Updated 1/18/2023

Topic 1: Data models and query languages
Unit 1: SQL (continued)
Lecture 3

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp23)

https://northeastern-datalab.github.io/cs7240/sp23/

1/17/2023

Pre-class conversations

- Last class summary
- New class members
- Intended extended focus on query languages
- First scribe arrived, I will comment Friday
  - Secondary posting of class scribes to Piazza (optionally anonymous). I will comment on both Canvas and Piazza

- Today:
  - SQL continued

HOME CALENDAR PROJECT

PRELIMINARY

CS 7240: Topics and approximate agenda (Spring'23)

This schedule will be updated regularly as the class progresses. Check back frequently. I will usually post lecture slides by the end of the day following a lecture (thus the next day). I post them here on this website (or in Canvas if I think they are not yet ready to be released in public). Please also check our DATA lab seminar for talks of interest.

Topic 1: Data Models and Query Languages

- **Lecture 1 (Tue 1/10):** Course introduction / T1-U1 SQL / PostgreSQL setup / SQL Activities
- **Lecture 2 (Fri 1/13):** T1-U1 SQL
- **Lecture 3 (Tue 1/17):** T1-U1 SQL
- **Lecture 4 (Fri 1/20):** T1-U2 Logic & Relational Calculus
- **Lecture 5 (Tue 1/24):** T1-U1 Logic & Relational Calculus
- **Lecture 6 (Fri 1/27):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 7 (Tue 1/31):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 8 (Fri 2/3):** T1-U4 Datalog & Recursion
- **Lecture 9 (Tue 2/7):** T1-U4 Datalog & Recursion
- **Lecture 10 (Tue 2/10):** T1-U4 Datalog & Recursion

Topic 2: Complexity of Query Evaluation & Reverse Data Management

- **Lecture 11 (Tue 2/14):** T2-U1 Conjunctive Queries
- **Lecture 12 (Fri 2/17):** T2-U1 Conjunctive Queries
- **Lecture 13 (Tue 2/21):** T2-U2 Beyond Conjunctive Queries
- **Lecture 14 (Fri 2/24):** T2-U3 Provenance
- **Lecture 15 (Tue 2/28):** T2-U3 Provenance
- **Lecture 16 (Fri 3/3):** T2-U4 Reverse Data Management

Topic 3: Efficient Query Evaluation & Factorized Representations

- Spring break (Tue 3/7, Fri 3/10: Northeast Database day 2023 @ Northeastern)
- **Lecture 17 (Tue 3/14):** T3-U1 Acyclic Queries
- **Lecture 18 (Fri 3/17):** T3-U1 Acyclic Queries
- **Lecture 19 (Tue 3/21):** T3-U2 Cyclic Queries
- **Lecture 20 (Fri 3/24):** T3-U2 Cyclic Queries
- **Lecture 21 (Tue 3/28):** T3-U2 Cyclic Queries
- **Lecture 22 (Fri 3/31):** T3-U2 Cyclic Queries
- **Lecture 23 (Tue 4/4):** T3-U3 Factorized Representations
- **Lecture 24 (Fri 4/7):** T3-U4 Optimization Problems & Top-k
- **Lecture 25 (Tue 4/11):** T3-U4 Optimization Problems & Top-k

Topic 4: Normalization, Information Theory & Axioms for Uncertainty

- **Lecture:** Normal Forms & Information Theory
- **Lecture:** Axioms for Uncertainty

Topic 5: Linear Algebra & Iterative Graph Algorithms

- **Lecture:** Graphs & Linear Algebra
- **Lecture:** Computation Graphs

Project presentations

- **Lecture 26 (Fri 4/14): P4 Project presentations**
- **Lecture 27 (Tue 4/18): P4 Project presentations**

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

A natural question

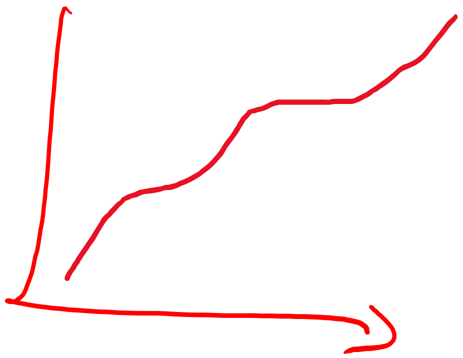
Q_2: Find all companies that make only products with price < 25

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

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

?

Queries that must be nested



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

Understanding nested queries with Relational Diagrams and QueryVis

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 *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

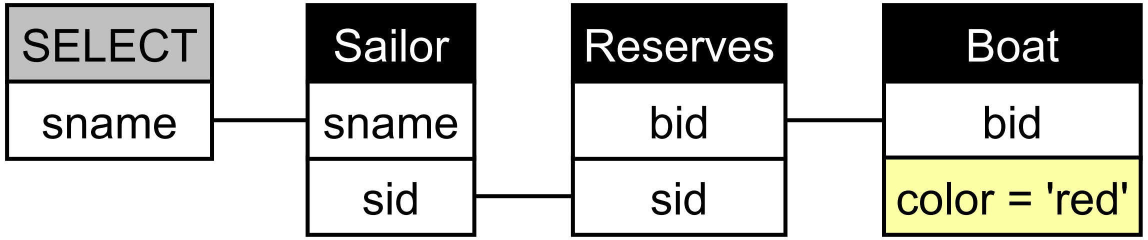
Schema and several of the following queries taken from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

Nested query 1

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

Q: ?

```sql
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'))
```



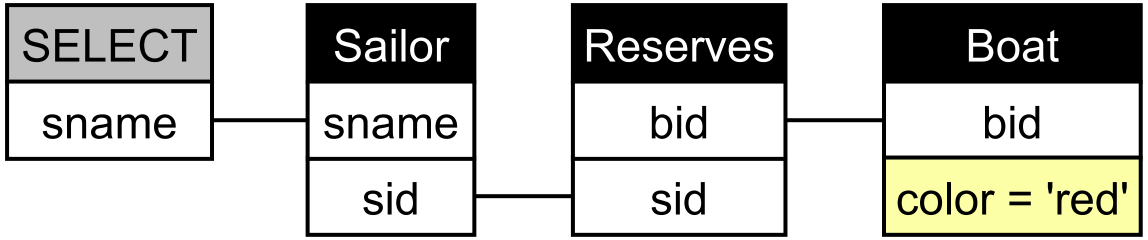
Nested query 1

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')))}

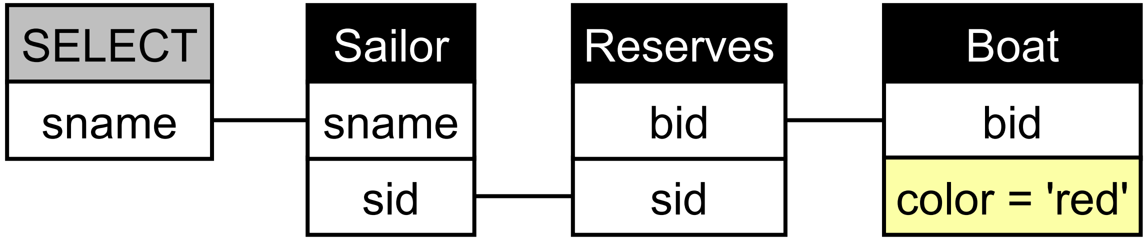
Nested query 1

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

340

Q: Find 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.bid = R.bid
    AND B.color = 'red'))
```



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

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

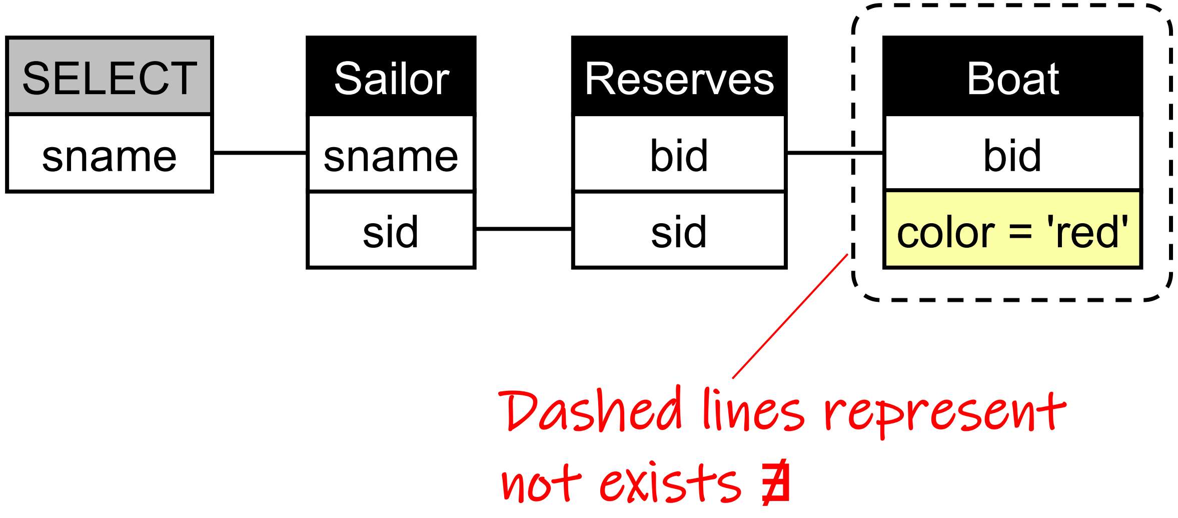
Nested query 2

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

340

Q: ?

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



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

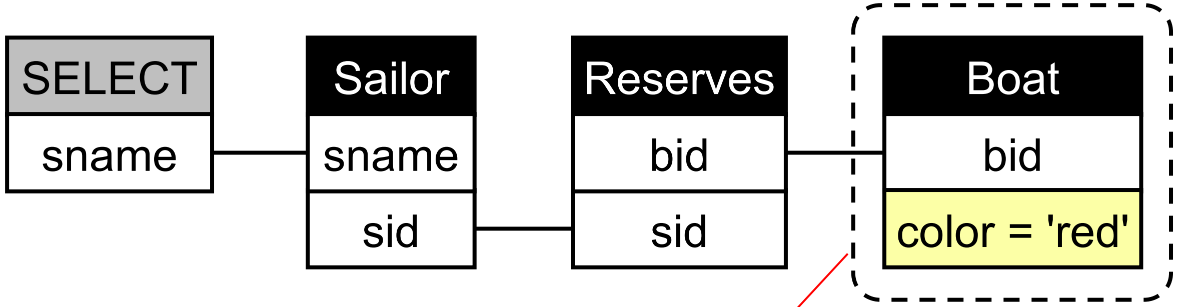
Nested query 2

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

340

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

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

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

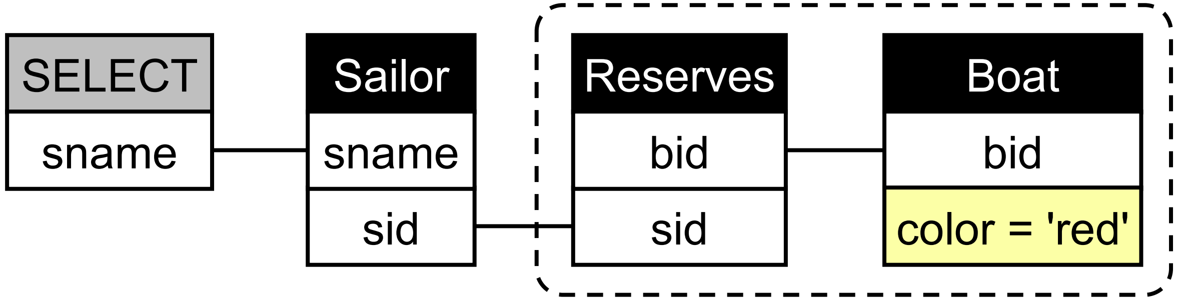
Nested query 3

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

340

Q: ?

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



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

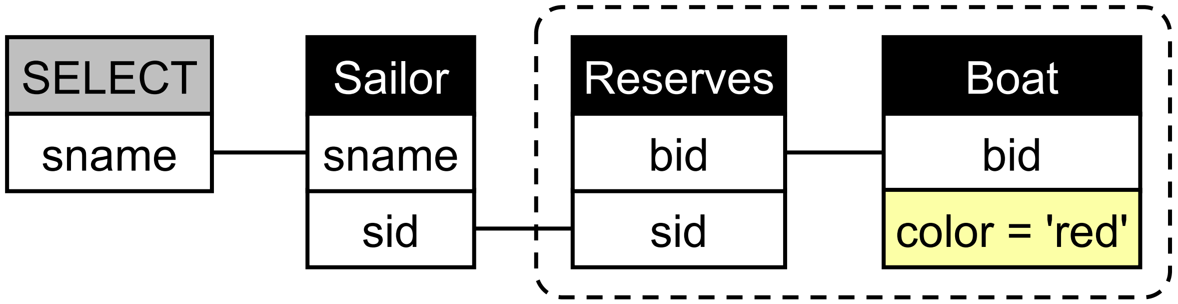
Nested query 3

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

340

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

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



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

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

Quiz: Dustin?

340

Sailor

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

Figure 5.1 An Instance *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 sailors who have reserved a boat that is not red.

Q3: Find 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 sailors who have reserved a boat that is not red. Yes!

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

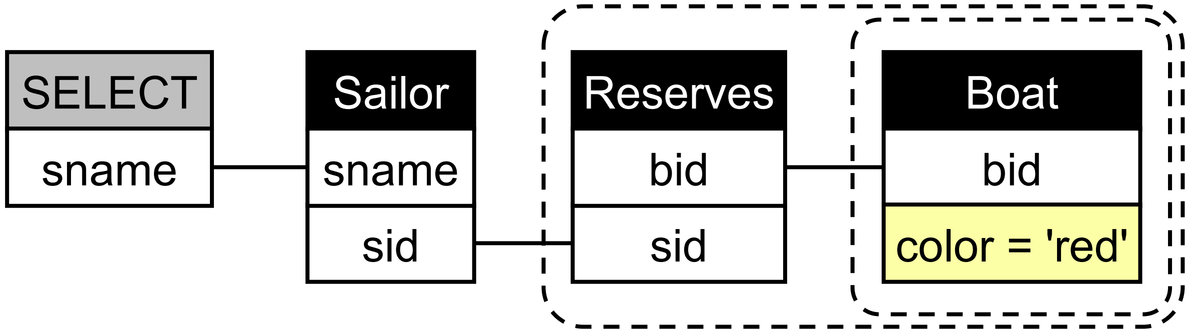
Nested query 4

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

340

Q: ?

