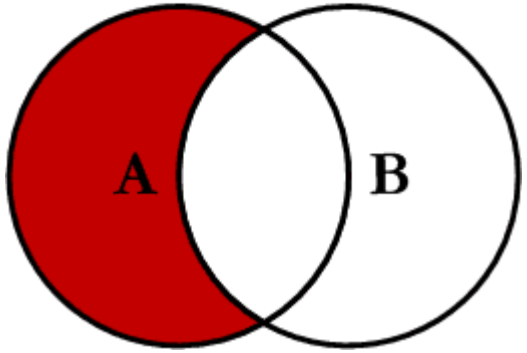
© C.L. Moffatt, 2008

Check this web page for illustrating examples

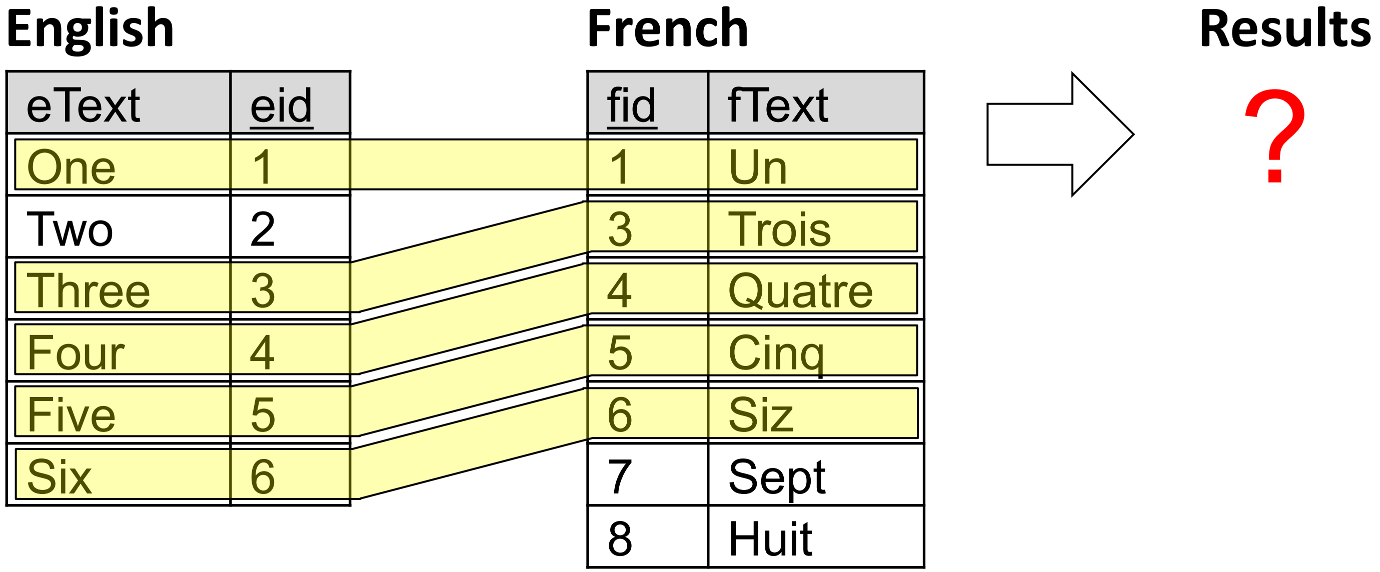
Source: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

Let's practice anti-joins

361



```
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

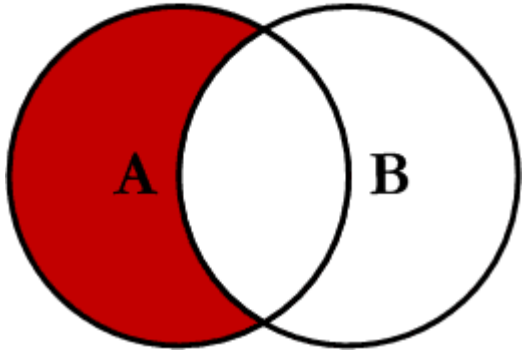
Results

?

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

Let's practice anti-joins

361