```sql
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
  (SELECT R.sid
  FROM Reserves R
  WHERE R.sid=S.sid
  AND NOT EXISTS
    (SELECT B.bid
    FROM Boat B
    WHERE B.bid = R.bid
    AND 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

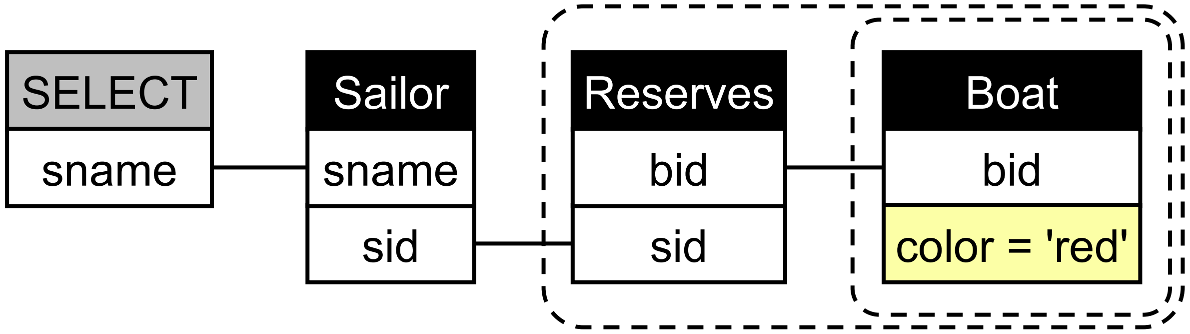
They can have reserved 0 or more boats in red, just no other color

= Find sailors who have reserved only red boats

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

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


```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
  (SELECT R.sid
   FROM Reserves R
   WHERE R.sid=S.sid
   AND NOT EXISTS
     (SELECT B.bid
      FROM Boat B
      WHERE B.bid = R.bid
      AND 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 4 (universal)

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

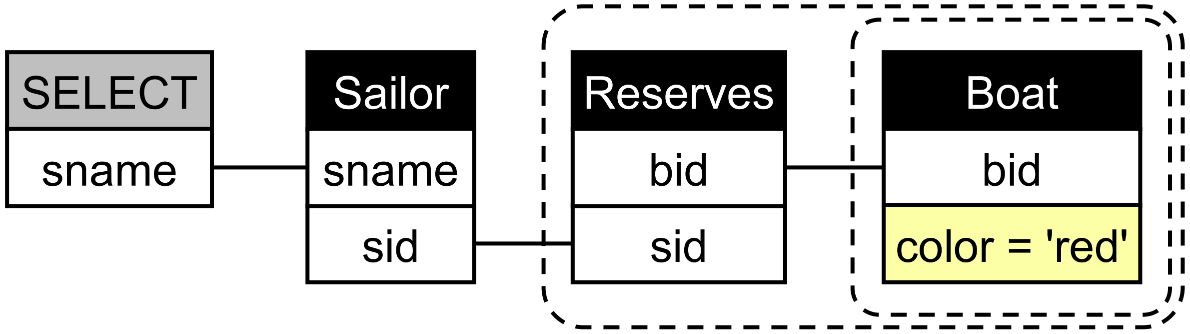
340

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

= Find sailors who have reserved only red boats

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

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

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

Nested query 4 (another variant)

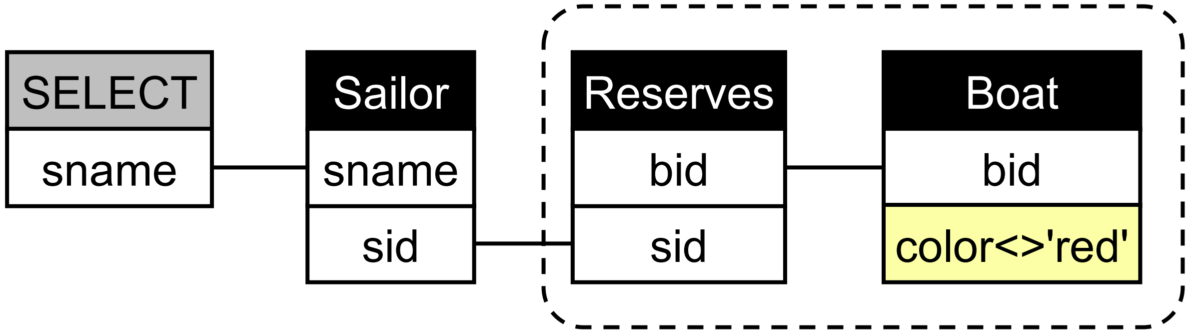
340

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

= Find sailors who have reserved only red boats

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

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



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

Equivalence with previous variant only because of FK-PK constraint!

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

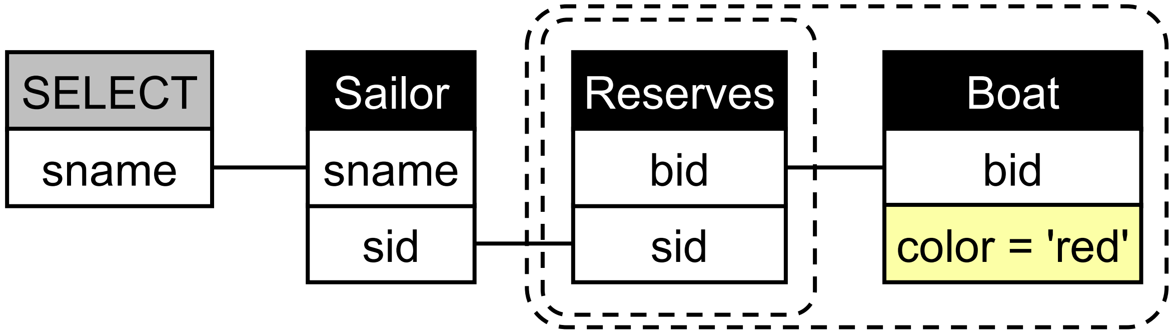
Nested query 5

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

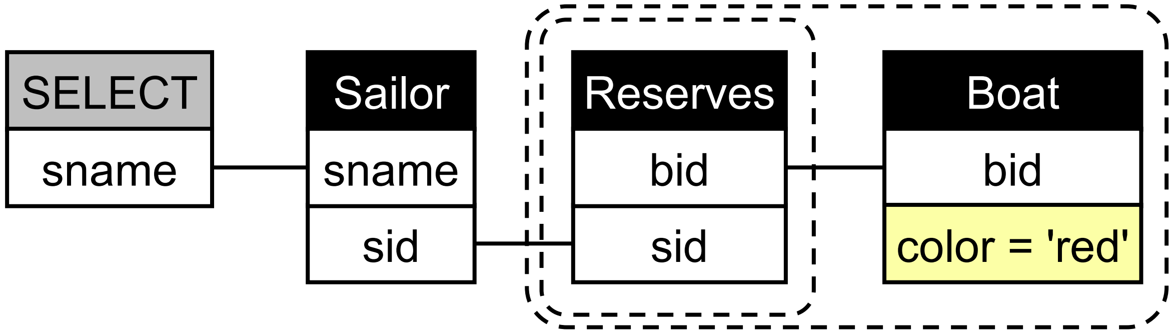
Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

340

= Find sailors who have reserved all red boats

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

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



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

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

Nested query 5 (universal)

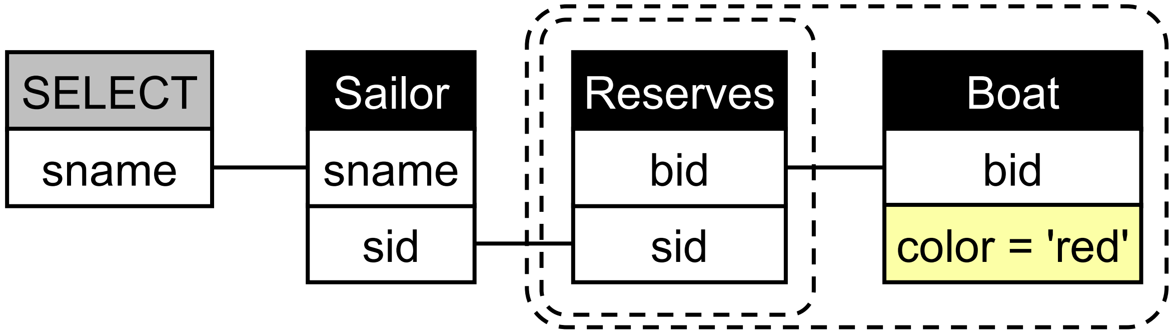
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find sailors who have reserved all red boats

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

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



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

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

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

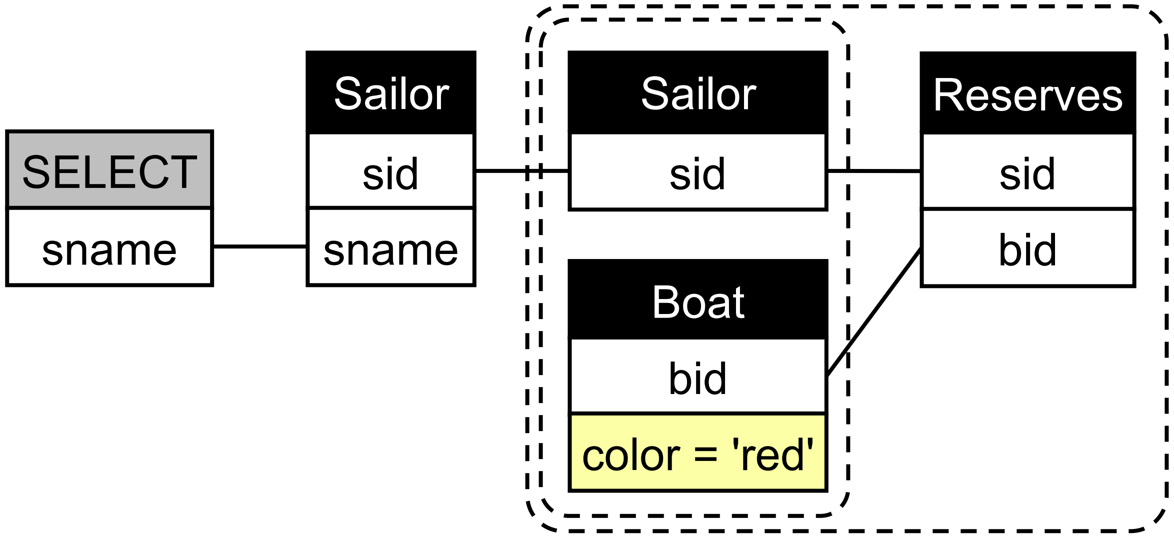
Nested query 5 (w/o correlation)

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

= Find sailors who have reserved all red boats

Q: Find 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 ∧ R.sid=S2.sid))))}

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

Nested query 5 (w/o correlation)

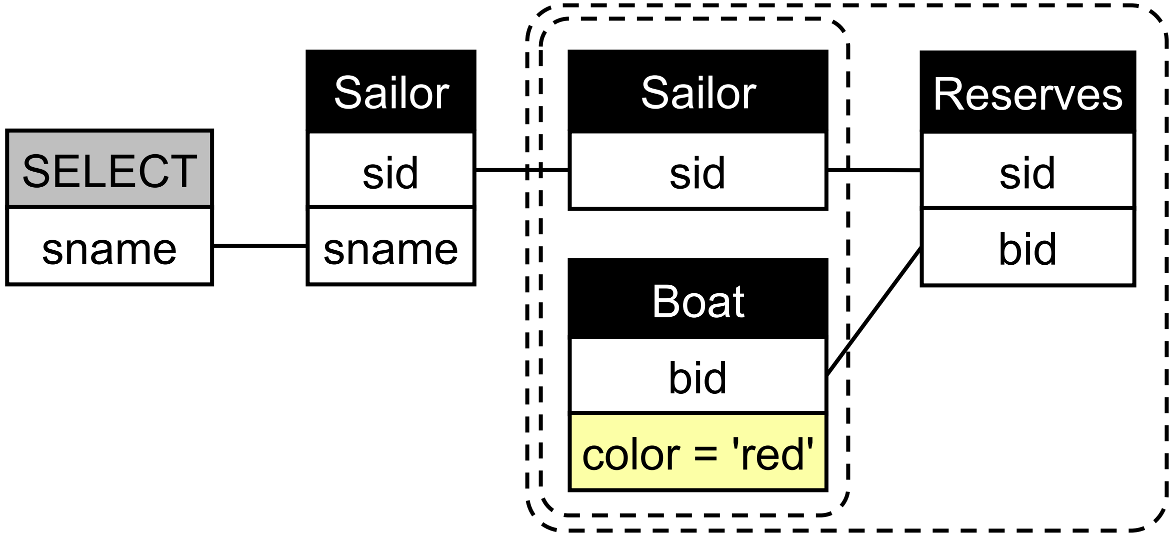
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find sailors who have reserved all red boats

Q: Find 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 ∧ R.sid=S2.sid))))}

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

Towards SQL patterns

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

	Sailors who **have not** reserved a red boat	Sailors who reserved **only** red boats	Sailors who reserved **all** red boats

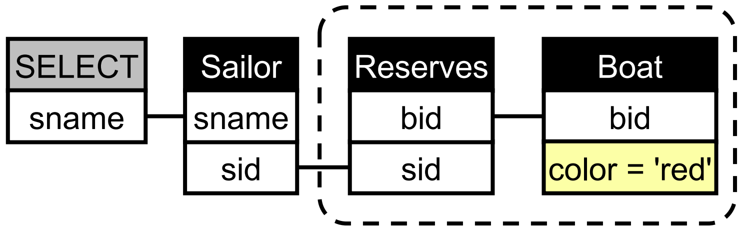
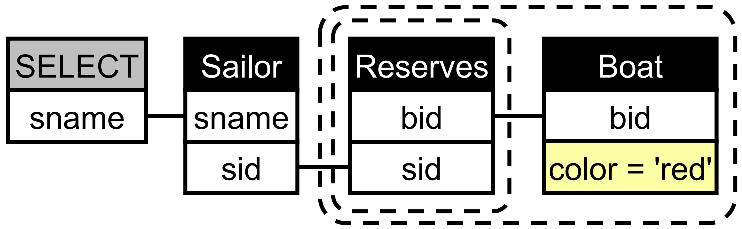
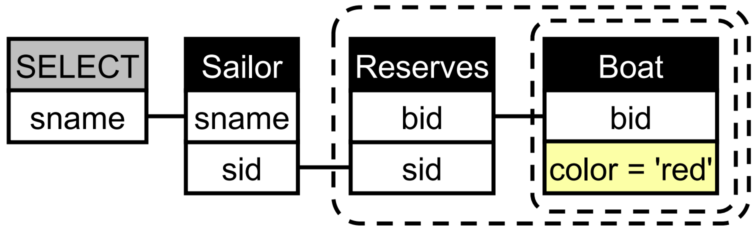
SQL	`SELECT DISTINCT S.sname`
	`FROM Sailor S`
	`WHERE NOT EXISTS(`
	`  SELECT *`
	`  FROM Reserves R, Boat B`
	`  WHERE R.sid = S.sid`
	`  AND R.bid = B.bid`
	`  AND B.color = 'red')`
		`SELECT DISTINCT S.sname`
		`FROM Sailor S`
		`WHERE NOT EXISTS(`
		`  SELECT *`
		`  FROM Reserves R`
		`  WHERE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

Towards SQL patterns

Sailor (<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 R.bid = B.bid
		    AND B.color = 'red'))
			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))

RD			

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

	not	only	all
Sailors renting boats	have not reserved a red boat	reserved only red boats	reserved all red boats

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

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

	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

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)

Sailors

not