```
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

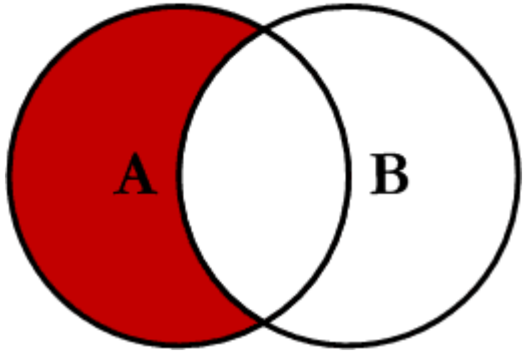
How to write in SQL?

?

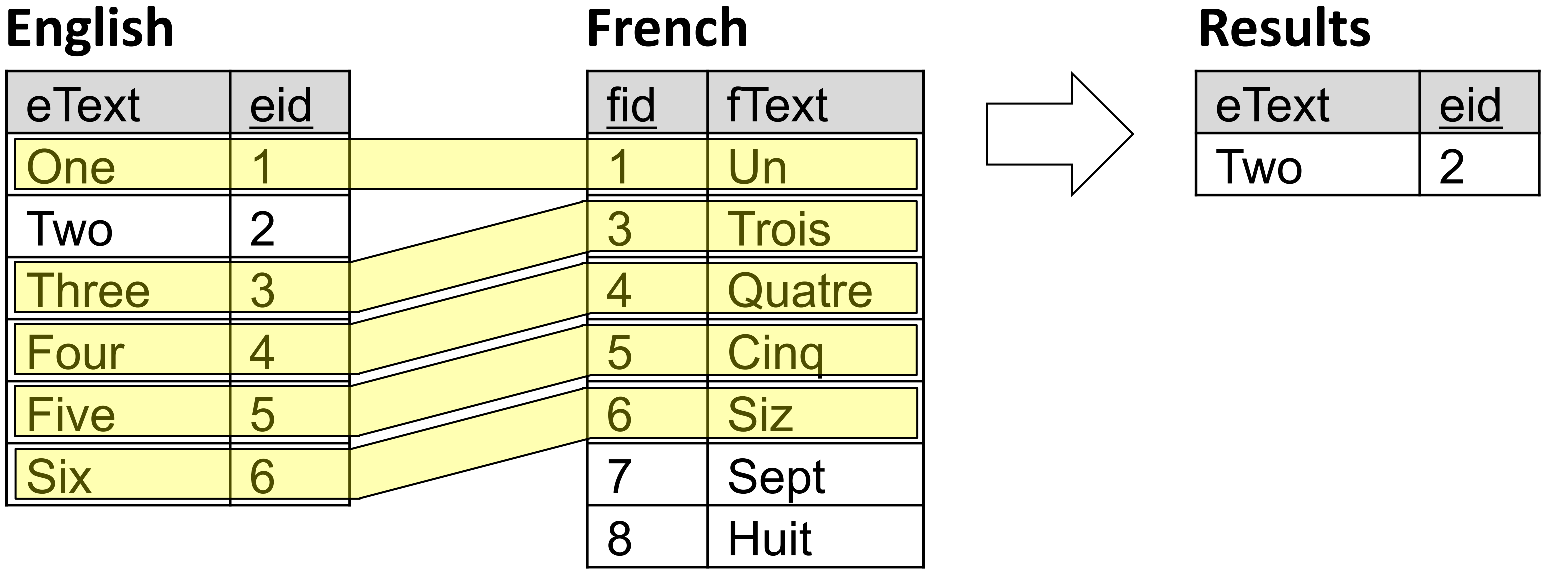
Source: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

Let's practice anti-joins

361



```
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
```

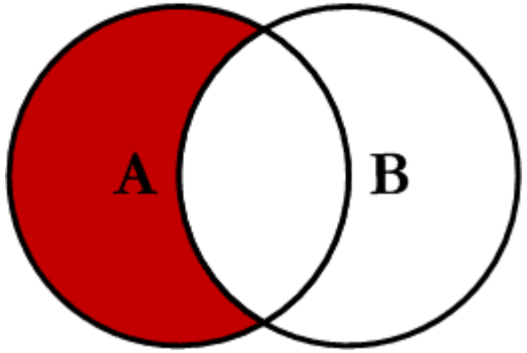
Any alternative?