```
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')
```

only

```
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 R.bid = B.bid
    AND B.color = 'red'))
```

all

```
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

not

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

only

```
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 T.cid = C.cid
    AND C.department= 'art'))
```

all

```
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

not

```
SELECT DISTINCT A.aname
FROM Actor A
WHERE NOT EXISTS(
  SELECT *
  FROM Plays P, Movie M
  WHERE P.aid = A.aid
  AND P.mid = M.mid
  AND M.director= 'Hitchcock')
```

only

```
SELECT DISTINCT A.aname
FROM Actor A
WHERE NOT EXISTS(
  SELECT *
  FROM Plays P
  WHERE P.aid = A.sid
  AND NOT EXISTS(
    SELECT *
    FROM Movie M
    WHERE P.mid = M.mid
    AND M.director= 'Hitchcock'))
```

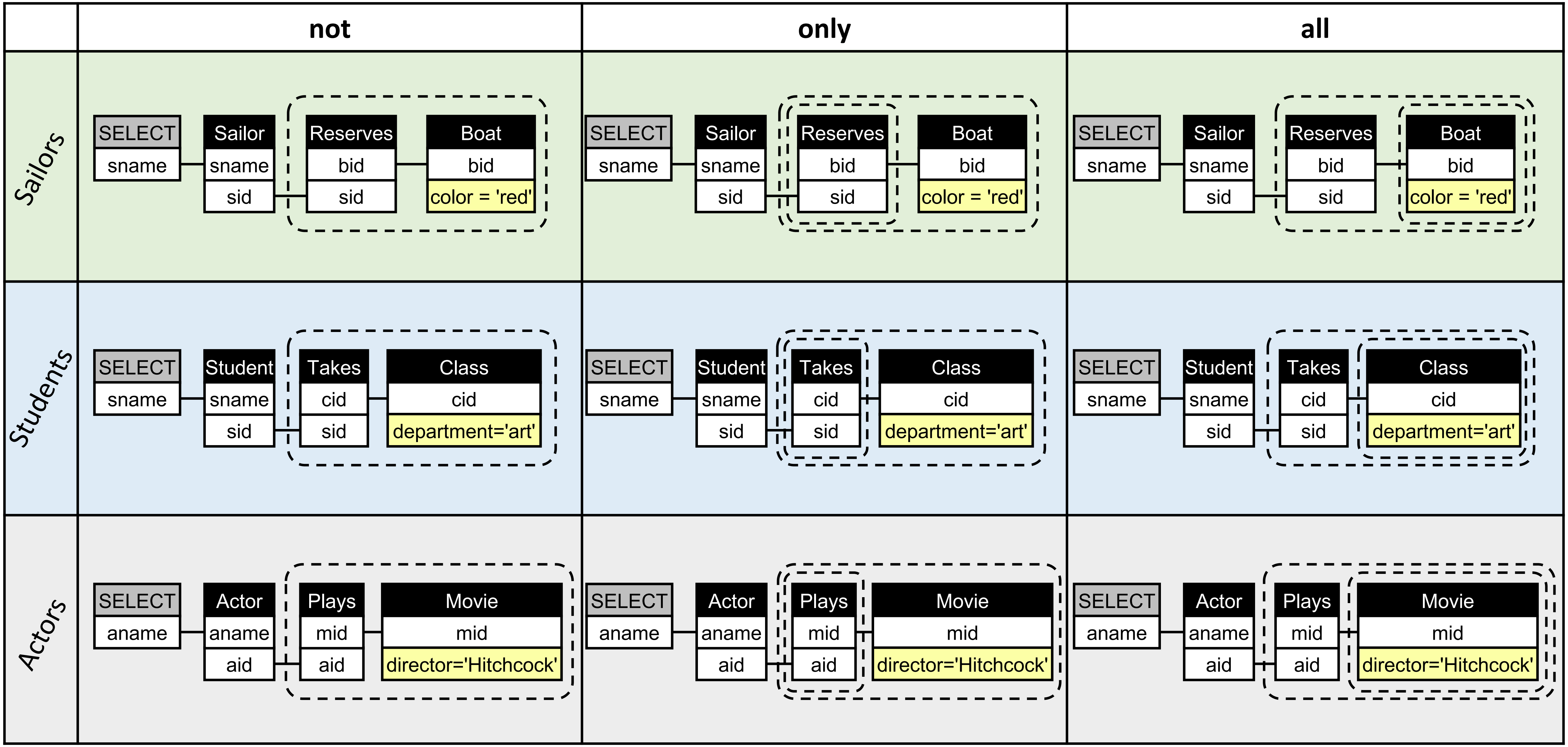
all

```
SELECT DISTINCT A.aname
FROM Actor A
WHERE NOT EXISTS(
  SELECT *
  FROM Movie M
  WHERE M.director= 'Hitchcock'
  AND NOT EXISTS(
    SELECT *
    FROM Plays P
    WHERE P.mid = M.mid
    AND P.aid = A.aid))
```

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

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

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



Logical SQL Patterns

Logical patterns are the building blocks of most SQL queries.

Patterns are very hard to extract from the SQL text.

A pattern can appear across different database schemas.

Think of queries like:

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

For an early formal definition of relational query patterns see: https://arxiv.org/pdf/2203.07284

What does this query return?

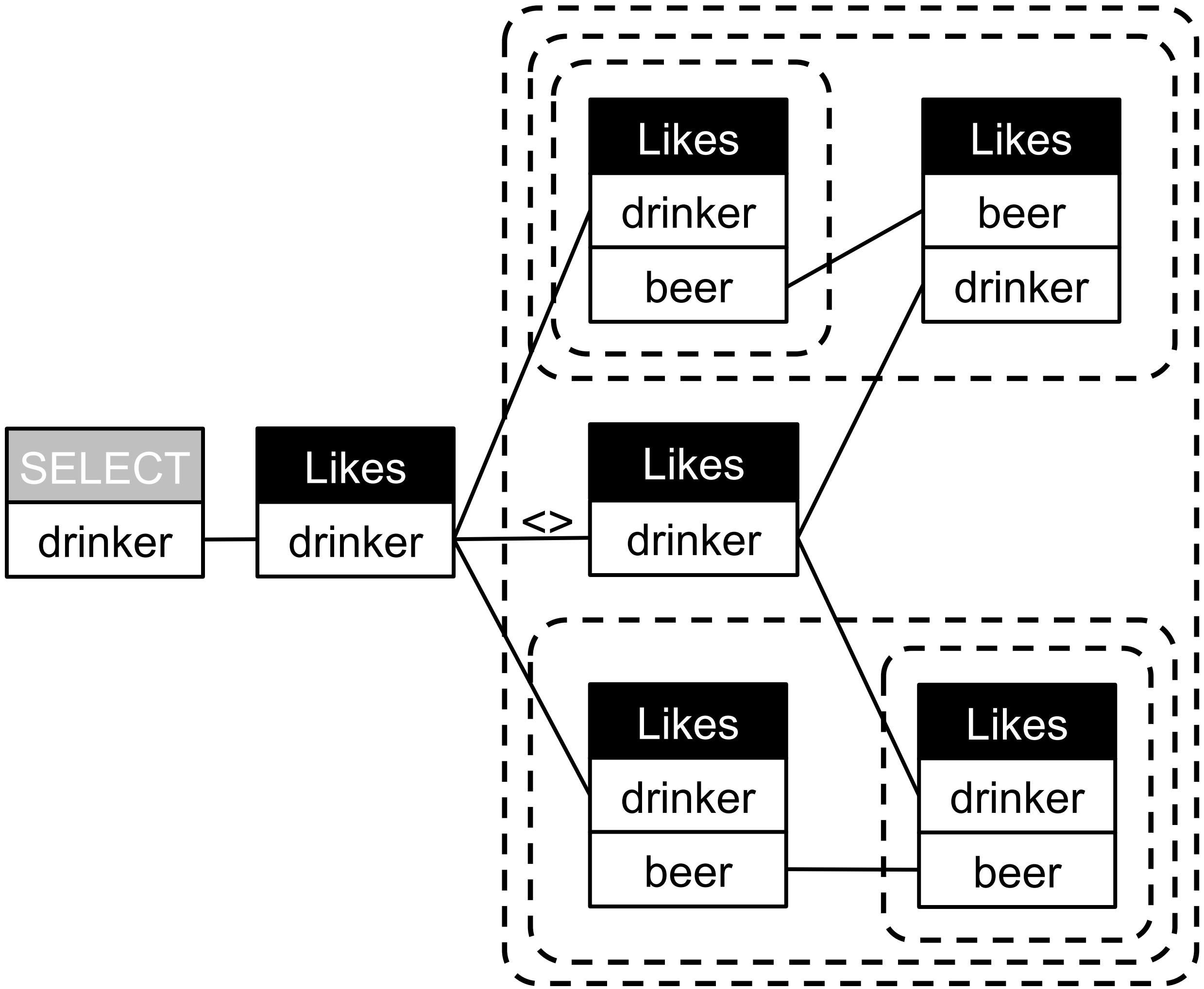
Likes(drinker,beer)