?

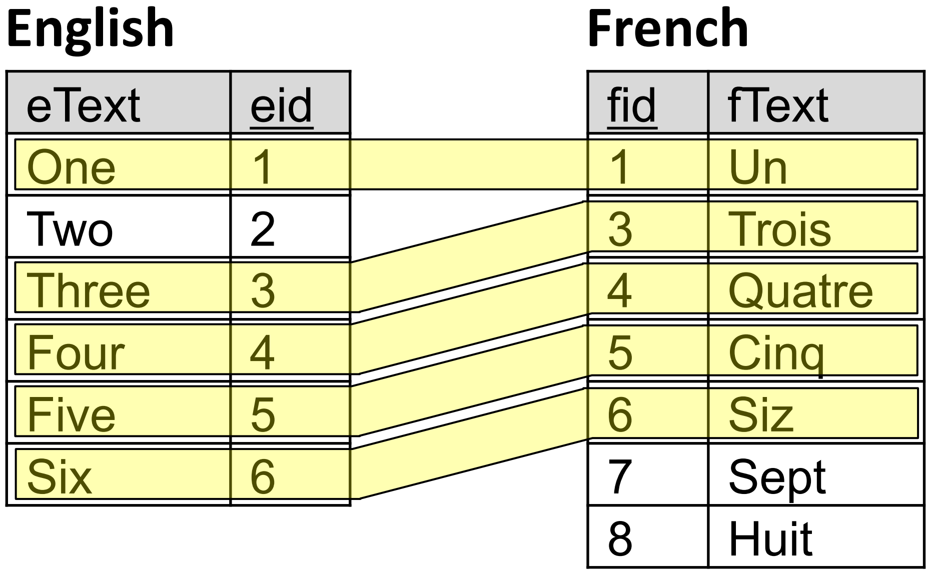
Source: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

Let's practice anti-joins

361



```
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
```

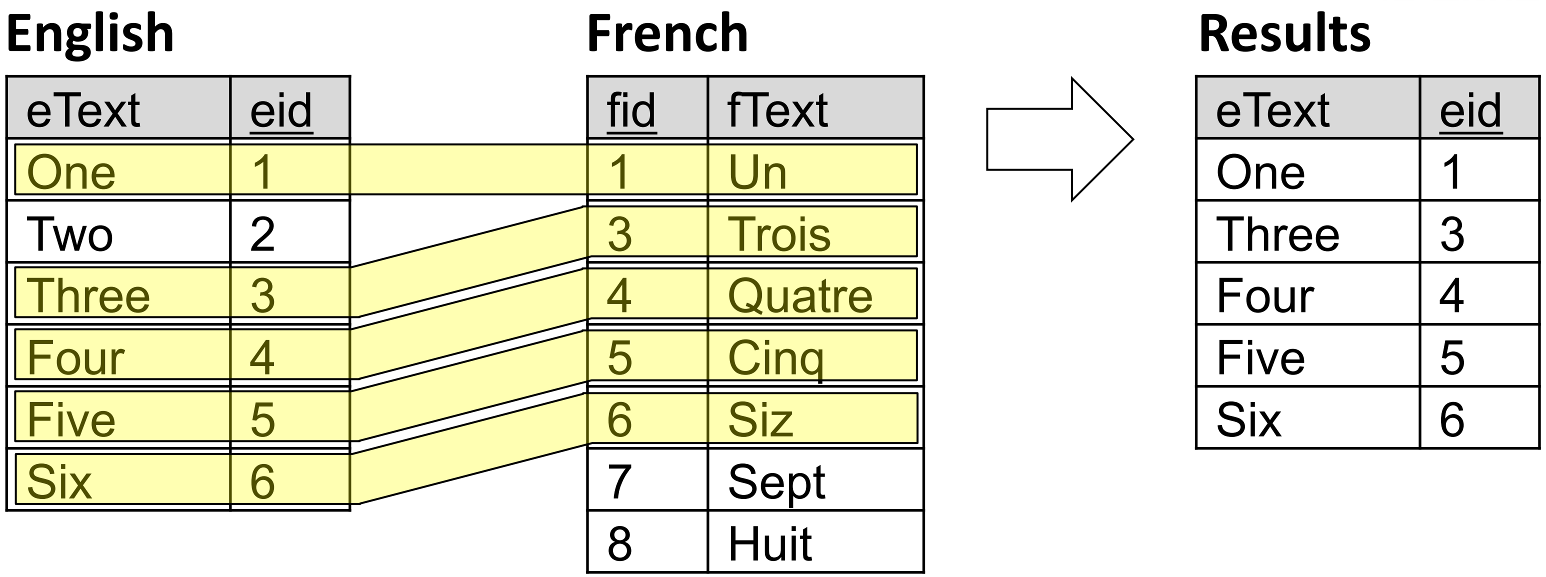
Any alternative?