```
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)))
```

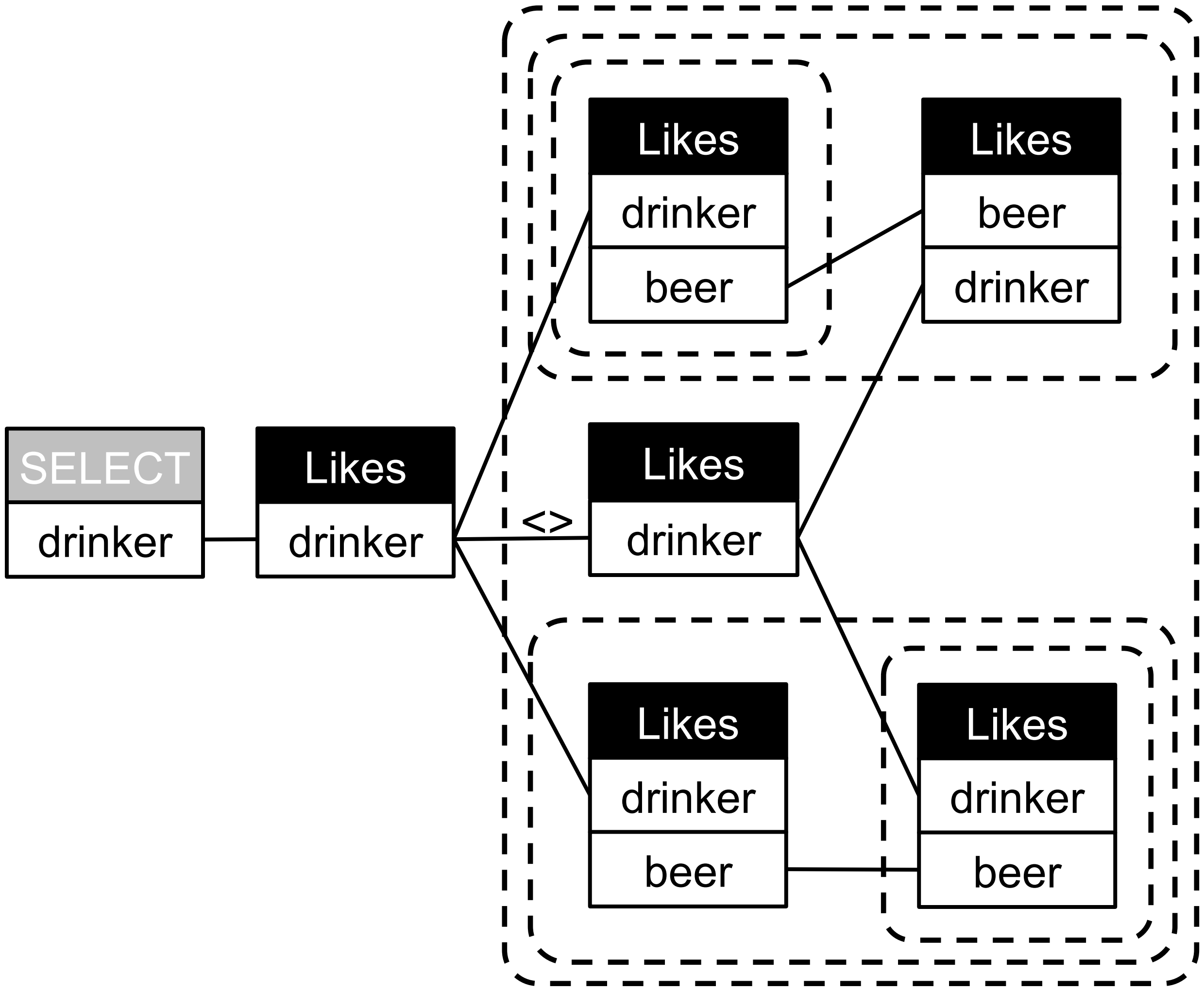


Relational Diagrams scoping

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)))
```

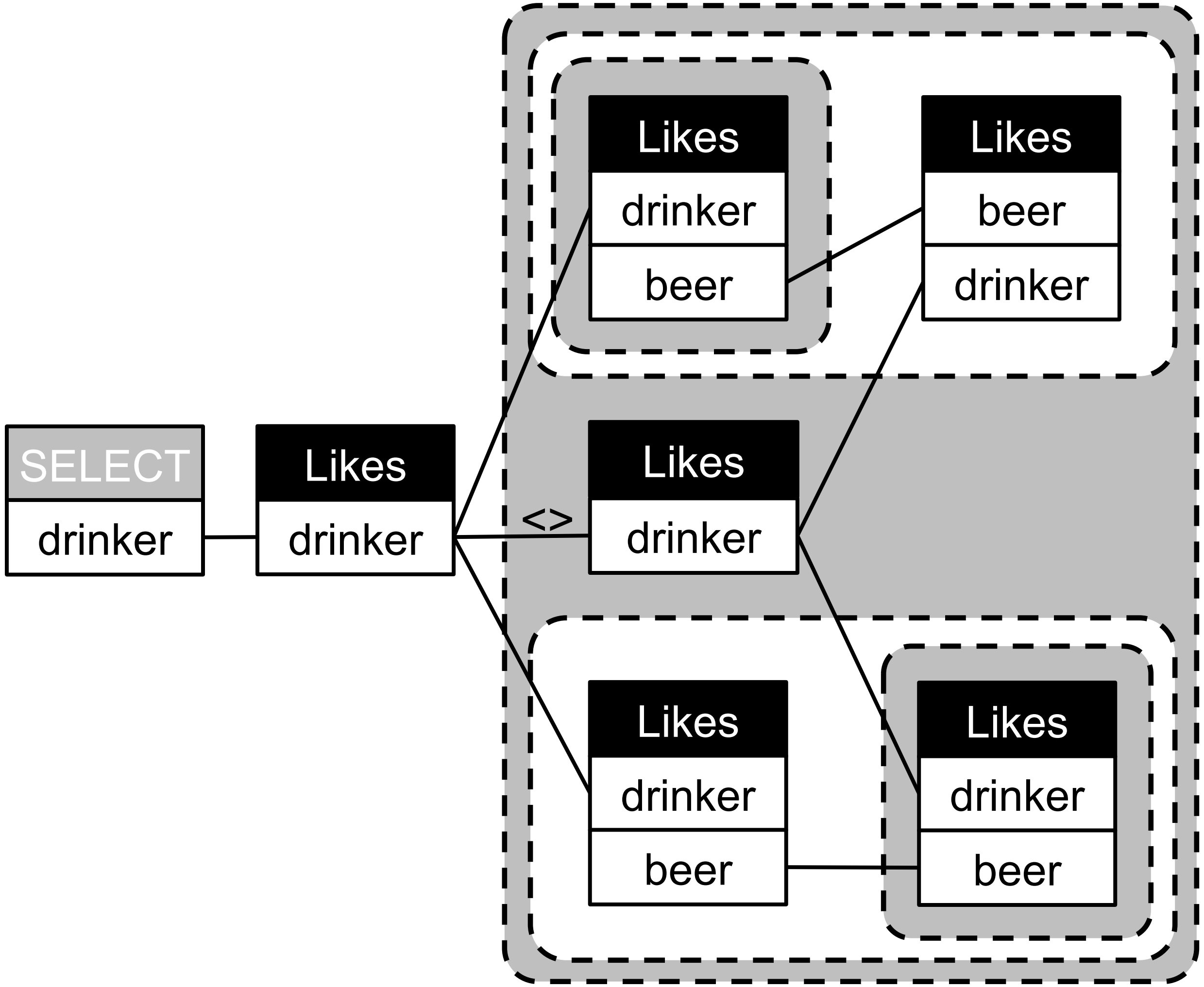


Relational Diagrams scoping

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)))
```

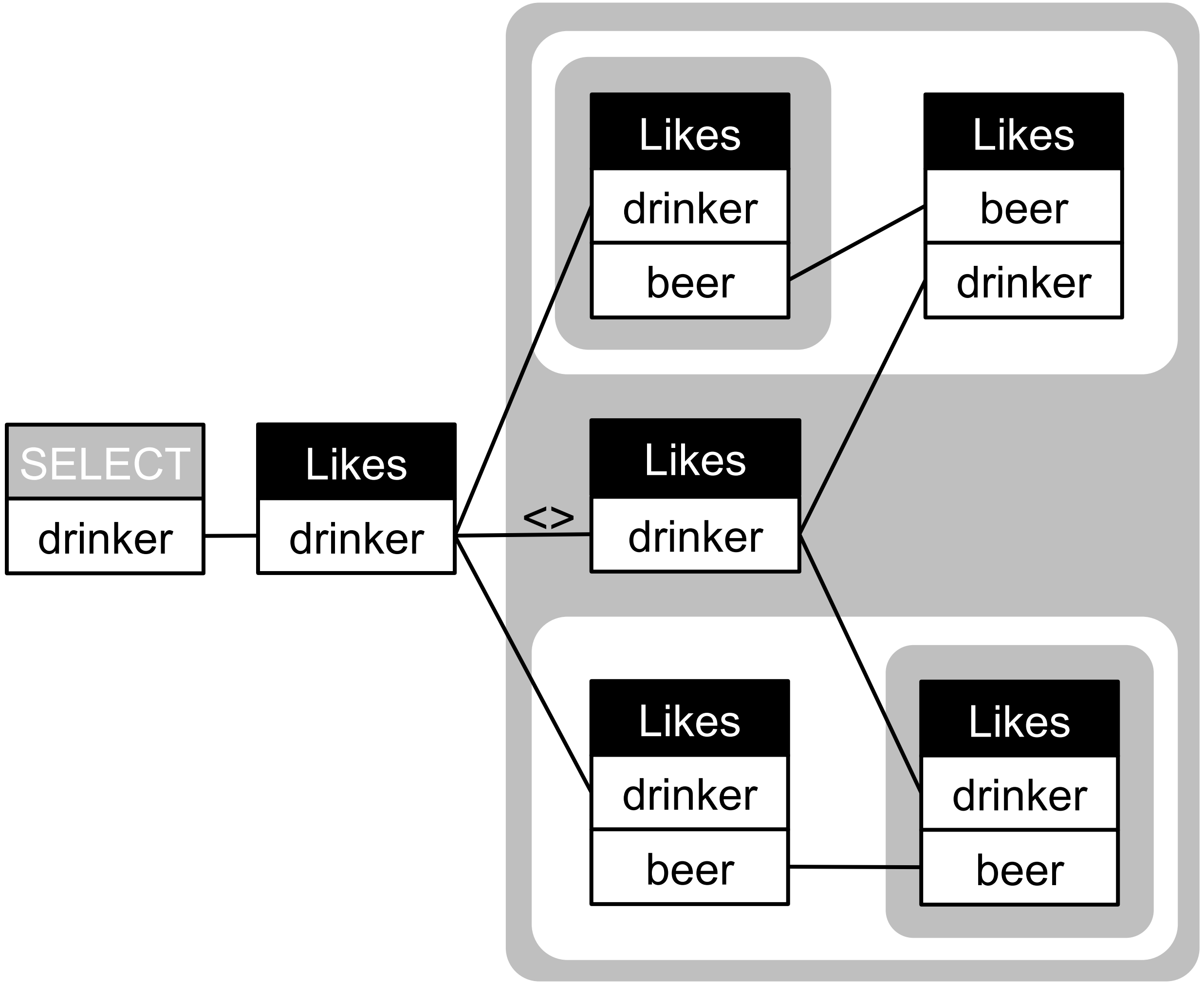


Relational Diagrams scoping

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)))
```

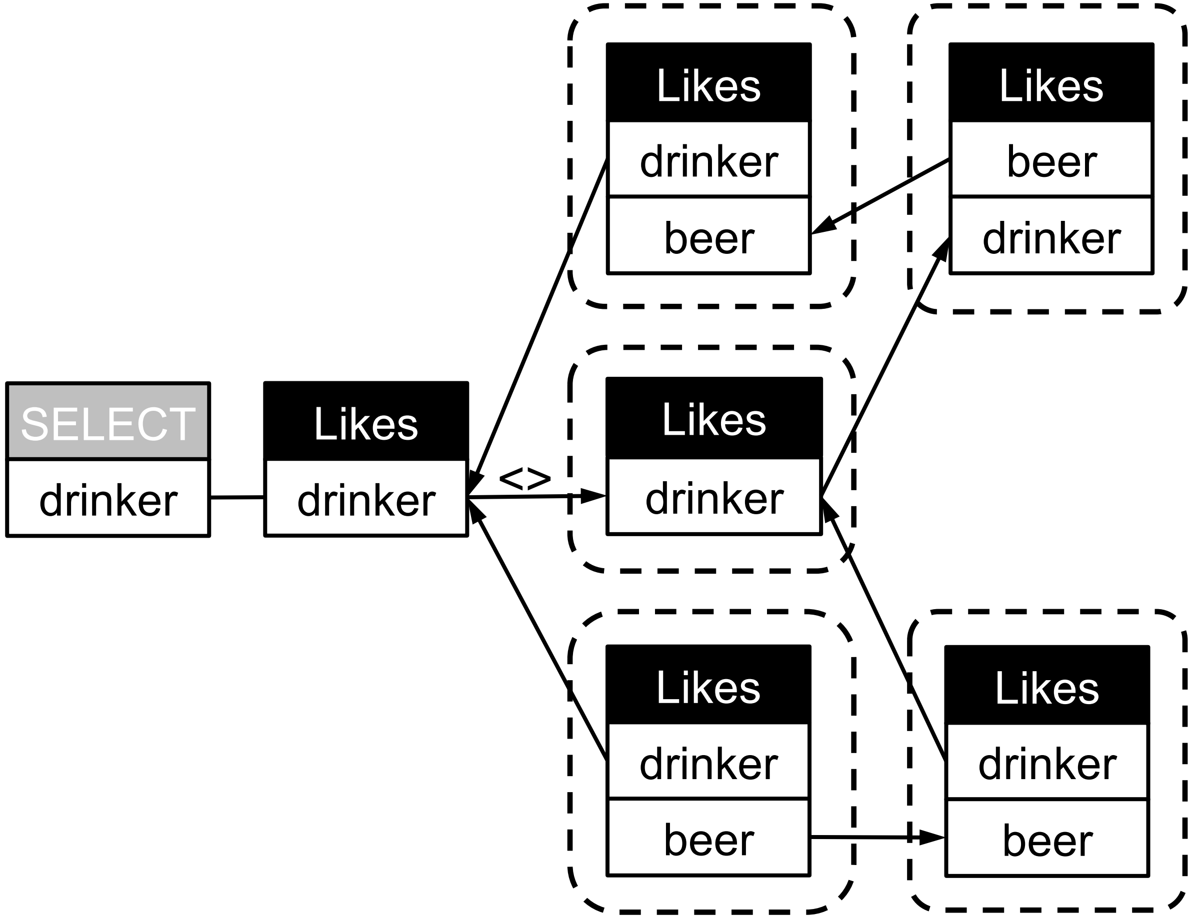


Relational Diagrams scoping

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)))
```

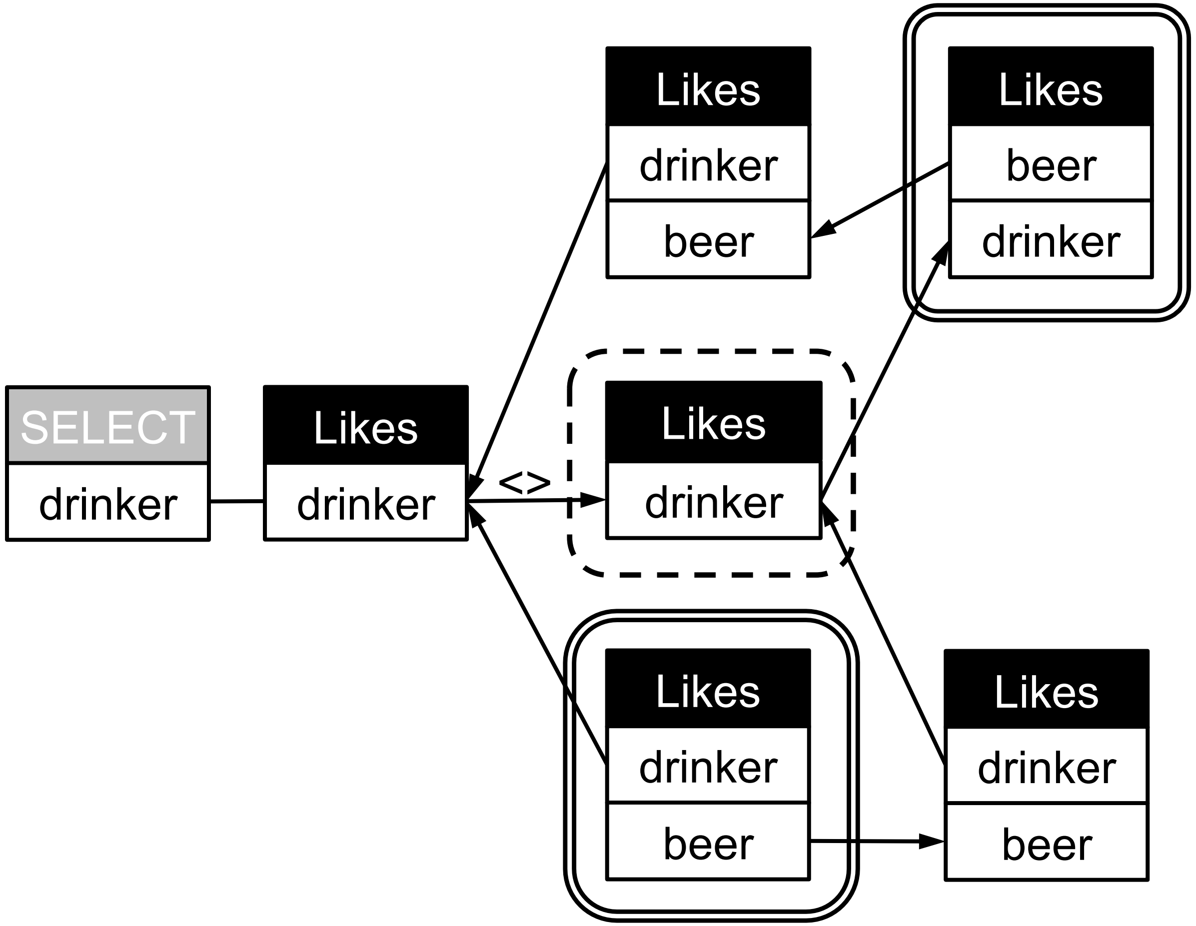


QueryVis scoping   Relational Diagrams scoping

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

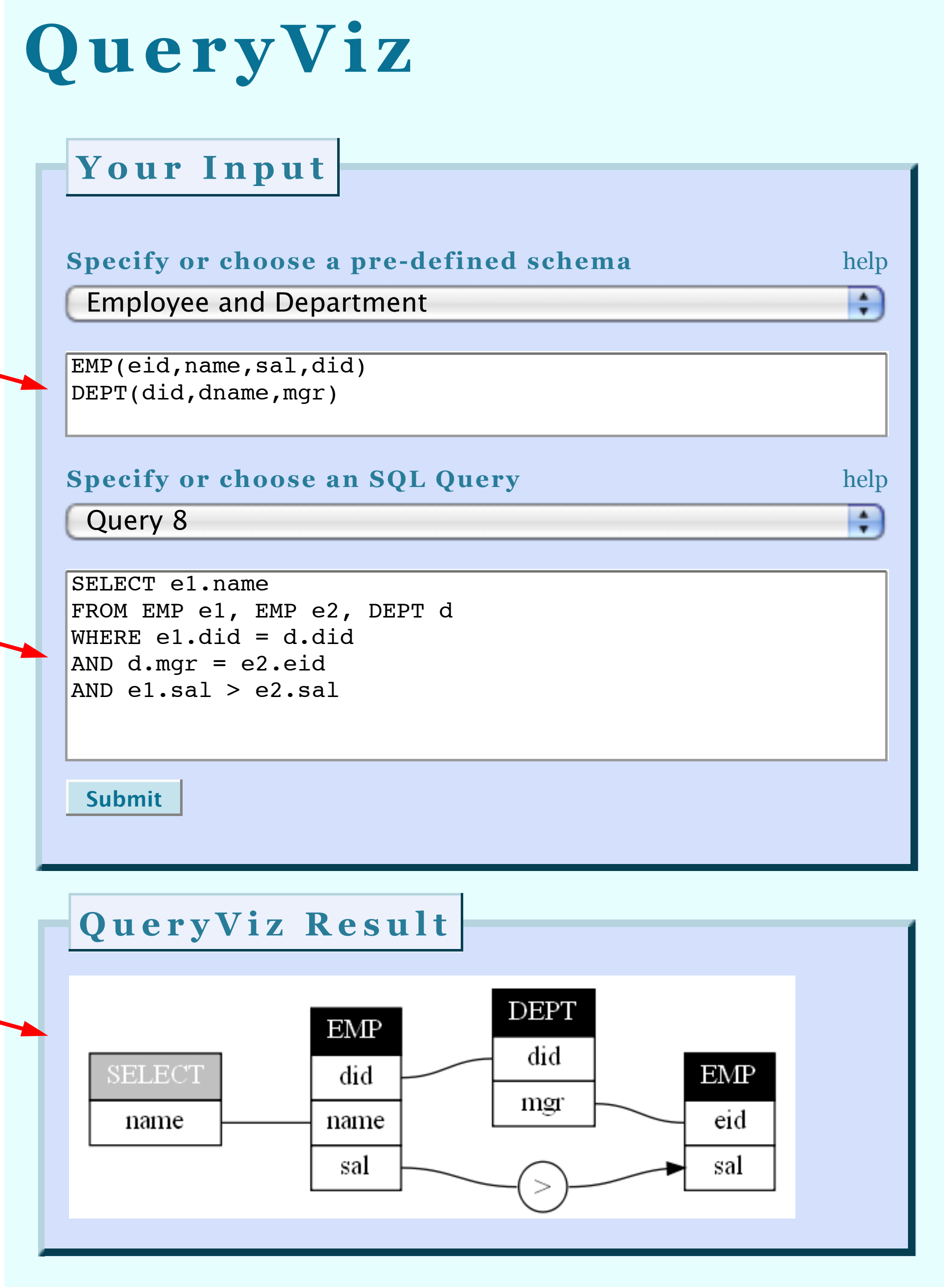
Relational Diagrams scoping

https://demo.queryvis.com

Input: Schema

Input Query

Output: Visualization



Danaparamita, G. [EDBT'11]

https://queryvis.com/

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

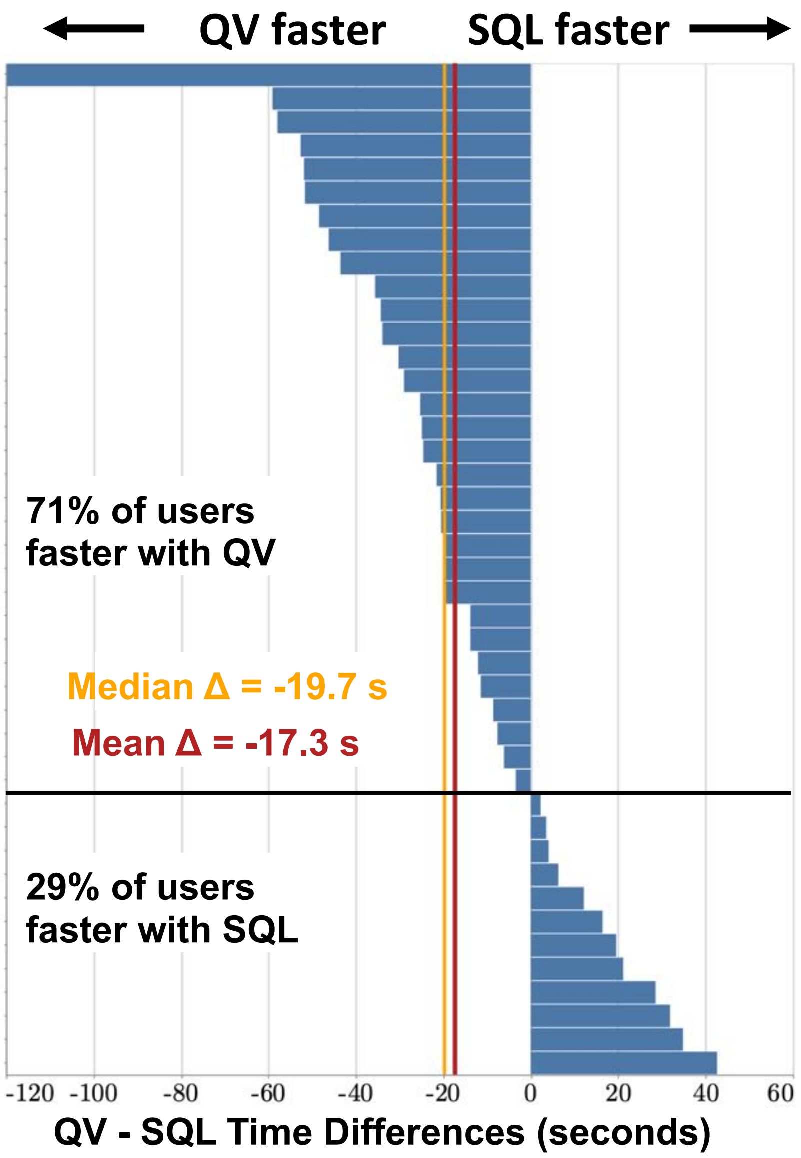
Source: Danaparamita, Gatterbauer: QueryViz: Helping users understand SQL queries and their patterns. EDBT 2011. https://doi.org/10.14778/3402755.3402805

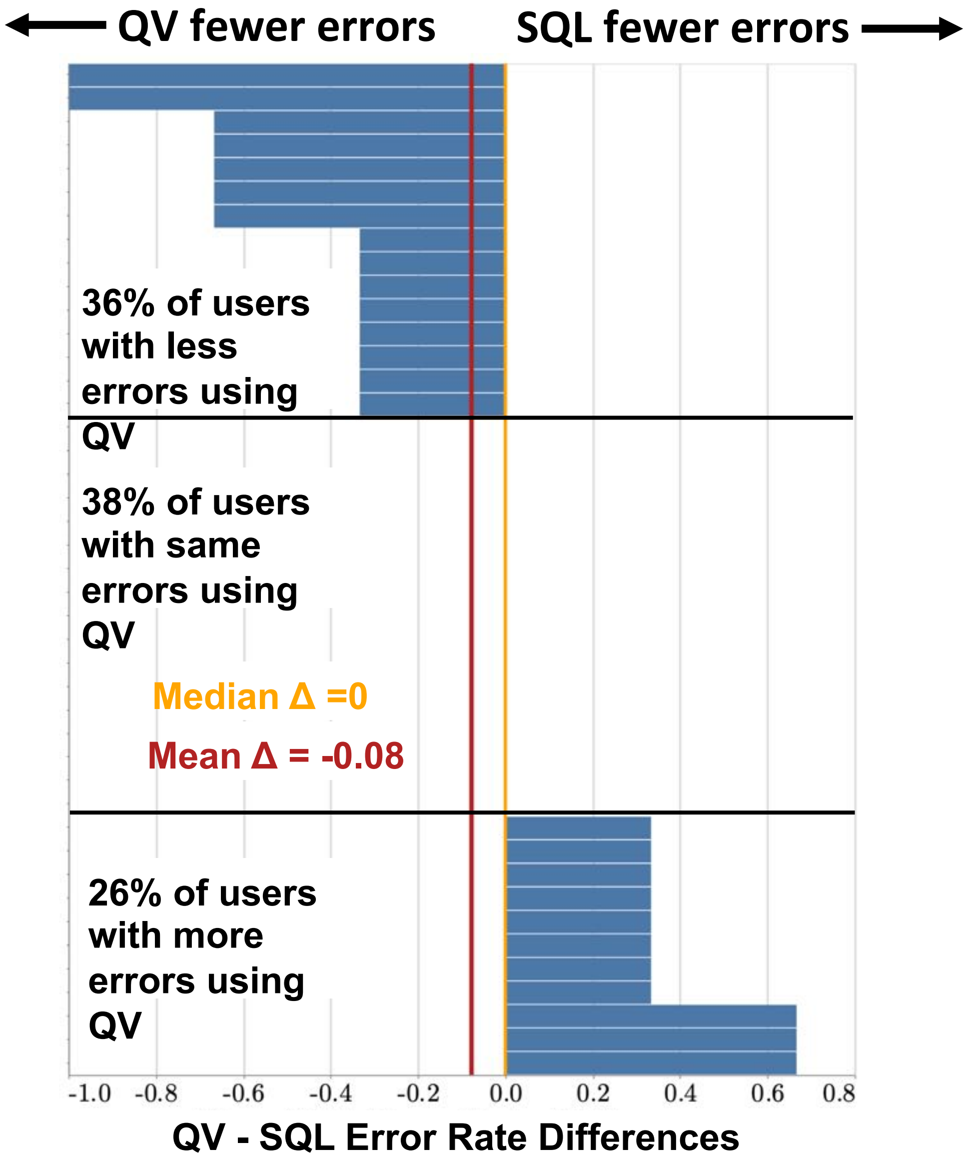
See also: Gatterbauer, Dunne, Jagadish, Riedewald: Principles of Query Visualization. IEEE Debull 2023. http://sites.computer.org/debull/A22sept/p47.pdf

Amazon Turk user study with SQL users

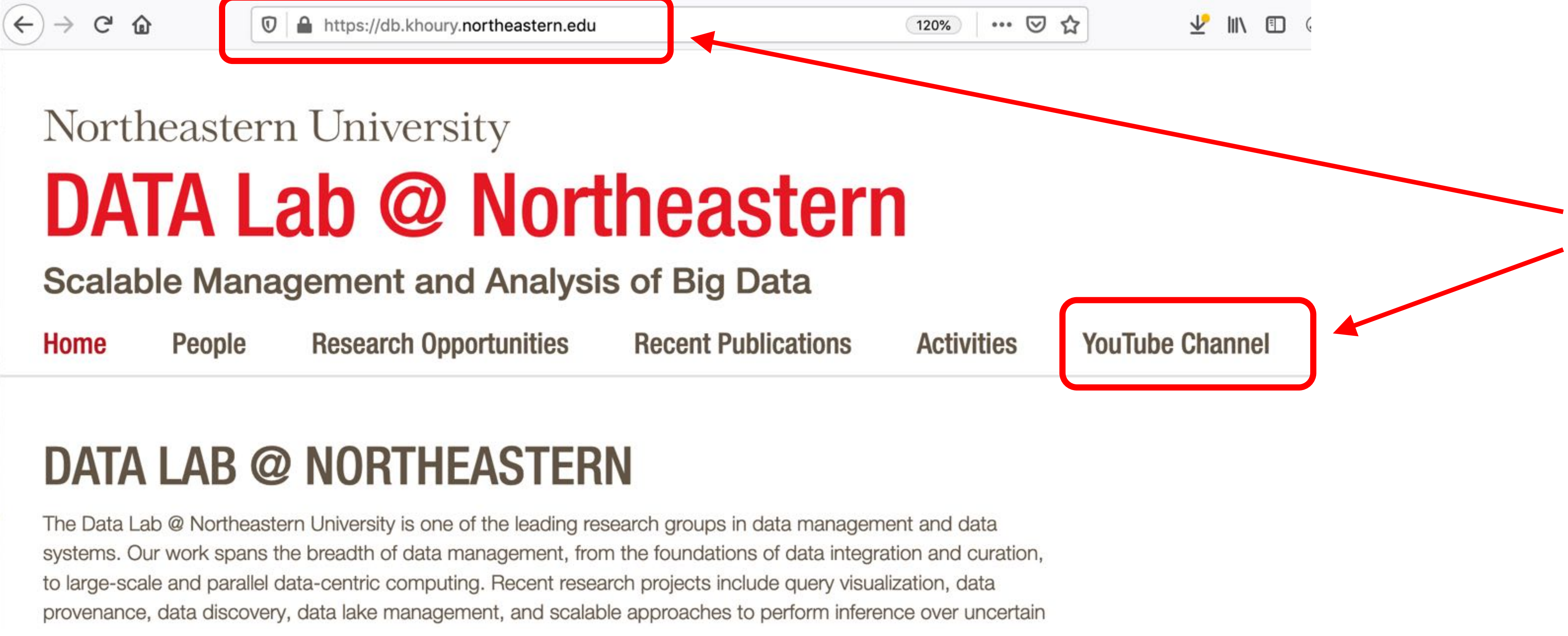
Leventidis+ [SIGMOD'20]

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

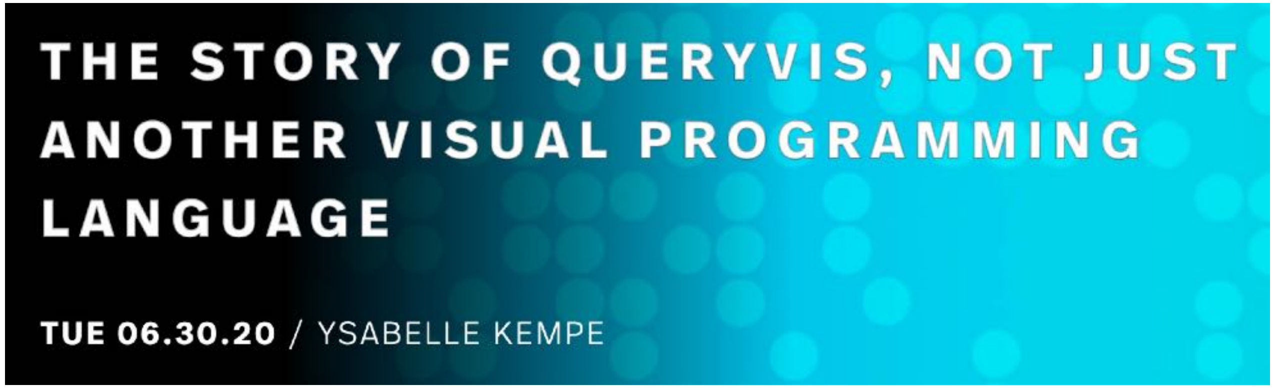




Source: Leventidis, Zhang, Dunne, Gatterbauer, Jagadish, Riedewald: QueryVis: Logic-based Diagrams help Users Understand Complicated SQL Queries Faster. SIGMOD 2020. https://doi.org/10.1145/3318464.3389767



https://queryvis.com



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

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

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.

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

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

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: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

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 (yet)

A new class without a grade

Student	Class	Semester	grade
Alice	cs3200	Fall 2022	B+
Bob	cs3200	Spring 2023	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(*))

- 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)

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 (alternative join syntax):

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

shortform is " JOIN"

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 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 INNER 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

shortform of:
"FULL OUTER JOIN"

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

~~SELECT *
FROM English INNER 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

Null also sometimes just shown as empty

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

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~~