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

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

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

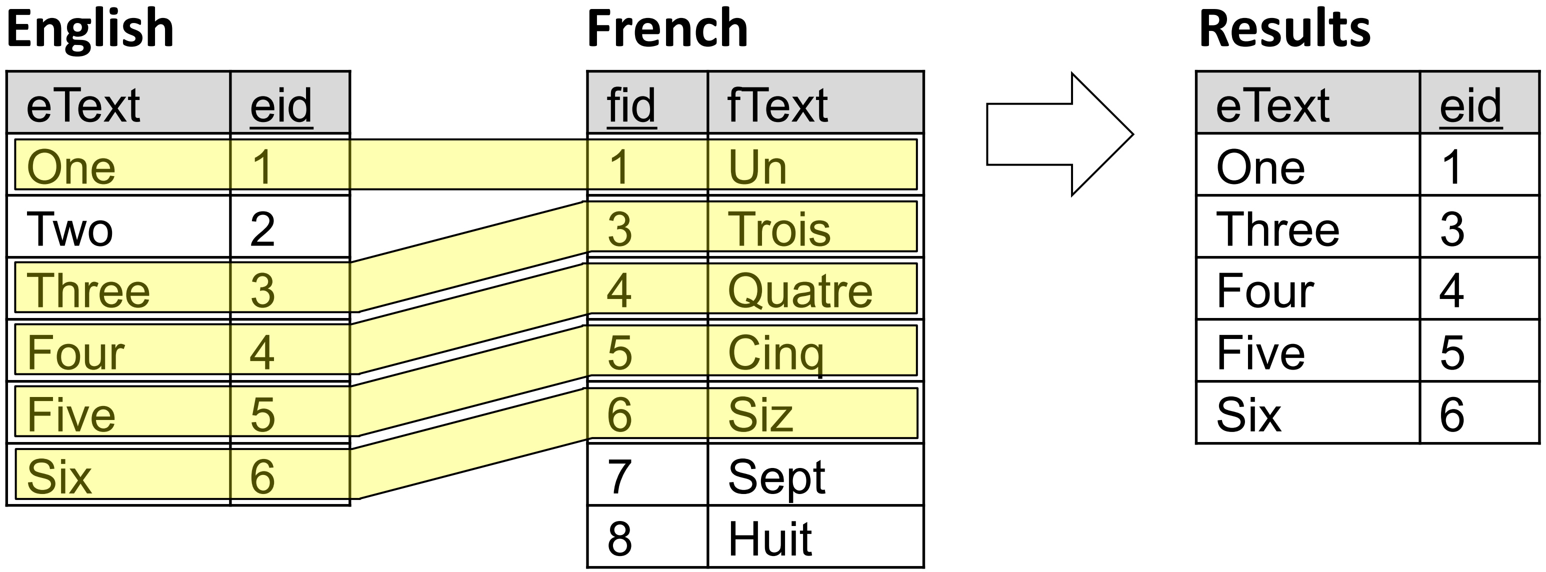
What do we have to change to these queries to get the tuples in English that have a partner in French?