```
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

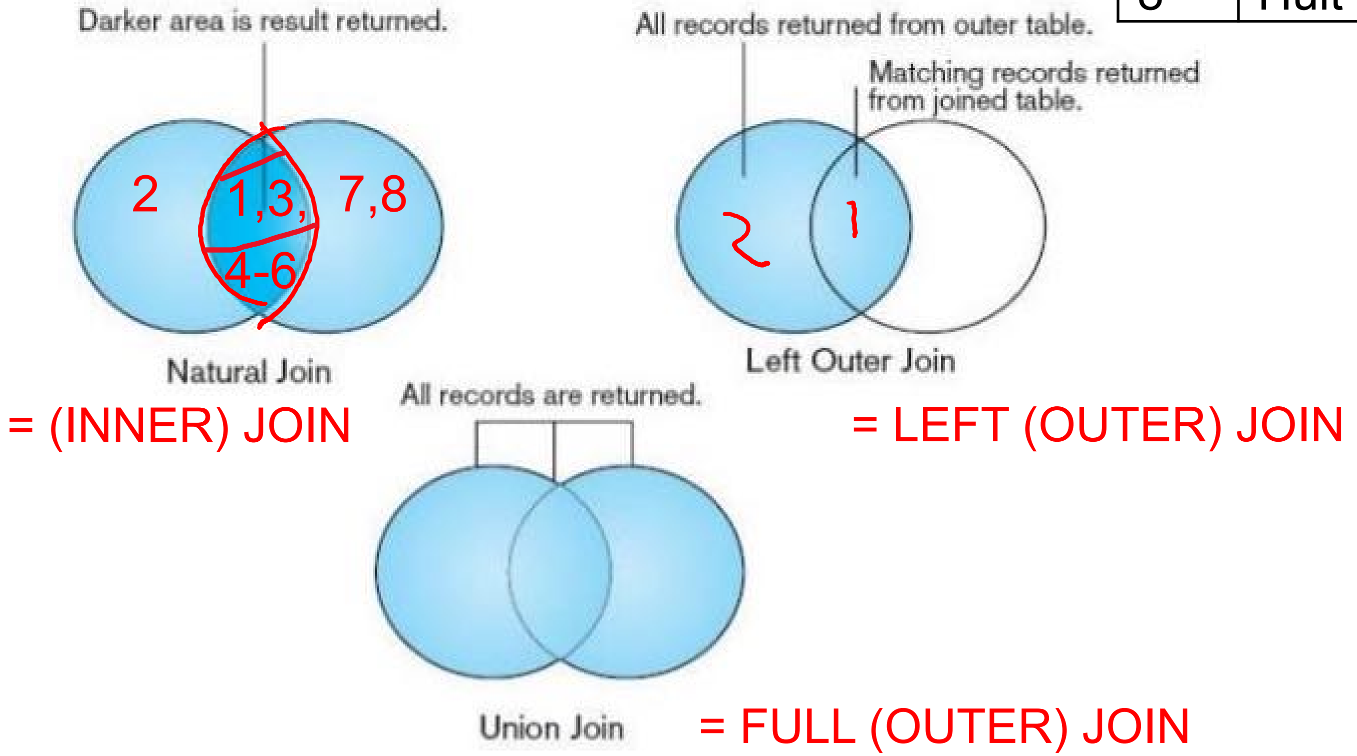
361

English

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

French

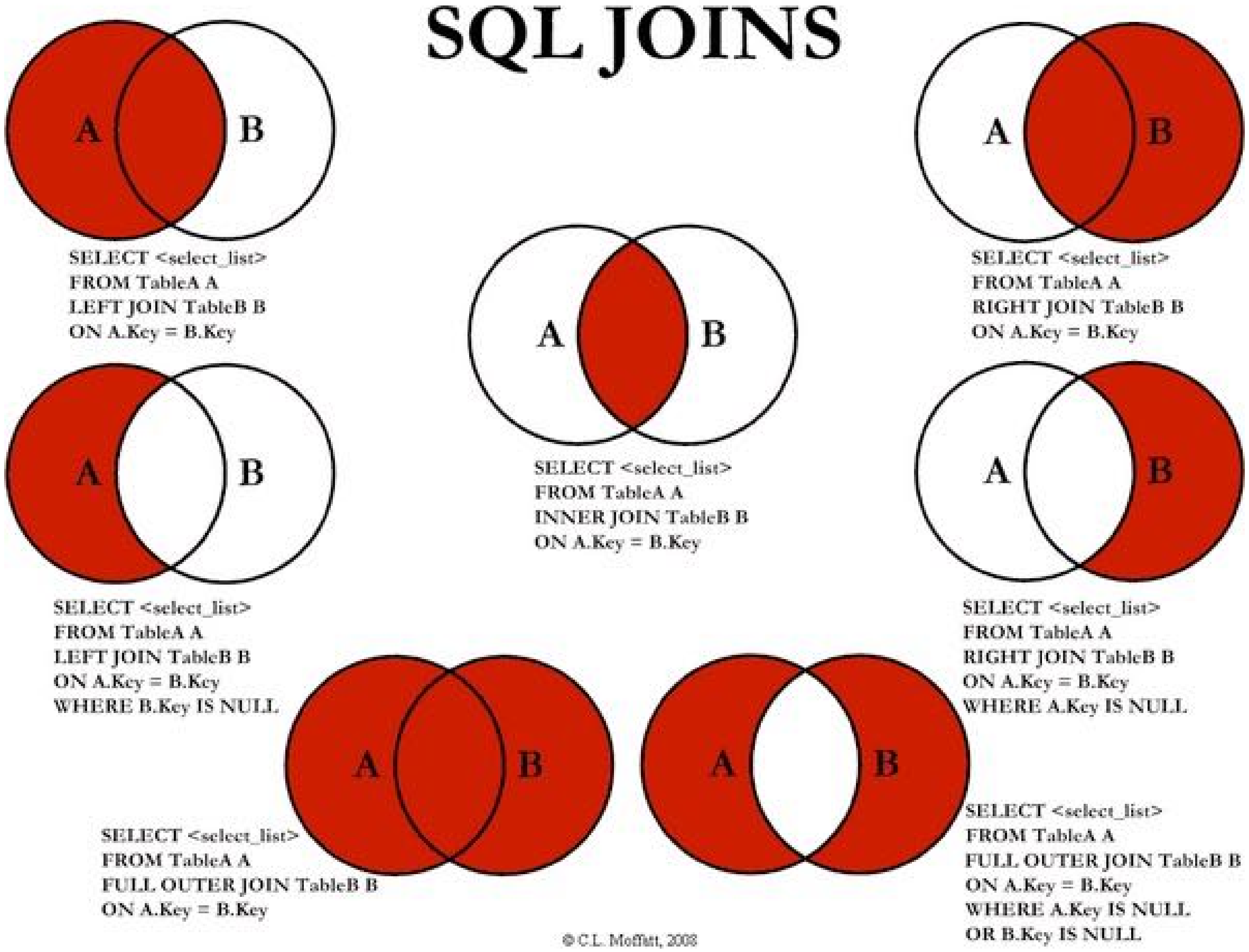
fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit



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

Detailed Illustration with Examples (follow the link)

also called "anti-join"

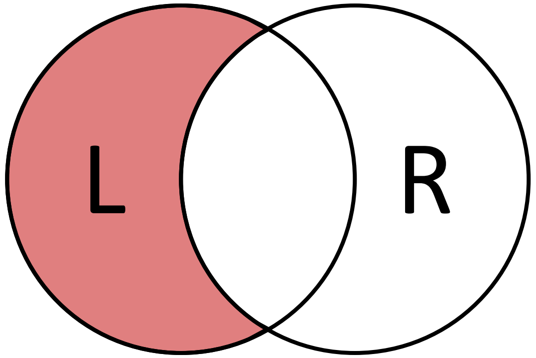


Check this web page for illustrating examples

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

Let's practice anti-joins


361




```
SELECT <select_list>
FROM L
LEFT JOIN R
ON L.key = R.key
WHERE R.key IS NULL
```

English

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

French

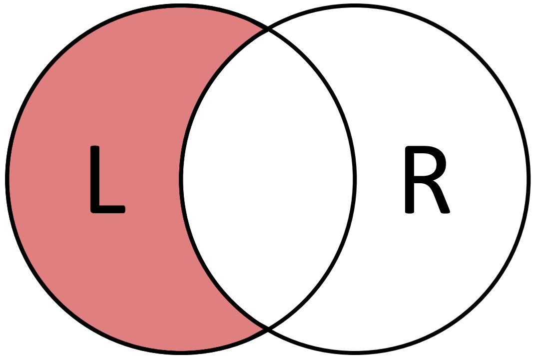
fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

?

361

Let's practice anti-joins




```
SELECT <select_list>
FROM L
LEFT JOIN R
ON L.key = R.key
WHERE R.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

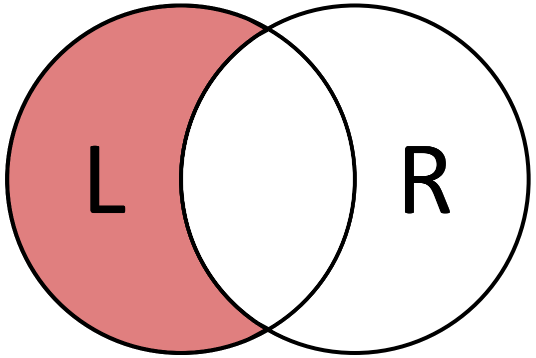
Results

eText	eid
Two	2

How to write in SQL?

?

Let's practice anti-joins




```
SELECT <select_list>
FROM L
LEFT JOIN R
ON L.key = R.key
WHERE R.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?

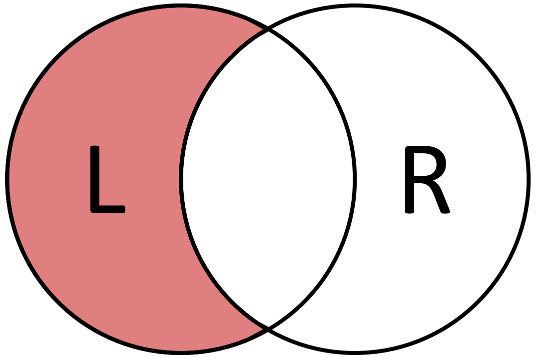
```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NULL
```

Any alternative?

?

Let's practice anti-joins

361




```
SELECT <select_list>
FROM L
LEFT JOIN R
ON L.key = R.key
WHERE R.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)
```

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

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?

?

```
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)
```

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

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)
```

Another look at Outer Joins

361

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

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

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

FULL JOIN can be written as union of inner join with anti-joins

?

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

Another look at Outer Joins

361

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

```
SELECT   etext,eid, fid, ftext
FROM     English INNER JOIN French
ON       English.eid = French.fid
UNION ALL
SELECT   etext, eid, NULL, NULL
FROM     English
WHERE    NOT EXISTS(
         SELECT *
         FROM French
         WHERE eid=fid)
UNION ALL
SELECT   NULL, NULL, fid, ftext
FROM     French
WHERE    NOT EXISTS(
         SELECT *
         FROM English
         WHERE eid=fid)
```

anti-join

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

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

Outer Joins, Coalesce, and non-associativity

Coalesce function


333

M

a
1
2

N

a
2
3

```
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

COALESCE: takes first non-NULL value,

```
SELECT COALESCE(1, NULL)
```

```
SELECT COALESCE(NULL, 3)
```

```
SELECT COALESCE(1, 2)
```

```
SELECT COALESCE(NULL, NULL)
```

?

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

Also see use of COALESCE across programming languages: https://en.wikipedia.org/wiki/Null_coalescing_operator

Coalesce function

M

a
1
2

N

a
2
3

```
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

COALESCE: takes first non-NULL value,
C(x,y,z)=C(x,C(y,z))=C(C(x,y),z)

Result

M.a	N.a	b
	?	?

`SELECT COALESCE(1, NULL)` ⇒ 1

`SELECT COALESCE(NULL, 3)` ⇒ 3

`SELECT COALESCE(1, 2)` ⇒ 1

`SELECT COALESCE(NULL, NULL)` ⇒ NULL

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Also see use of COALESCE across programming languages: https://en.wikipedia.org/wiki/Null_coalescing_operator

Coalesce function

M

a
1
2

N

a
2
3

```
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

COALESCE: takes first non-NULL value,
C(x,y,z)=C(x,C(y,z))=C(C(x,y),z)

Result

M.a	N.a	b
1	NULL	1
2	2	2
NULL	3	3

```
SELECT COALESCE(1, NULL)
```
⇒ 1

```
SELECT COALESCE(NULL, 3)
```
⇒ 3

```
SELECT COALESCE(1, 2)
```
⇒ 1

```
SELECT COALESCE(NULL, NULL)
```
⇒ NULL

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Also see use of COALESCE across programming languages: https://en.wikipedia.org/wiki/Null_coalescing_operator

Coalesce, Natural Outer Join, Union


333

M

a
1
2

N

a
2
3

```
SELECT *
FROM M
NATURAL FULL JOIN N
```

Result

a
1
2
3

Natural full join models "coalesce"

Join vs. Union – it is actually the same:
Union is a special case of a join ☺
(under set semantics)

Quick recap: Commutativity & Associativity

Multiplication

$3 \bullet 2 \bullet 4 = 24$

?

Multiplication is associative ☺

Matrix multiplication

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \bullet \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \bullet \begin{bmatrix} 3 \\ 1 \end{bmatrix} =$$

Quick recap: Commutativity & Associativity

Multiplication

$$(3 \bullet 2) \bullet 4 = 24$$

Order of operations can be exchanged:

$$3 \bullet (2 \bullet 4) = 24$$

Multiplication is associative ☺

?

and commutative ☺

Matrix multiplication

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \bullet \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \bullet \begin{bmatrix} 3 \\ 1 \end{bmatrix} =$$

Quick recap: Commutativity & Associativity

Multiplication

$(3 \bullet 2) \bullet 4 = 24$

Order of operations can be exchanged:

$3 \bullet (2 \bullet 4) = 24$

Multiplication is associative ☺

Order of operands can be exchanged:

$4 \bullet 2$

and commutative ☺

Matrix multiplication

$\begin{bmatrix} 2 & 3 \end{bmatrix} \bullet \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \bullet \begin{bmatrix} 3 \\ 1 \end{bmatrix} =$

?