?

```
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?

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

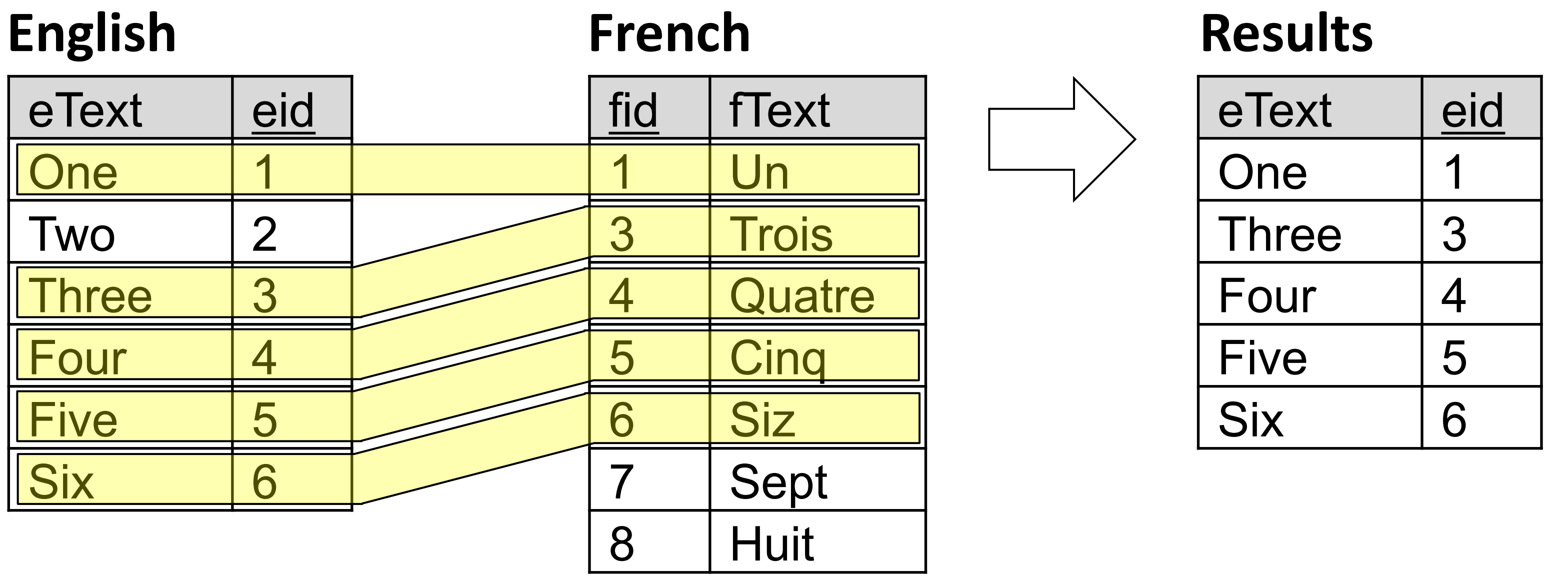
```
SELECT *
FROM English
WHERE eid ~~NOT~~ IN
    (SELECT fid
    FROM French)
```

What if fid is not a key?

?

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 ~~NOT~~ IN
    (SELECT fid
    FROM French)
```

Empty Group Problem

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

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

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

What is wrong?

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

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

⇨

Result

?

Empty Group Problem

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

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

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

We don't get the info for each product ☹

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	1

Empty Group Problem

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

334

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

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

How do you need to change the query to get what we want?

?

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

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

Gizmo

Result

Name	Store
Camera	1
~~Camera~~	0
OneClick	0

Empty Group Problem

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

334

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

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

Will this query work ?

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	1
Camera	0
OneClick	0

Empty Group Problem

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

334

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

*

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

No ☹ Still same result

Name	Category	iName	Store	Month
Gizmo	Gadget	Gizmo	Wiz	8
Camera	Photo	Camera	Ritz	8
Camera	Photo	Camera	Wiz	9
OneClick	Photo	NULL	NULL	NULL

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	1

Empty Group Problem

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

334

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

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

Now it works ☺

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	1
Camera	0
OneClick	0

Empty Group Problem

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

334

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

[illegible]

table alias

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

Previous page is a short form of this query here ☺

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	1
Camera	0
OneClick	0