Quick recap: Commutativity & Associativity

Multiplication

$$(3 \bullet 2) \bullet 4 = 24$$

Order of <u>operations</u> can be exchanged:

$$3 \bullet (2 \bullet 4) = 24$$

Multiplication is associative ☺

Order of <u>operands</u> can be exchanged:

$$4 \bullet 2$$

and commutative ☺

Matrix multiplication

$$\left(\begin{bmatrix}1 & 1\\2 & 3\end{bmatrix} \bullet \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\right) \bullet \begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}18\\49\end{bmatrix}$$

$$\begin{bmatrix}4 & 6\\11 & 16\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\2 & 3\end{bmatrix} \bullet \left(\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \bullet \begin{bmatrix}3\\1\end{bmatrix}\right) = \begin{bmatrix}18\\49\end{bmatrix}$$

$$\begin{bmatrix}5\\13\end{bmatrix}$$

Matrix multipl. is associative ☺

$$\begin{bmatrix}3\\1\end{bmatrix} \bullet \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$$

#col ≠ #row

... but *not* commutative ☹

It turns out this is mainly a problem of syntax, not semantics. Einstein notation (and similar more recent extensions like "EINSUM") solves that. See e.g. Laue et al. *A Simple and Efficient Tensor Calculus. AAAI 2020.* https://arxiv.org/abs/2010.03313 . Alternatively, think about the relational join operator as a commutative notation for sparse matrix multiplication

The power of associativity

Option 1: $\left(\begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} \bullet \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right) \bullet \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 18 \\ 49 \end{bmatrix}$

Option 2: $\begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} \bullet \left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \bullet \begin{bmatrix} 3 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 18 \\ 49 \end{bmatrix}$

Which option would you choose to evaluate this matrix multiplication ?

The power of associativity

Option 1: $$\left(\begin{bmatrix}1 & 1\\2 & 3\end{bmatrix} \bullet \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\right) \bullet \begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}18\\49\end{bmatrix}$$

$$\begin{bmatrix}4 & 6\\11 & 16\end{bmatrix}$$

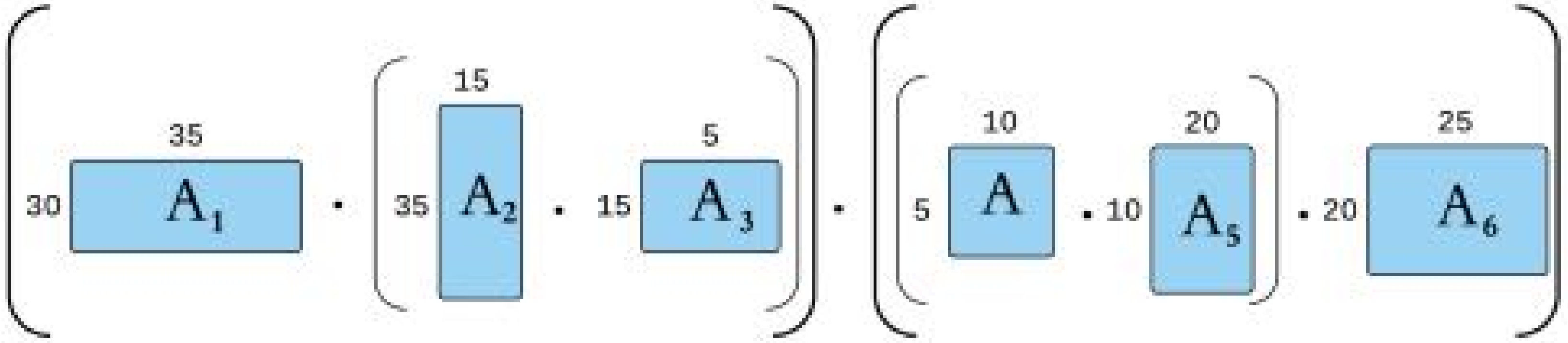
Option 2: $$\begin{bmatrix}1 & 1\\2 & 3\end{bmatrix} \bullet \left(\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \bullet \begin{bmatrix}3\\1\end{bmatrix}\right) = \begin{bmatrix}18\\49\end{bmatrix}$$

$$\begin{bmatrix}5\\13\end{bmatrix}$$

All variants give the same result. But some are faster.
Intuition: we like to have small intermediate result sizes!

Matrix chain multiplication

Given n matrices, what is the optimal sequence to multiply them? ?

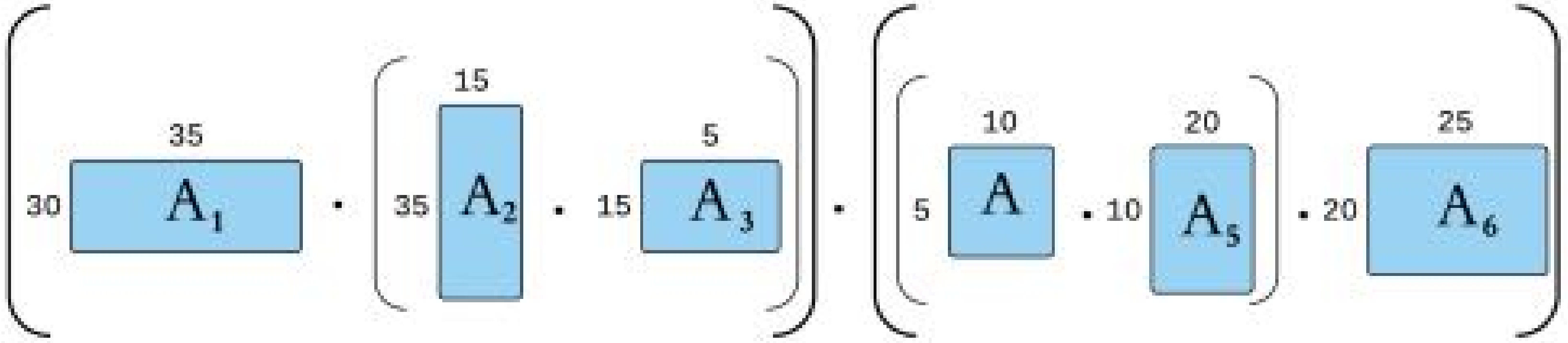


This is an example optimal factorization. What is its cost? ?

See also https://en.wikipedia.org/wiki/Catalan_number , https://en.wikipedia.org/wiki/Matrix_chain_multiplication , https://en.wikipedia.org/wiki/Matrix_multiplication#Associativity
Source figure: https://bruceoutdoors.wordpress.com/2015/11/24/matrix-chain-multiplication-with-c-code-part-3-extracting-the-sequence/

Matrix chain multiplication

Given n matrices, what is the optimal sequence to multiply them? ?



This is an example optimal factorization. What is its cost? ?

MinCost: (30*35*5 + (35*15*5)) + 30*5*25 + (5*10*20) + 5*20*25)

Nave method: all possible way to place closed parentheses: "Catalan numbers"

Via Dynamic programming: $O(n^3)$

Best known: $O(n \log n)$

C_n is the number of different ways $n + 1$ factors can be completely parenthesized (or the number of ways of associating n applications of a binary operator, as in the matrix chain multiplication problem). For $n = 3$, for example, we have the following five different parenthesizations of four factors:

((ab)c)d (a(bc))d (ab)(cd) a((bc)d) a(b(cd))

See also https://en.wikipedia.org/wiki/Catalan_number , https://en.wikipedia.org/wiki/Matrix_chain_multiplication , https://en.wikipedia.org/wiki/Matrix_multiplication#Associativity
Source figure: https://bruceoutdoors.wordpress.com/2015/11/24/matrix-chain-multiplication-with-c-code-part-3-extracting-the-sequence/

Commutativity & Associativity

333

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

?

Result

?

Commutativity & Associativity

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

?

Result

?

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

Commutativity & Associativity

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

A	B	C
1	2	3

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

A	B	C
1	2	3
4	NULL	5

Result

?

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

Commutativity & Associativity

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

A	B	C
1	2	3
4	NULL	5

Result

?

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

Commutativity & Associativity

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

A	B	C
⊥	2	3
4	⊥	5

```sql
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

Result

A	B	C
1	2	3
4	NULL	5

```sql
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

A	B	C
1	2	NULL
NULL	2	3
4	NULL	5

Thus outer joins are not associative! (but they are commutative)

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

Commutativity & Associativity

333

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

```
SELECT R.a RA, T.a TA, coalesce(R.a, T.a) a,
       R.b RB, S.b SB, coalesce(R.b, S.b) b,
       S.c SC, T.c TC, coalesce(S.c, T.c) c
FROM (R
FULL JOIN S on R.B=S.B)
FULL JOIN T on S.C=T.C AND R.A = T.A
```

```
SELECT R.a RA, T.a TA, coalesce(R.a, T.a) a,
       R.b RB, S.b SB, coalesce(R.b, S.b) b,
       S.c SC, T.c TC, coalesce(S.c, T.c) c
FROM R
FULL JOIN (S
FULL JOIN T on S.C=T.C) on R.B=S.B AND R.A = T.A
```

Data Output | Explain | Messages | Notifications

	ra integer	ta integer	a integer	rb integer	sb integer	b integer	sc integer	tc integer	c integer
1	1	[null]	1	2	2	2	3	[null]	3
2	[null]	4	4	[null]	[null]	[null]	[null]	5	5

Data Output | Explain | Messages | Notifications

	ra integer	ta integer	a integer	rb integer	sb integer	b integer	sc integer	tc integer	c integer
1	[null]	[null]	[null]	[null]	2	2	3	[null]	3
2	[null]	4	4	[null]	[null]	[null]	[null]	5	5
3	1	[null]	1	2	[null]	2	[null]	[null]	[null]

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM (Accommodations
NATURAL FULL JOIN Climates)
NATURAL FULL JOIN Sites
```

Result

?

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

Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM (Accommodations
NATURAL FULL JOIN Climates)
NATURAL FULL JOIN Sites
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada					Mount Logan
UK	London				Buckingham
UK	London				Hyde Park
UK		temperate			
Bahamas	Nassau	Tropical	Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM (Accommodations
NATURAL FULL JOIN Climates)
NATURAL FULL JOIN Sites
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada					Mount Logan
UK	London				Buckingham
UK	London				Hyde Park
UK		temperate			
Bahamas	Nassau	Tropical	Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM (Accommodations
NATURAL FULL JOIN Climates)
NATURAL FULL JOIN Sites
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada					Mount Logan
UK	London				Buckingham
UK	London				Hyde Park
UK		temperate			
Bahamas	Nassau	Tropical	Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```sql
SELECT  *
FROM Accommodations
NATURAL FULL JOIN (Climates
NATURAL FULL JOIN Sites)
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto		Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada		diverse			Mount Logan
UK	London	temperate			Buckingham
UK	London	temperate			Hyde Park
Bahamas		Tropical			
Bahamas	Nassau		Hilton		

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

Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM Accommodations
NATURAL FULL JOIN (Climates
NATURAL FULL JOIN Sites)
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto		Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada		diverse			Mount Logan
UK	London	temperate			Buckingham
UK	London	temperate			Hyde Park
Bahamas		Tropical			
Bahamas	Nassau		Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Full disjunction

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM FULL DISJUNCTION(Climates,
(Accommodations, Sites)
```

FD: variation of the join operator that maximally combines join consistent tuples from connected relations, while preserving all information in the relations.

Not available in SQL! We may discuss later in class in more detail (or skip this year)

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada		diverse			Mount Logan
UK	London	temperate			Buckingham
UK	London	temperate			Hyde Park
Bahamas	Nassau	tropical	Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Full disjunction: definition

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

t_1 t_2 t_3

Accommodations

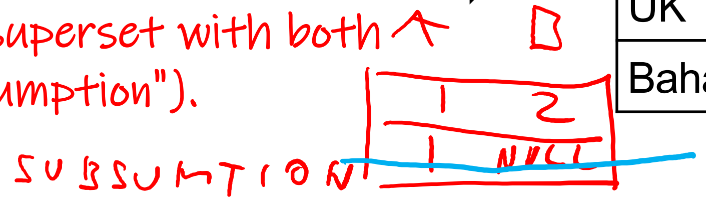
Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

t_4

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

- Two tuples (max one from each relation) are join consistent if they agree on common attributes, e.g. t_1/t_2, t_3/t_4. A set of tuples is join consistent if every pair is join consistent.
- Set of tuples (max one from each relation) is connected if the schema is connected, thus share attributes
- A tuple is in the Full disjunction if it is the inner join from tuples that are connected, join consistent, and there is no superset with both conditions (related to "subsumption").



Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada		diverse			Mount Logan
UK	London	temperate			Buckingham
UK	London	temperate			Hyde Park
Bahamas	Nassau	tropical	Hilton		

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

Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf