Updated 1/24/2024

Topic 1: Data models and query languages
Unit 2: Logic & relational calculus
Lecture 4

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

1/23/2024

Pre-class conversations

- Last class summary
- New class members: quick introduction
    1. What area are you working on? Who is your PhD advisor?
    2. What do you hope to get out of this course ☺
    3. What is your biggest fear for this course ☹
    4. What the topic from the course that are you most familiar with or excited about?
- Quick comments on my "slide posting policy"
- Please keep asking questions, in class and/or on Piazza

- Today:
    - Logic as the foundation for relational databases

PRELIMINARY

CS 7240: Topics and approximate agenda (Spring'24)

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), or latest two days after class. Notice that I post one single slide deck for each unit (e.g. Topic 1 - Unit 1- SQL), and I keep those slide decks updated as we progress with the unit across lectures. 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 1/9):** Course introduction / T1-U1 SQL / PostgreSQL setup / SQL Activities
- **Lecture 2 (Fri 1/12):** T1-U1 SQL T1-U1 SQL
- **Lecture 3 (Tue 1/16) via Zoom:** T1-U1 SQL
- **Lecture 4 (Fri 1/19): no class**
- **Lecture 5 (Tue 1/23):** T1-U2 Logic & Relational Calculus
- **Lecture 6 (Fri 1/26):** T1-U2 Logic & Relational Calculus
- **Lecture 7 (Tue 1/30):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 8 (Fri 2/2):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 9 (Tue 2/6):** T1-U4 Datalog & Recursion & ASP
- **Lecture 10 (Tue 2/9):** T1-U4 Datalog & Recursion & ASP
- **Lecture 11 (Tue 2/13):** T1-U4 Datalog & Recursion & ASP
- **Lecture 12 (Fri 2/16):** T1-U4 Datalog & Recursion & ASP

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**

PRELIMINARY

- **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

Pointers to relevant concepts & supplementary material:

- **Unit 1. SQL**: [SAMS'12], [CS 3200], [Cow'03] Ch3 & Ch5, [Complete'08] Ch6, [Silberschatz+'20] Ch3.8
- **Unit 2. Logic & Relational Calculus**: First-Order Logic (FOL), relational calculus (RC): [Barland+'08] 4.1.2 & 4.2.1 & 4.4, [Genesereth+] Ch6, [Halpern+'01], [Cow'03] Ch4.3 & 4.4, [Elmasri, Navathe'15] Ch8.6 & Ch8.7, [Silberschatz+'20] Ch27.1 & Ch27.2, [Alice'95] Ch3.1-3.3 & Ch4.2 & Ch4.4 & Ch5.3-5.4, [Barker-Plummer+'11] Ch11
- **Unit 3. Relational Algebra & Codd's Theorem**: Relational Algebra (RA), Codd's theorem: [Cow'03] Ch4.2, [Complete'08] Ch2.4 & Ch5.1-5.2, [Elmasri, Navathe'15] Ch8, [Silberschatz+'20] Ch2.6, [Alice'95] Ch4.4 & Ch5.4
- **Unit 4. Datalog & Recursion**: Datalog, recursion, Stratified Datalog with negation, Datalog evaluation strategies, Stable Model semantics, Answer Set Programming (ASP): [Complete'08] Ch5.3, [Cow'03] Ch 24, [Koutris'19] L9 & L10, [G., Suciu'10]
- **Unit 5. Alternative Data Models**: NoSQL: [Hellerstein, Stonebraker'05], [Sadalage, Fowler'12], [Harrison'16]

Topic 1: Data models and query languages

- U1: SQL
  - [SAMS'19] SAMS: Teach yourself SQL in 10min by Forta. 5th ed. 2019. It is available for free for Northeastern students from Safari books eBook (you may have to first login from our library website, then try again the previous link). If the book is checked out online, you can use the 4th edition (there is almost no difference between 4th and 5th ed) as Safari books eBook, or as EBSCOhost eBook.
  - [cs3200] PostgreSQL setup, PgAdmin 4 tutorial. Files to follow along our SQL lectures: SQL Activities.
  - [Cow'03] Ramakrishnan, Gehrke. Database Management Systems. 3rd ed 2003. Ch 5: SQL.
  - [Complete'08] Garcia-Molina, Ullman, Widom. Database Systems. 2nd ed 2008. Ch 6: SQL.
  - [Elmasri, Navathe'15] Fundamentals of Database Systems. 7th ed 2015. Ch 6: SQL
  - [Silberschatz+'10] Silberschatz, Korth, Sudarshan. Database system concepts. 6th ed 2011. Ch 3.8: Nested subqueries.
- U2: Logic, relational calculus
  - [Barland+'08] Barland, Kolaitis, Vardi, Felleisen, Greiner. Intro to Logic, (alternative PDF version). 4.1.2 First-Order Logic: bound variables, free variables, 4.2.1 First-Order Logic: equivalences, 4.4 Exercises for First-Order Logic.
  - [Genesereth+] Genesereth et al. Introduction to logics. Ch 6: Relational Logic.
  - [Halpern+'01] Halpern, Harper, Immerman, Kolaitis, Vardi, Vianu. On the Unusual Effectiveness of Logic in Computer Science. Bulletin of Symbolic Logic 2001.
  - [Cow'03] Ramakrishnan, Gehrke. Database Management Systems. 3rd ed 2003. Ch 4.3: Relational calculus, Ch 4.4: Safety.
  - [Elmasri, Navathe'15] Fundamentals of Database Systems. 7th ed 2015. Ch 8.6: Tuple relational calculus, Ch 8.7: Domain relational calculus.
  - [Silberschatz+'10] Silberschatz, Korth, Sudarshan. Database system concepts. 6th ed 2011. Ch 6.2: Tuple relational calculus, Ch 6.3: Domain relational calculus.
  - [Alice'95] Abiteboul, Hull, Vianu. Foundations of Databases. 1995. Ch 3.1: Structure of the relational model, Ch 3.2: Named vs. unnamed perspective, Ch 3.3: Conventioanl vs. logic programming perspective, Ch 4.2: Logic-based perspective, Ch 4.4: Algebraic perspectives, Ch 5.3: Relational calculus, domain independence, Codd's theorem, Ch 5.4: Syntactic Restrictions for Domain Independence.
  - [Barker-Plummer+'11]: Barker-Plummer, Barwise, Etchemendy. Language, Proof and Logic. 2nd ed. 2011. Ch 11: Multiple Quantifiers.

Please feel free to point me to other interesting material you find or alrady know of ☺

Queries and the connection to logic

- Why logic?
- A crash course in FOL
- Relational Calculus (RC)
  - Syntax and Semantics
  - Domain RC (DRC) vs Tuple RC (TRC)
  - Domain Independence and Safety
- 4 categorical propositions

Logic as foundation of Computer Science and Databases

- Logic has had an immense impact on CS
- Computing has strongly driven a particular branch of logic: finite model theory
  - That is, First-order logic (FOL) restricted to finite models
  - Has strong connections to complexity theory
  - The basis of various branches in Artificial Intelligence (not the ones favored today)
- It is a natural tool to capture and attack fundamental problems in data management
  - Relations as first-class citizens
  - Inference for assuring data integrity (integrity constraints)
  - Inference for question answering (queries)
- It has been used for developing and analyzing the relational model from the early days [Codd'72]

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.

See also: Halpern, Harper, Immerman, Kolaitis, Vardi, Vianu. "*On the unusual effectiveness of logic in computer science*", 2001. https://doi.org/10.2307/2687775

A play on: Wigner. "*The unreasonable effectiveness of mathematics in the natural sciences*", 1960. https://doi.org/10.1142/9789814503488_0018

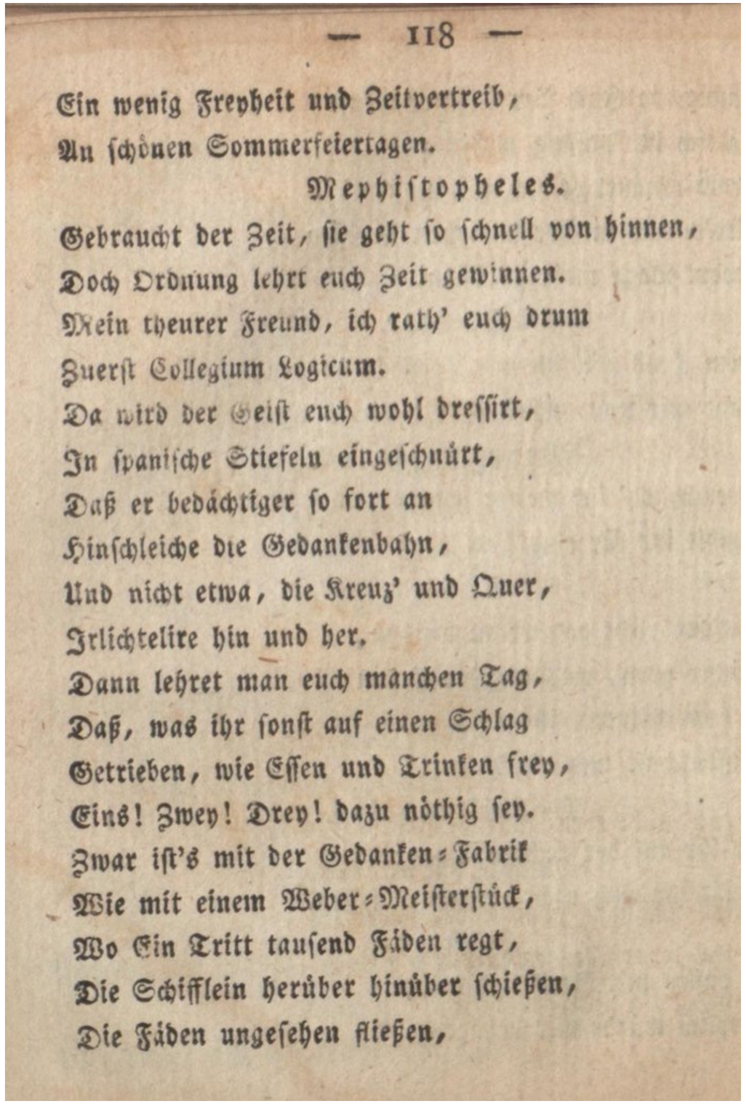
Why has Logic turned out to be so powerful?

- Basic Question: What on earth does an obscure, old intellectual discipline have to do with the youngest intellectual discipline?
- Cosma R. Shalizi, CMU:
  - “If, in 1901, a talented and sympathetic outsider had been called upon (say, by a granting-giving agency) to survey the sciences and name the branch that would be least fruitful in century ahead, his choice might well have settled upon mathematical logic, an exceedingly recondite field whose practitioners could all have fit into a small auditorium. It had no practical applications, and not even that much mathematics to show for itself: its crown was an exceedingly obscure definition of cardinal numbers.”

See here for pointers to some of these discussions: https://en.wikipedia.org/wiki/Cardinal_number

Source: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Logics as the start of everything ["Mephistopheles" 1806]


— 118 —

Ein wenig Freyheit und Zeitvertreib,
An schönen Sommerfeiertagen.

Mephistopheles.

Gebraucht der Zeit, sie geht so schnell von hinnen,
Doch Ordnung lehrt euch Zeit gewinnen.
Mein theurer Freund, ich rath' euch drum
Zuerst Collegium Logicum.
Da wird der Geist euch wohl dressirt,
In spanische Stiefeln eingeschnürt,
Daß er bedächtiger so fort an
Hinschleiche die Gedankenbahn,
Und nicht etwa, die Kreuz' und Quer,
Irrlichtelire hin und her.
Dann lehret man euch manchen Tag,
Daß, was ihr sonst auf einen Schlag
Getrieben, wie Essen und Trinken frey,
Eins! Zwey! Drey! dazu nöthig sey.
Zwar ist's mit der Gedanken-Fabrik
Wie mit einem Weber-Meisterstück,
Wo Ein Tritt tausend Fäden regt,
Die Schifflein herüber hinüber schießen,
Die Fäden ungesehen fließen,

GERMAN	ENGLISH TRANSLATION
Mephistopheles.	Mephistopheles.
Gebraucht der Zeit, sie geht so schnell von hinnen,	Use your time well: it slips away so fast, yet
Doch Ordnung lehrt Euch Zeit gewinnen.	Discipline will teach you how to win it.
Mein teurer Freund, ich rat Euch drum	My dear friend, I'd advise, in sum,
<u>Zuerst Collegium Logicum</u>.	<u>First, the Collegium Logicum.</u>
Da wird der Geist Euch wohl dressiert,	There your mind will be trained,
In spanische Stiefeln eingeschnürt,	As if in Spanish boots, constrained,
Daß er bedächtiger so fortan	So that painfully, as it ought,
Hinschleiche die Gedankenbahn,	It creeps along the way of thought,
Und nicht etwa, die Kreuz und Quer,	Not flitting about all over,
Irrlichteliere hin und her.	Wandering here and there.
...	...

Source: Johan Wolfgang von Goethe. Faust Part I: Scene IV: The Study. ~1806. https://www.deutschestextarchiv.de/book/view/goethe_faust01_1808?p=124 ,
English Translation: https://www.poetryintranslation.com/PITBR/German/FaustIScenesIVtoVI.php

Back to The Future

- M. Davis (1988): Influences of Mathematical Logic on Computer Science:
  - “When I was a student, even the topologists regarded mathematical logicians as living in outer space. Today the connections between logic and computers are a matter of engineering practice at every level of computer organization.”

- Question: Why on earth?

Source: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Birth of Computer Science: 1930s

- Church, Gödel, Kleene, Post, Turing: Mathematical proofs have to be "machine checkable" - computation lies at the heart of mathematics!
  - Fundamental Question: What is "machine checkable"?
- Fundamental Concepts:
  - algorithm: a procedure for solving a problem by carrying out a precisely determined sequence of simpler, unambiguous steps
  - distinction between hardware and software
  - a universal machine: a machine that can execute arbitrary programs
  - a programming language: notation to describe algorithms

Source: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Leibniz's Dream

An Amazing Dream: a universal mathematical language, lingua characteristica universalis, in which all human knowledge can be expressed, and calculational rules, calculus ratiocinator, carried out by machines, to derive all logical relationships

- "If controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, and say to each other: Calculemus–Let us calculate."

Source: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Example: Aristotle' Syllogisms

- "All humans are mortal"

?

Example: Aristotle' Syllogisms

- "All humans are mortal"

- "For all x, if x is a human, then x is mortal"

?

Example: Aristotle' Syllogisms

- "All humans are mortal"
- "For all x, if x is a human, then x is mortal"
- $\forall x\ [\text{Human}(x) \rightarrow \text{Mortal}(x)]$

Do you see the connection to referential integrity constraints ?

Product

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

Company

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

Based on: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Example: Aristotle' Syllogisms

- "All humans are mortal"

- "For all x, if x is a human, then x is mortal"

- $\forall x\ [\text{Human}(x) \rightarrow \text{Mortal}(x)]$

Do you see the connection to referential integrity constraints

$\forall x\ [\text{Product}(\_,\_,\_,x) \rightarrow \text{Company}(x,\_,\_,\_)]$

Product

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

Company

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

Based on: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Logic and Databases

Two main uses of logic in databases:

- Logic used as a database query language to express questions asked against databases (our main focus)
- Logic used as specification language to express integrity constraints in databases (product/company example from previous slide)

Why Logic?

- Logic provides both a unifying framework and a set of tools for formalizing and studying data management tasks.

Logic in Computer Science

- During the past fifty years there has been extensive, continuous, and growing interaction between logic and computer science. In many respects, logic provides computer science with both a unifying foundational framework and a tool for modeling computational systems. In fact, logic has been called "the calculus of computer science".
- The argument is that logic plays a fundamental role in computer science, similar to that played by calculus in the physical sciences and traditional engineering disciplines.
  - Indeed, logic plays an important role in areas of computer science as disparate as machine architecture, computer-aided design, programming languages, databases, artificial intelligence, algorithms, and computability and complexity.

Source: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Queries and the connection to logic

- Why logic?
- A crash course in FOL
- Relational Calculus (RC)
  - Syntax and Semantics
  - Domain RC (DRC) vs Tuple RC (TRC)
  - Domain Independence and Safety
- 4 categorical propositions

First-Order Logic: some notions

- Objects, e.g., “2” or “Alice”
- Predicates (relations), e.g., “$2 < 3$”
  - notice predicates are Boolean-valued functions (the codomain is Boolean)
  - e.g., Define $f(x,y)=\text{true}$ iff $x<y$. Thus $f(2,3)=\text{true}$
- Operations (non-Boolean functions), e.g., “$2 + 3$”
  - such functions usually return an object from the same domain as the inputs
- Logical operations, e.g., “and” ($\wedge$), “or” ($\vee$), “implies” ($\rightarrow$)
  - Both inputs and outputs are Boolean
- Quantifiers, e.g., “for all” ($\forall$), “exists” ($\exists$)

First-Order Logic

- A formalism for specifying properties of mathematical structures, such as graphs, partial orders, groups, rings, fields, . . .
- For any given structure, we can verify whether the properties hold

$D = \{1,2,3\}$

$D^m \rightarrow \{T, F\}$

- Mathematical Structure:
  - $A = (D, R_1, \ldots, R_k, f_1, \ldots, f_l)$
  - D is a non-empty set: universe, or domain
  - R_i is an m-ary relation on D, for some m (i.e., $R_i \subseteq D^m$)
  - f_j is an n-ary function on D, for some n (i.e., $f_i\colon D^n \rightarrow D$)

$f(w_1, w_2) = w_1 + w_2$

R

A	B	C
1	1	1
2	1	3

$\subseteq D \times D \times D$

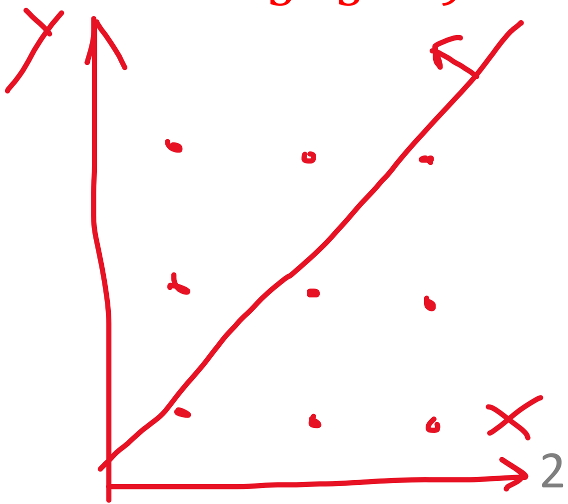
$3 \cdot 3 \cdot 3 = 27$

<

x	y
1	2
1	3
2	3

$\subseteq D \times D$

$3 \cdot 3 = 9$



Based on: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Two examples of "Mathematical Structures"

- Graph $G = (V, E)$

?

- Groups $G = (D, \cdot)$

?

Two examples of "Mathematical Structures"

- Graph $G = (V, E)$
  - V: set of nodes
  - $E \subseteq V^2$: edges, a binary relation on V
- Groups $G = (D, \cdot)$

?

Two examples of "Mathematical Structures"

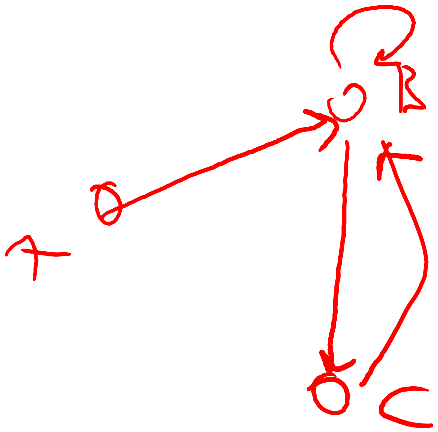
- Graph $G = (V, E)$
  - V: set of nodes
  - $E \subseteq V^2$: edges, a binary relation on V

- Groups $G = (D, \cdot)$
  - D: elements
  - "$\cdot$": $D^2 \rightarrow D$: group operation
  - Example: $(\mathbb{Z}, +)$: Integers under addition
    - groups also require following conditions:
      - an identity element e specified by $\exists e \in D\ \forall x \in D [e+x = x+e = x]$ and often written explicitly as in $(\mathbb{Z}, +, 0)$
      - the associativity of the operation $(x+y)+z = x+(y+z)$, and
      - an inverse element $\forall x \in D\ \exists(-x) \in D\ [(-x)+x = x+(-x) = e]$

First-Order Logic on Graphs

Syntax:

- First-order variables: $x, y, z, \ldots$ (range over nodes)
- Atomic formulas: $E(x, y), x = y$
- Formulas:
  - Atomic Formulas, and
  - Boolean Connectives (∨, ∧, ¬), and
  - First-Order Quantifiers $\exists x, \forall x$



How to represent that graph in relations?

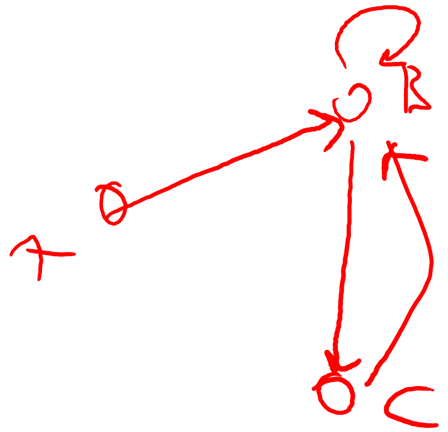
?

Source: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

First-Order Logic on Graphs

Syntax:

- First-order variables: $x, y, z, \ldots$ (range over nodes)
- Atomic formulas: $E(x, y), x = y$
- Formulas:
  - Atomic Formulas, and
  - Boolean Connectives (∨, ∧, ¬), and
  - First-Order Quantifiers $\exists x, \forall x$



binary edge relation

Edge

from	to
A	B
B	B
C	B
B	C

Vertex

name
A
B
C

notice that we will use "edge" and "E" for both directed and undirected edges (instead of "arc" for directed)

Example properties for graph

Assume schema E(source, target) is undirected.
Thus for every edge E(x,y), we also have E(y,x).

- “node 'a' has at least two distinct neighbors”

?

- “each node has at least two distinct neighbors”

?

Example adopted from: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Example properties for graph

Assume schema E(source, target) is undirected.
Thus for every edge E(x,y), we also have E(y,x).

- “node 'a' has at least two distinct neighbors”
  - $\exists y\, \exists z\, [E('a', y) \wedge E('a', z) \wedge y \neq z]$
  - Notice that if we replace $'a'$ with a variable x (which is then free) in the above formula, then this becomes a query (find nodes x that have ...). Let's do that!

- “each node has at least two distinct neighbors”

?

Example adopted from: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Example properties for graph

Assume schema E(source, target) is undirected.
Thus for every edge E(x,y), we also have E(y,x).

- "node x has at least two distinct neighbors"
  - $\exists y\ \exists z\ [E(x, y) \wedge E(x, z) \wedge y \neq z]$
  - Notice: x is free in the above formula, which expresses a property of a node x.
  - You can also think about this as a query (find nodes x that have ...)

- "each node has at least two distinct neighbors"

?

Example properties for graph

Assume schema E(source, target) is undirected.
Thus for every edge E(x,y), we also have E(y,x).

- "node x has at least two distinct neighbors"
  - $\exists y\, \exists z\, [E(x,y) \land E(x,z) \land y{\neq}z]$
  - Notice: x is free in the above formula, which expresses a property of a node x.
  - You can also think about this as a query (find nodes x that have ...)

- "each node has at least two distinct neighbors"
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \land E(x,z) \land y{\neq}z]$
  - The above is a sentence, that is, a formula with no free variables; it expresses a property of graphs.

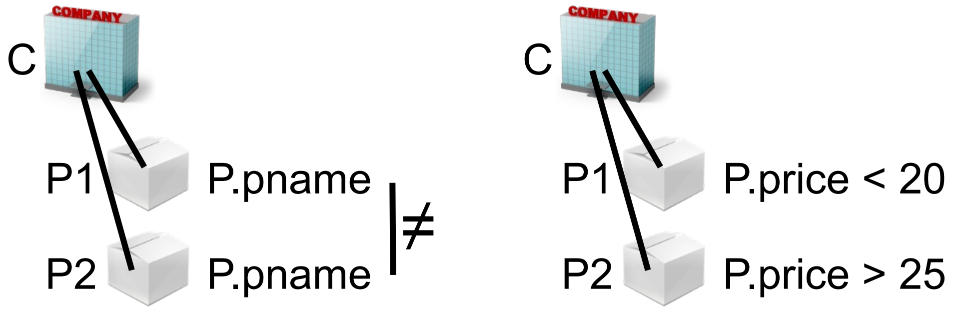
We will sometimes use ∃x,y,z as short form for ∃x∃y∃z

Example adopted from: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Now in SQL

E(S,T) 501

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \land E(x,z) \land y \neq z]\}$




```
SELECT
FROM        ?
WHERE
```

- "each node has at least two distinct neighbors" (statement = Boolean query)
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \land E(x,z) \land y \neq z]$

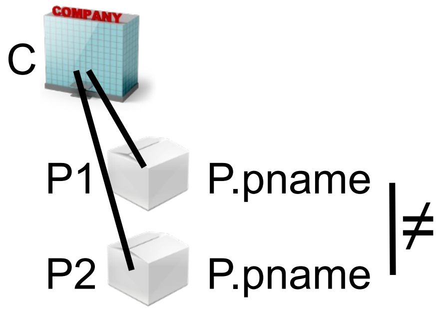
```
SELECT exists
  (
          ?
                  )
```

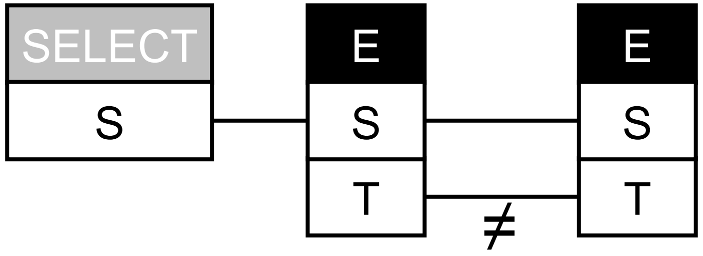
Now in SQL

E(S,T) 501

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]\}$

```sql
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
```



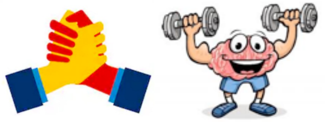


- "each node has at least two distinct neighbors" (statement = Boolean query)
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]$
  - $\neg(\exists x\, \neg(\exists y \exists z [E(x,y) \wedge E(x,z) \wedge y \neq z]))$

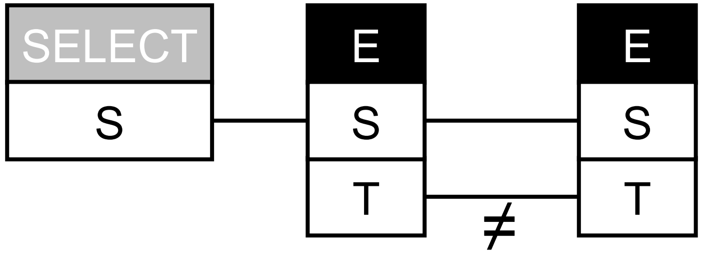
```sql
SELECT exists
  (

           ?

                 )
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Now in SQL

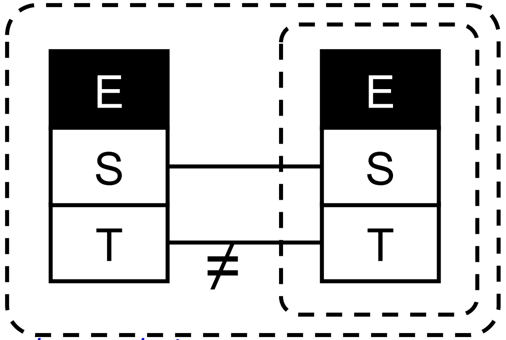
E(S,T)  501

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]\}$




```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
```

- "each node has at least two distinct neighbors"
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]$
  - $\neg(\exists x\, \neg(\exists y \exists z [E(x,y) \wedge E(x,z) \wedge y \neq z]))$



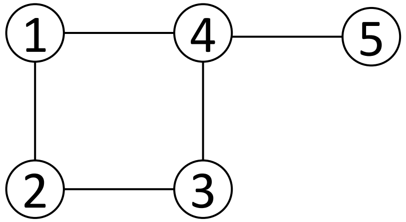

```
SELECT not exists
  (SELECT *
  FROM E E1
  WHERE not exists
    (SELECT *
    FROM E E2
    WHERE E1.S = E2.S
    AND E1.T <> E2.T))
```

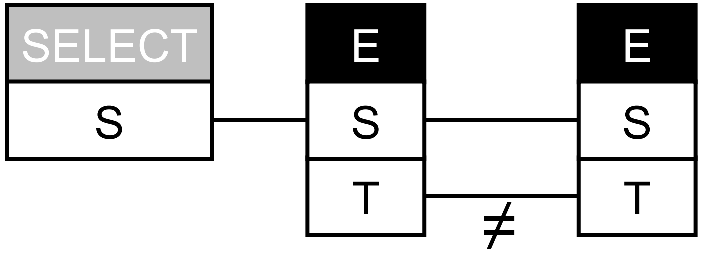
Now in SQL

what do the queries return over the shown "graph database" instance

E(S,T) 501

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]\}$

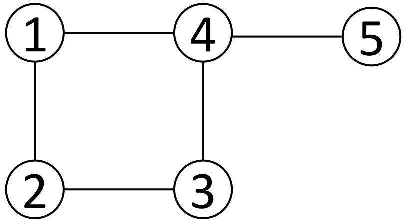


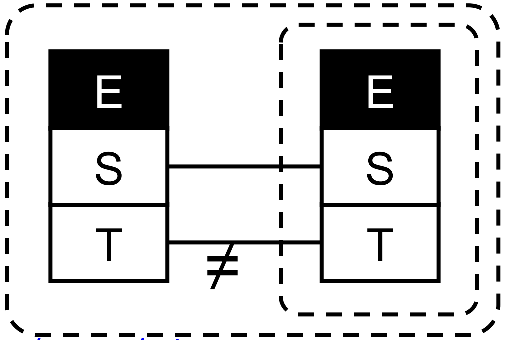



```
SELECT DISTINCT E1.S
FROM    E E1, E E2
WHERE   E1.S = E2.S
AND     E1.T <> E2.T
```

?

- "each node has at least two distinct neighbors"
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]$
  - $\neg(\exists x\, \neg(\exists y \exists z [E(x,y) \wedge E(x,z) \wedge y \neq z]))$






```
SELECT not exists
  (SELECT *
  FROM E E1
  WHERE not exists
    (SELECT *
    FROM E E2
    WHERE E1.S = E2.S
    AND E1.T <> E2.T))
```

?

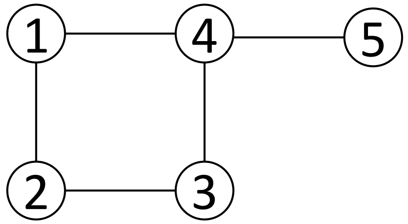
Now in SQL

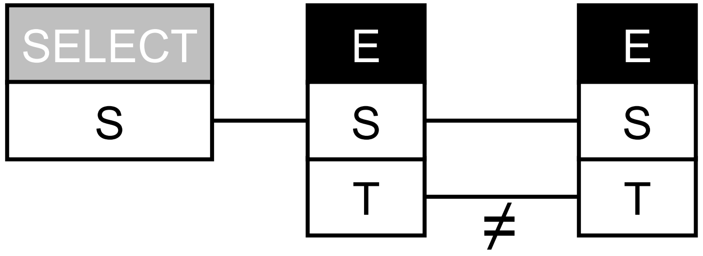
what do the queries return over the shown "graph database" instance

E(S,T)

501

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \land E(x,z) \land y \neq z]\}$

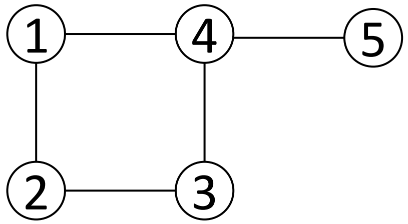


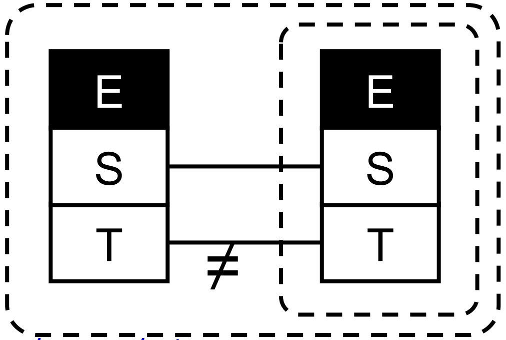



```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
```

{1, 2, 3, 4}

- "each node has at least two distinct neighbors"
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \land E(x,z) \land y \neq z]$
  - $\neg(\exists x\, \neg(\exists y \exists z [E(x,y) \land E(x,z) \land y \neq z]))$






```
SELECT not exists
  (SELECT *
  FROM E E1
  WHERE not exists
    (SELECT *
    FROM E E2
    WHERE E1.S = E2.S
    AND E1.T <> E2.T)) as answer
```

false

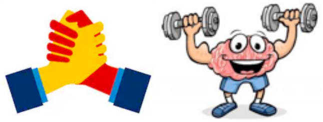
	?column? boolean
1	false

	answer boolean
1	false

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

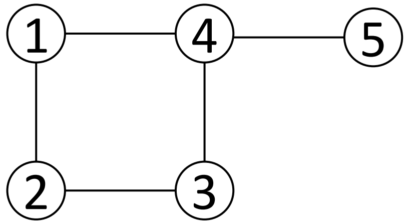
Now in SQL

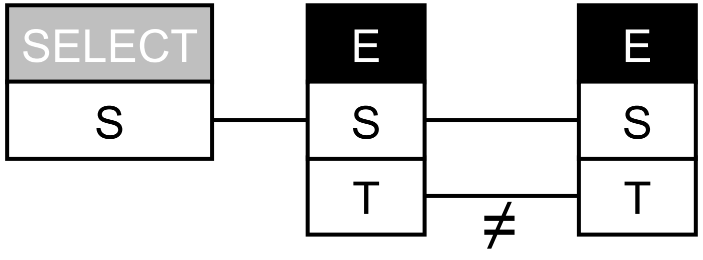
What is a minimal change to the two queries to evaluate them only over nodes 1-4 ?

E(S,T) 

501

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]\}$

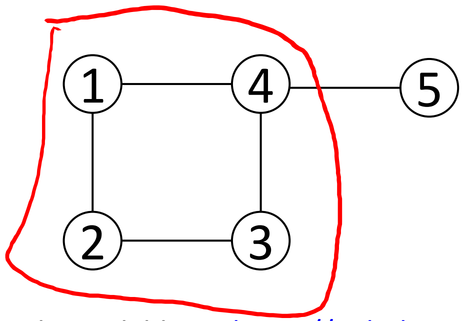


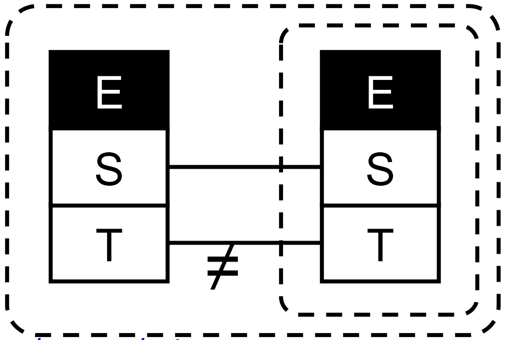



```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
```

{1, 2, 3, 4}

- "each node has at least two distinct neighbors"
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]$
  - $\neg(\exists x\, \neg(\exists y \exists z [E(x,y) \wedge E(x,z) \wedge y \neq z]))$






```
SELECT not exists
  (SELECT *
  FROM E E1
  WHERE not exists
    (SELECT *
    FROM E E2
    WHERE E1.S = E2.S
    AND E1.T <> E2.T))
```

false

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

Now in SQL

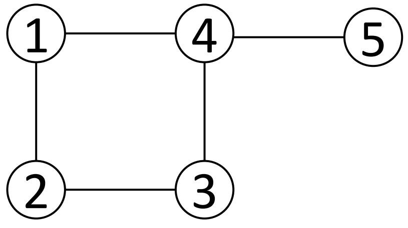
A minimal change to the two queries to evaluate them only over nodes 1-4:

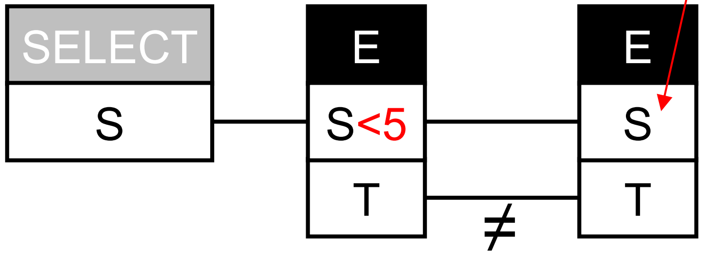
E(S,T)

501

still allows x=5 in the neighbors

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \land E(x,z) \land y \neq z] \land x < 5\}$

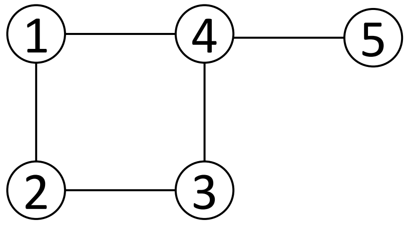


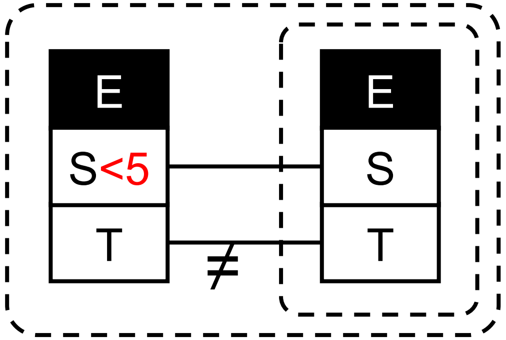



```
SELECT DISTINCT E1.S
FROM     E E1, E E2
WHERE    E1.S = E2.S
AND      E1.T <> E2.T
AND      E1.S<5
```

{1, 2, 3, 4}

- "each node has at least two distinct neighbors"
  - $\forall x\, [x < 5 \Rightarrow \exists y\, \exists z\, [E(x,y) \land E(x,z) \land y \neq z]]$
  - $\neg(\exists x\, [x < 5 \land \neg(\exists y \exists z [E(x,y) \land E(x,z) \land y \neq z]]))$






```
SELECT not exists
  (SELECT *
  FROM E E1
  WHERE E1.S<5
  AND not exists
    (SELECT *
    FROM E E2
    WHERE E1.S = E2.S
    AND E1.T <> E2.T))
```

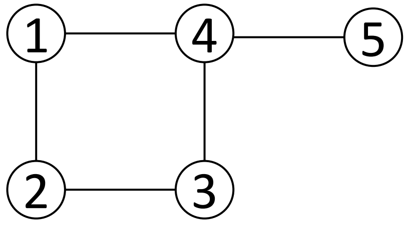
true

Now in SQL with grouping

E(S,T)

501

- "Find nodes that have at least two distinct neighbors" (query)

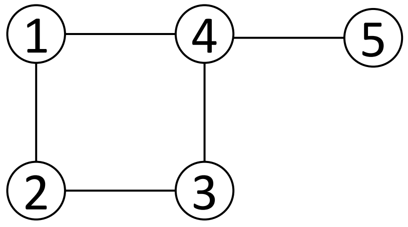


?

```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
```

{1, 2, 3, 4}

- "each node has at least two distinct neighbors"



?

```
SELECT not exists
  (SELECT *
  FROM E E1
  WHERE not exists
    (SELECT *
    FROM E E2
    WHERE E1.S = E2.S
    AND E1.T <> E2.T))
```

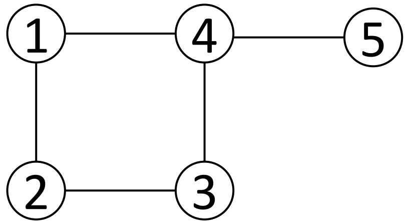
false

Now in SQL with grouping

E(S,T)

501

- "Find nodes that have at least two distinct neighbors" (query)

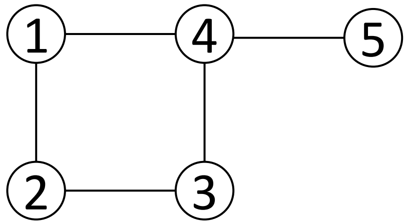



```
SELECT DISTINCT S
FROM   E
GROUP BY S
HAVING COUNT(T)>=2
```

```
SELECT DISTINCT E1.S
FROM    E E1, E E2
WHERE   E1.S = E2.S
AND     E1.T <> E2.T
```

{1, 2, 3, 4}

- "each node has at least two distinct neighbors"



?

```
SELECT not exists
  (SELECT *
  FROM E E1
  WHERE not exists
    (SELECT *
    FROM E E2
    WHERE E1.S = E2.S
    AND E1.T <> E2.T))
```

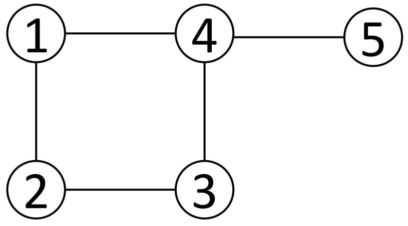
false

Now in SQL with grouping

E(S,T)

501

- "Find nodes that have at least two distinct neighbors" (query)

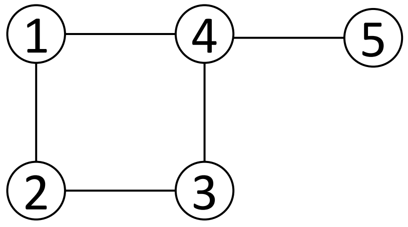



```
SELECT DISTINCT S
FROM   E
GROUP BY S
HAVING COUNT(T)>=2
```

```
SELECT DISTINCT E1.S
FROM    E E1, E E2
WHERE   E1.S = E2.S
AND     E1.T <> E2.T
```

{1, 2, 3, 4}

- "each node has at least two distinct neighbors"




```
SELECT not exists
  (SELECT S
  FROM E
  GROUP BY S
  HAVING COUNT(T)=1)
```

```
SELECT not exists
  (SELECT *
  FROM E E1
  WHERE not exists
    (SELECT *
    FROM E E2
    WHERE E1.S = E2.S
    AND E1.T <> E2.T))
```

false

More practice

- "A small, happy dog is at home"

?

- "Every small dog that is at home is happy."

?

- "Jiahui owns a small, happy dog"

?

- "Jiahui owns every small, happy dog."

?

Exercise for next class ☺

Example adopted from Barker-Plummer, Barwise, Etchemendy - Language, Proof, And Logic (book, 2nd ed), 2011. https://www.gradegrinder.net/

One more example

- "There are infinitely many prime numbers"

?

Exercise for
next class ☺

Source: Vasco Brattka. Logic and computation (lecture notes), 2007. http://cca-net.de/vasco/lc/
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Semantics of First-Order Logic on Graphs

E(x,y)
Parent('Alice','Bob')

Semantics:

- First-order variables range over (can be " bound to") elements of the universes of structures
- To evaluate a formula φ, we need a graph G and a binding α that maps the free variables of φ to nodes of G
  - Notation: $G \vDash_{\alpha} \varphi(x_1, \ldots, x_k)$

Fundamental Distinction: Syntax vs. semantics (Tarski, 1930)

- Syntax: grammar, how to construct correct sentence, the combinatorics of units of a language (e.g. "This water is triangular.")
- Semantics: relates to meaning

Source: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

Relational Databases

Codd's Two Fundamental Ideas:

- Tables are relations: a row in a table is just a tuple in a relation; order of rows/tuples does not matter!

- Formulas are queries: they specify the **What** rather then the **How**! That's declarative programming

Source: Moshe Vardi: Database Queries: Logic and Complexity, talk 2012. https://abiteboul.com/College/1.140312.Vardi.pdf

3 Components of FOL

1. Syntax (or language)

?

2. Interpretation

?

3. Semantics

?

3 Components of FOL

1. Syntax (or language)
   - *What are the allowed syntactic expressions?*

2. Interpretation
   - *Mapping symbols to an actual world*

3. Semantics
   - *When is a statement "true" under some interpretation?*

3 Components of FOL

1. Syntax (or language)
   - *What are the allowed syntactic expressions?*
   - For DB's: ?
2. Interpretation
   - *Mapping symbols to an actual world*
   - For DB's: ?
3. Semantics
   - *When is a statement "true" under some interpretation?*
   - For DB's: ?

3 Components of FOL

1. Syntax (or language)
   - *What are the allowed syntactic expressions?*
   - For DB's: schema, constraints, query language
2. Interpretation
   - *Mapping symbols to an actual world*
   - For DB's: database
3. Semantics
   - *When is a statement “true” under some interpretation?*
   - For DB's: meaning of integrity constraints and query results (recall the conceptual evaluation strategy of SQL)

Components of FOL: (1) Syntax = First-order language

vocabulary

- Alphabet: symbols in use
  - Variables, constants, **function** symbols, **predicate** symbols, connectives, quantifiers, punctuation symbols

x Alice MotherOf(x) Friends(x,y) $\wedge$ $\exists$ ( )

terms

relation b/w objects

- Term: expression that stands for an element or object
  - Variable, constant
  - Inductively $f(t_1,\ldots,t_n)$ where t_i are terms, f a function symbol

MotherOf(MotherOf(x))

- (Well-formed) formula: parameterized statement

x = 'Alice'

  - Atom $p(t_1,\ldots,t_n)$ where p is a predicate symbol, t_i terms (atomic formula, together with predicates $t_1=t_2$)
  - Inductively, for formulas F, G, variable x:

$$F \wedge G \quad F \vee G \quad \neg F \quad F \longrightarrow G \quad F \longleftrightarrow G \quad \forall x\, F \quad \exists x\, F$$

- A first-order language refers to the set of all formulas over an alphabet

Components of FOL: (2) Interpretation

- How to assign meaning to the symbols of a formal language
- An interpretation INT for an alphabet consists of:
  - A non-empty set **Dom**, called domain
    - {Alice, Bob, Charly}
  - An assignment of an element in **Dom** to each constant symbol
    - Alice (recall we often write constants with quotation marks 'Alice')
  - An assignment of a function **Dom**^{n} ⟶**Dom** to each n-ary function symbol
    - Alice = MotherOf(Bob)
  - An assignment of a function **Dom**^{n} ⟶{true, false} (i.e., a relation) to each n-ary predicate symbol
    - Friends(Bob, Charly) = TRUE

Components of FOL: (3) Semantics

- A variable assignment V to a formula in an interpretation INT assigns to each free variable X a value from **Dom**
  - Recall, a free variable is one that is not quantified

Person(X) ∃Y Married(X,Y)

bound

- Truth value for formula F under interpretation INT and variable assignment V:
  - Atom $p(t_1,\ldots,t_n)$: $q(s_1,\ldots,s_n)$ where q is the interpretation of the predicate p and s_i the interpretation of t_i
  - $F \wedge G \quad F \vee G \quad \neg F \quad F \longrightarrow G \quad F \longleftrightarrow G$: according to truth table
  - $\exists X F$: true iff there exists d∈**Dom** such that if V assigns d to X then the truth value of F is true; otherwise false
  - $\forall X F$: true iff for all d∈**Dom**, if V assigns d to X then the truth value of F is true; otherwise false
- If a formula has no free vars (closed formula or sentence), we can simply refer to its truth value under INT

∀X: Person(X) ⟶ Mortal(X)

Formula — Sentence

Formula — Query

Operator precedence

Operator precedence is an ordering of logical operators designed to allow the dropping of parentheses in logical expressions. The following table gives a hierarchy of precedences for the operators of propositional logic. The ¬ operator has higher precedence than ∧; ∧ has higher precedence than ∨; and ∨ has higher precedence than ⇒ and ⇔.

$$\begin{array}{c} \neg \\ \wedge \\ \vee \\ \Rightarrow \ \Leftrightarrow \end{array}$$

In unparenthesized sentences, it is often the case that an expression is flanked by operators, one on either side. In interpreting such sentences, the question is whether the expression associates with the operator on its left or the one on its right. We can use precedence to make this determination. In particular, we agree that an operand in such a situation always associates with the operator of higher precedence. When an operand is surrounded by operators of equal precedence, the operand associates to the right. The following examples show how these rules work in various cases. The expressions on the right are the fully parenthesized versions of the expressions on the left.

$$\begin{array}{ll} \neg p \wedge q & ((\neg p) \wedge q) \\ p \wedge \neg q & (p \wedge (\neg q)) \\ p \wedge q \vee r & ((p \wedge q) \vee r) \\ p \vee q \wedge r & (p \vee (q \wedge r)) \\ p \Rightarrow q \Rightarrow r & (p \Rightarrow (q \Rightarrow r)) \\ p \Rightarrow q \Leftrightarrow r & (p \Rightarrow (q \Leftrightarrow r)) \end{array}$$

Source: http://intrologic.stanford.edu/glossary/operator_precedence.html

Queries and the connection to logic

- Why logic?
- A crash course in FOL
- Relational Calculus (RC)
  - Syntax and Semantics
  - Domain RC (DRC) vs Tuple RC (TRC)
  - Domain Independence and Safety
- 4 categorical propositions

The entire story of Relational Calculus (RC) in 1 slide

1. RC = FOL over DB's
2. RC can express "bad queries" that depend not only on the DB, but also on the domain from which values are taken (called "domain dependence" which is bad)
3. We cannot test whether an RC query is "good," but we can use a "good" subset of RC that captures all "good" queries (safety)
4. "Good" RC and RA can express the same queries! (equivalence = Codd's theorem)

- RC is, essentially, first-order logic (FOL) over the schema relations
  - A query has the form "find all tuples $(x_1,\ldots,x_k)$ that satisfy an FOL condition"
  - Thus RC is a declarative query language: a query is not defined by a sequence of operations, but rather by a logical condition that the result should satisfy

Queries and the connection to logic

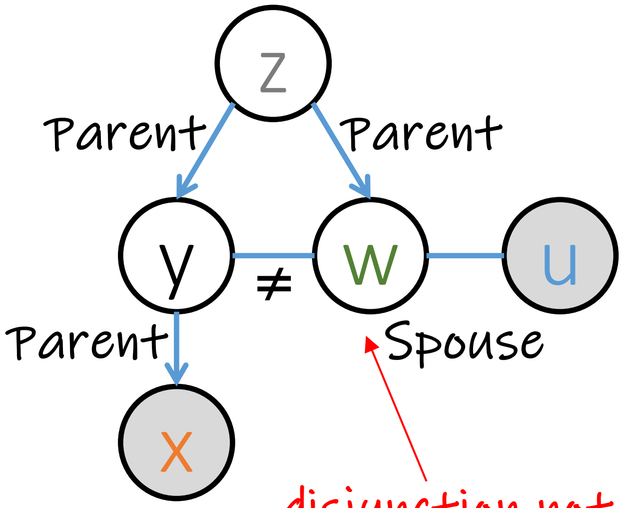
- Why logic?
- A crash course in FOL
- Relational Calculus (RC)
  - Syntax and Semantics
  - Domain RC (DRC) vs Tuple RC (TRC)
  - Domain Independence and Safety
- 4 categorical propositions

RC Query

Person(id, gender, country)
Parent(parent, child)
Spouse(person1, person2)

assume symmetric relation
$(a,b) \in \text{Spouse} \Leftrightarrow (b,a) \in \text{Spouse}$

$$\{ (x,u) \mid \text{Person}(u, \text{'female'}, \text{'Canada'}) \wedge \exists z,y \, [\text{Parent}(z,y) \wedge \text{Parent}(y,x) \wedge \exists w \, [\text{Parent}(z,w) \wedge y \neq w \wedge (u=w \vee \text{Spouse}(u,w))]] \}$$



Which relatives does this query find?

?

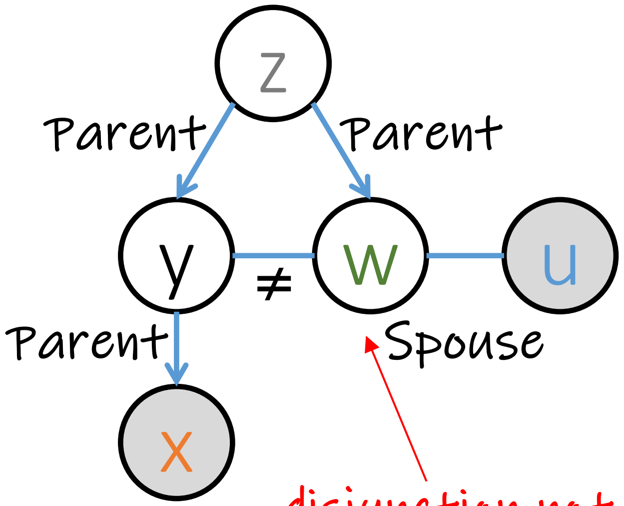
disjunction not shown here (difficult to visualize)

RC Query

Person(id, gender, country)
Parent(parent, child)
Spouse(person1, person2)

assume symmetric relation
$(a,b)\in Spouse \Leftrightarrow (b,a)\in Spouse$

$$\{\, (x,u) \mid \text{Person}(u, \text{'female'}, \text{'Canada'}) \land \\ \exists z,y\, \big[\text{Parent}(z,y) \land \text{Parent}(y,x) \land \\ \exists w\, [\text{Parent}(z,w) \land y \neq w \land (u=w \lor \text{Spouse}(u,w))]\,\big] \,\}$$



disjunction not shown here (difficult to visualize)

Which relatives does this query find?

Persons and their Canadian aunts
(incl. female spouses of uncles and aunts)

RC Symbols (more precisely “Domain RC” = DRC)

- Constant values: ?
  - Values that may appear in table cells (optionally with quotation marks)
- Variables: ?
  - Range over the values that may appear in table cells
- Relation symbols: ?
  - Each with a specified arity (fixed by the given relational schema)

RC Symbols (more precisely “Domain RC” = DRC)

- Constant values: 'female', 'Canada'
  - Values that may appear in table cells (optionally with quotation marks)
- Variables: x, y, z, w, u
  - Range over the values that may appear in table cells
- Relation symbols: Person, Parent, Spouse
  - Each with a specified arity (fixed by the given relational schema)
  - Two variants:
    - No attribute names, only attribute positions: “unnamed perspective”
    - Attribute names: “named perspective”
- What about functions ?

RC Symbols (more precisely "Domain RC" = DRC)

- Constant values: 'female', 'Canada'
  - Values that may appear in table cells (optionally with quotation marks)
- Variables: x, y, z, w, u
  - Range over the values that may appear in table cells
- Relation symbols: Person, Parent, Spouse
  - Each with a specified arity (fixed by the given relational schema)
  - Two variants:
    - No attribute names, only attribute positions: "unnamed perspective"
    - Attribute names: "named perspective"
- Unlike general FOL, no function symbols!

Updated 1/26/2024

Topic 1: Data models and query languages
Unit 2: Logic & relational calculus
Lecture 5

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

1/26/2024

Pre-class conversations

- Last class recapitulation
- Scribes: perfect example for first iteration posted to Piazza. Thanks!

- today:
  - logic continued (likely next time algebra and the connection)
  - logic is super important for our class; thus lots of practice today ☺
  - in particular the concept of "undecidability": intuition for why things can quickly get complicated without giving proofs

order. For example, when setting goals, just set goals. Don't think about how you will achieve them or what you will do if something goes wrong. When you are diagnosing problems, don't think about how you will solve them—just diagnose them. Blurring the steps leads to suboptimal outcomes because it interferes with uncovering the true problems. The process is iterative: Doing each step thoroughly will provide you with the information you need to move on to the next step and do it well.

a. **Focus on the "what is" before deciding "what to do about it."** It is a common mistake to move in a nanosecond from identifying a tough problem to proposing a solution for it. Strategic thinking requires both diagnosis and design. A good diagnosis typically takes between fifteen minutes and an hour, depending on how well it's done and how complex the issue is. It involves speaking with the relevant people and looking at the evidence together to determine the root causes. Like principles, root causes manifest themselves over and over again in seemingly different situations. Finding them and dealing with them pays dividends again and again.

f. **Recognize that it doesn't take a lot of time to design a good plan.** A plan can be sketched out and refined in just hours or spread out over days or weeks. But the process is essential because it determines what you will have to do to be effective. Too many people make the mistake of spending virtually no time on designing because they are preoccupied with execution. Remember: Designing precedes doing!

b. **Good work habits are vastly underrated.** People who push through successfully have to-do lists that are reasonably prioritized, and they make certain each item is ticked off in order.

Separation of concerns: WHAT from HOW

Source: Ray Dalio. "Principles", 2017. https://en.wikipedia.org/wiki/Principles_(book)

More practice

- "A small, happy dog is at home"

?

- "Every small dog that is at home is happy."

?

- "Jiahui owns a small, happy dog"

?

- "Jiahui owns every small, happy dog."

?

More practice

$\exists x \in \text{Dogs} [\ldots]$

- "A small, happy dog is at home"
  - ∃x [(Small(x) ∧ Happy (x) ∧ Dog (x)) ∧ Home(x)]

associativity of conjunction: no need of evaluation to follow blue parentheses

- "Every small dog that is at home is happy."

?

- "Jiahui owns a small, happy dog"

?

- "Jiahui owns every small, happy dog."

?

More practice

- "A small, happy dog is at home"
  - $\exists x\ [(\text{Small}(x) \wedge \text{Happy}(x) \wedge \text{Dog}(x)) \wedge \text{Home}(x)]$

associativity of conjunction: no need of evaluation to follow blue parentheses

- "Every small dog that is at home is happy."
  - $\forall x\ [(\text{Small}(x) \wedge \text{Dog}(x) \wedge \text{Home}(x)) \rightarrow \text{Happy}(x)]$

here evaluation needs to follow blue parentheses

- "Jiahui owns a small, happy dog"

?

- "Jiahui owns every small, happy dog."

?

More practice

- "A small, happy dog is at home"
  - ∃x [(Small(x) ∧ Happy (x) ∧ Dog (x)) ∧ Home(x)]

associativity of conjunction: no need of evaluation to follow blue parentheses

- "Every small dog that is at home is happy."
  - ∀x [(Small(x) ∧ Dog (x) ∧ Home(x)) → Happy (x)]

here evaluation needs to follow blue parentheses

- "Jiahui owns a small, happy dog"
  - ∃x [(Small(x) ∧ Happy (x) ∧ Dog (x)) ∧ Owns('Jiahui', x)]
- "Jiahui owns every small, happy dog."

?

notice that we deviate here from the usual notation in logics of constants like 'Jiahui' written w/o quotation marks

Example adopted from Barker-Plummer, Barwise, Etchemendy - Language, Proof, And Logic (book, 2nd ed), 2011. https://www.gradegrinder.net/

More practice

- "A small, happy dog is at home"
  - $\exists x\, [(\text{Small}(x) \wedge \text{Happy}(x) \wedge \text{Dog}(x)) \wedge \text{Home}(x)]$

associativity of conjunction: no need of evaluation to follow blue parentheses

- "Every small dog that is at home is happy."
  - $\forall x\, [(\text{Small}(x) \wedge \text{Dog}(x) \wedge \text{Home}(x)) \rightarrow \text{Happy}(x)]$

here evaluation needs to follow blue parentheses

- "Jiahui owns a small, happy dog"
  - $\exists x\, [(\text{Small}(x) \wedge \text{Happy}(x) \wedge \text{Dog}(x)) \wedge \text{Owns}(\text{'Jiahui'}, x)]$
- "Jiahui owns every small, happy dog."
  - $\forall x\, [(\text{Small}(x) \wedge \text{Happy}(x) \wedge \text{Dog}(x)) \rightarrow \text{Owns}(\text{'Jiahui'}, x)]$

notice that we deviate here from the usual notation in logics of constants like 'Jiahui' written w/o quotation marks

Example adopted from Barker-Plummer, Barwise, Etchemendy - Language, Proof, And Logic (book, 2nd ed), 2011. https://www.gradegrinder.net/

Two more examples

- "There are infinitely many prime numbers"

?

Two more examples

- "There are infinitely many prime numbers"
  - $\forall x\ \exists y\ [y > x \wedge \text{Prime}(y)]$

Source first example: Vasco Brattka. Logic and computation (lecture notes), 2007. http://cca-net.de/vasco/lc/

Two more examples

- "There are infinitely many prime numbers"
  - $\forall x\ \exists y\ [y > x \land \text{Prime}(y)]$

- $\forall x\ \exists y\ [y = \text{sqrt}(x)]$

?

Source first example: Vasco Brattka. Logic and computation (lecture notes), 2007. http://cca-net.de/vasco/lc/

Two more examples

- "There are infinitely many prime numbers"
  - $\forall x\, \exists y\, [y > x \wedge \text{Prime}(y)]$

- $\forall x\, \exists y\, [y = \text{sqrt}(x)]$
  - Truth of this expression depends on domain:
    - evaluates to false if x and y have the domain of the real numbers $\mathbb{R}$
    - evaluates to true if their domain is the complex numbers $\mathbb{C}$

Source first example: Vasco Brattka. Logic and computation (lecture notes), 2007. http://cca-net.de/vasco/lc/

Queries and the connection to logic

- Why logic?
- A crash course in FOL
- Relational Calculus (RC)
  - Syntax and Semantics
  - Domain RC (DRC) vs Tuple RC (TRC)
  - Domain Independence and Safety
- 4 categorical propositions

RC Formulas (atomic and non-atomic)

- Atomic formulas:
  - $R(t_1,...,t_k)$ Person(u, 'female', 'Canada')
    - R is a k-ary relation, Each t_i is a variable or a constant
    - Semantically it states that $(t_1,...,t_k)$ is a tuple in R
  - x op u   u=w, y≠w, z>5, z='female'
    - x is a variable, u is a variable/constant, op is one of >, <, =, ≠
    - Simply binary predicates, predefined interpretation
- Formula:
  - Atomic formula
  - If ϕ and ψ are formulas then these are formulas:

    $\phi \wedge \psi$ $\phi \vee \psi$ $\phi \rightarrow \psi$ $\phi \rightarrow \psi$ $\neg\phi$ $\exists x\, \phi$ $\forall x\, \phi$

Person(u, 'female', 'Canada') ∧
∃z,y [Parent(z,y) ∧ Parent(y,x)]

Free Variables

- Intuitively: free variable are not bound to quantifiers
- Formally:
  - A free variable of an atomic formula is a variable that occurs in the atomic formula
  - A free variable of $\varphi \land \psi, \varphi \lor \psi, \varphi \rightarrow \psi$ is a free variable of either φ or ψ
  - A free variable of $\neg\varphi$ is a free variable of φ
  - A free variable of $\exists x\ \varphi$ and $\forall x\ \varphi$ is a free variable y of φ such that $y \neq x$
- We write $\varphi(x_1,\ldots,x_k)$ to state that $x_1,\ldots,x_k$ are the free variables of formula φ (in some order)

$x < y$

$<$

x	y
1	2
1	3
2	3

$\exists x\ [x=y]$
... is y free ?

Free Variables

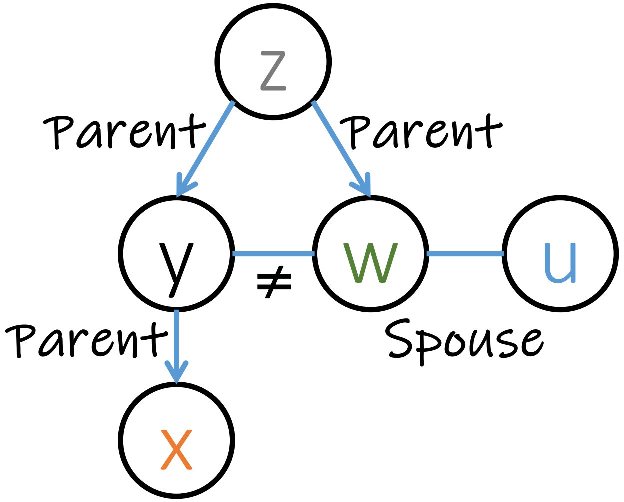
- Intuitively: free variable are not bound to quantifiers
- Formally:
  - A free variable of an atomic formula is a variable that occurs in the atomic formula
  - A free variable of $\varphi \land \psi, \varphi \lor \psi, \varphi \rightarrow \psi$ is a free variable of either φ or ψ
  - A free variable of $\neg\varphi$ is a free variable of φ
  - A free variable of $\exists x\ \varphi$ and $\forall x\ \varphi$ is a free variable y of φ such that $y \neq x$
- We write $\varphi(x_1,\ldots,x_k)$ to state that $x_1,\ldots,x_k$ are the free variables of formula φ (in some order)

$\exists x\ [x=y]$
y is free

Back to our earlier example

This is a formula!

$$\text{Person}(u, \text{'female'}, \text{'Canada'}) \wedge$$
$$\exists z,y \left[\text{Parent}(z,y) \wedge \text{Parent}(y,x) \wedge\right.$$
$$\left.\exists w \left[\text{Parent}(z,w) \wedge y \neq w \wedge (u=w \vee \text{Spouse}(u,w))\right]\right]$$

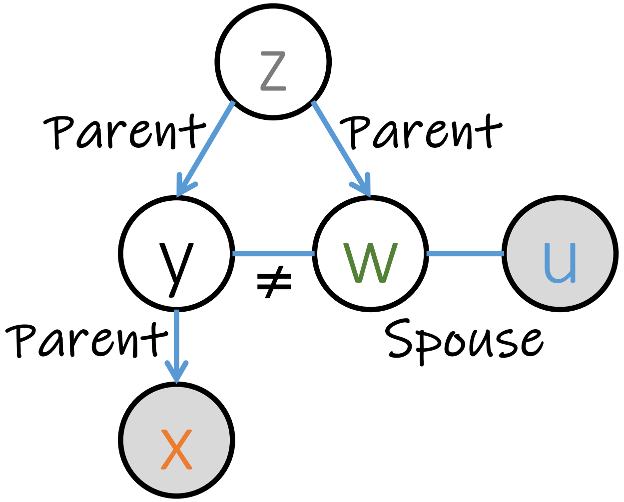


What are the free variables?

?

Back to our earlier example

$$\text{Person}(u, \text{'female'}, \text{'Canada'}) \wedge$$
$$\exists z,y\ \Big[\text{Parent}(z,y) \wedge \text{Parent}(y,x) \wedge$$
$$\exists w\ [\text{Parent}(z,w) \wedge y \neq w \wedge (u=w \vee \text{Spouse}(u,w))]\Big]$$

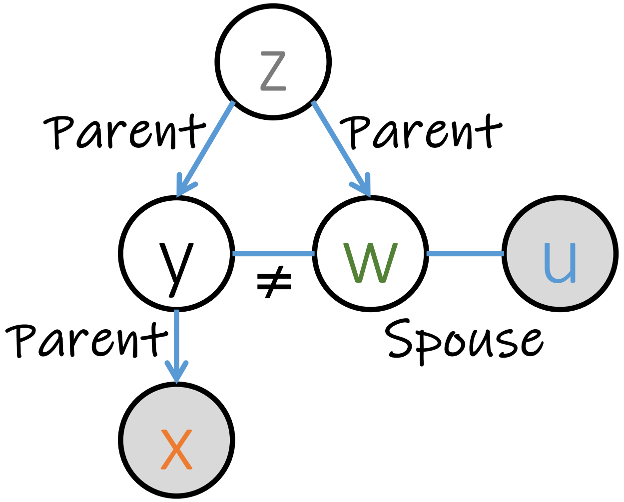


Notation:

$\varphi(x,u)$ or CanadianAunt(u,x)

RC query

$$\{ (x,u) \mid \text{Person}(u, \text{'female'}, \text{'Canada'}) \land \exists z,y\, [\text{Parent}(z,y) \land \text{Parent}(y,x) \land \exists w\, [\text{Parent}(z,w) \land y \neq w \land (u=w \lor \text{Spouse}(u,w))]] \}$$



$$\{ (x_1,\ldots,x_k) \mid \varphi(x_1,\ldots,x_k) \}$$

$\varphi(x,u)$ or CanadianAunt(u,x)

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Relation Calculus Query

some condition on the variables
$COND(x_1,\ldots,x_k)$

- An RC query is an expression of the form

$\{ (x,y) \mid x < y \}$

$$\{ (x_1,\ldots,x_k) \mid \varphi(x_1,\ldots,x_k) \}$$

where $\varphi(x_1,\ldots,x_k)$ is an RC formula

- An RC query is *over* a relational schema $\mathbf{S}$ if all the relation symbols belong to $\mathbf{S}$ (with matching arities)

Queries and the connection to logic

- Why logic?
- A crash course in FOL
- Relational Calculus (RC)
  - Syntax and Semantics
  - Domain RC (DRC) vs Tuple RC (TRC)
  - Domain Independence and Safety
- 4 categorical propositions

DRC vs. TRC (Domain vs. Tuple RC)

Two common variants of RC:

- DRC (Domain RC): attributes as sets (what we have seen so far)
  - DRC applies typical FO: terms interpreted as attribute (domain) values, relations have arity but no attribute names (= unnamed perspective)
  - Example: $x = 4 \land R(x, y)$
- TRC (Tuple RC): tuples as sets
  - TRC is more "database friendly": terms interpreted as tuples with named attributes
  - Example: $R.A = 4$ for schema $R(A, B)$
- There are easy conversions between the two formalisms

DRC vs. TRC (Domain vs. Tuple RC)

R

A	B
1	3
1	4
2	2

domain variables range over the domain

DRC $\{ (x,y) \mid R(x,y) \land y>2 \}$

predicate (unnamed)

TRC $\{ r \mid r \in R \land r.B>2 \}$

$\{ r \mid r \in R[r.B>2] \}$

predicate (named)

tuple variables range over relations
(domain of tuple variable)

DRC $\{ (x) \mid \exists y[R(x,y) \land y>2] \}$

TRC ?

DRC vs. TRC (Domain vs. Tuple RC)

R

A	B
1	3
1	4
2	2

domain variables range over the domain

DRC $\{ (x,y) \mid R(x,y) \wedge y>2 \}$

predicate (unnamed)

TRC $\{ r \mid r \in R \wedge r.B>2 \}$

$\{ r \mid r \in R[r.B>2] \}$

predicate (named)

tuple variables range over relations
(domain of tuple variable)

DRC $\{ (x) \mid \exists y[R(x,y) \wedge y>2] \}$

TRC $\{ q \mid \exists r \in R[q.A=r.A \wedge r.B>2] \}$

Which are here bound and which are free variables ?

Other sources often use "t" as tuple variable. I prefer to use "q" to identify the output relation with the query

DRC vs. TRC (Domain vs. Tuple RC)

domain variables range over the domain

R	
A	B
1	3
1	4
2	2

DRC $\{ (x,y) \mid R(x,y) \wedge y>2 \}$

predicate (unnamed)

TRC $\{ r \mid r \in R \wedge r.B>2 \}$

$\{ r \mid r \in R[r.B>2]\}$

predicate (named)

$\{ q \mid \exists r \in R[q.A=r.A \wedge q.B=r.B \wedge r.B>2] \}$

$\{ q(A,B) \mid \exists r \in R[q.A=r.A \wedge q.B=r.B \wedge r.B>2] \}$

tuple variables range over relations
(domain of tuple variable)

free bound

DRC $\{ (x) \mid \exists y[R(x,y) \wedge y>2] \}$

TRC $\{ q \mid \exists r \in R[q.A=r.A \wedge r.B>2] \}$

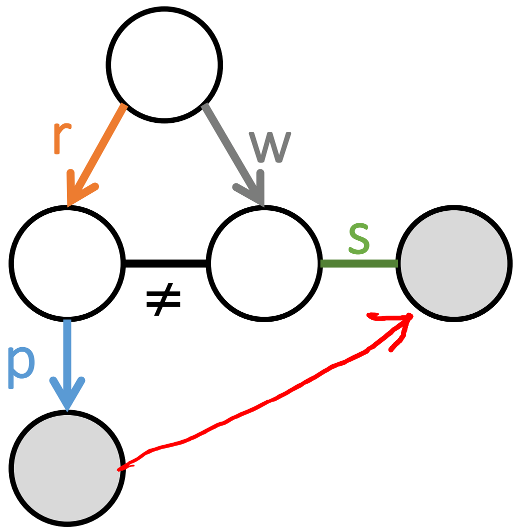
free bound

Our Example in TRC

Person(id, gender, country)
Parent(parent, child)
Spouse(person1, person2)

optionally "q(nephew, aunt)"

{ **q** | ∃a ∈ Person [a.gender = 'female' ∧ a.country = 'Canada'] ∧
∃p,r,w ∈ Parent [p.child = **q.nephew** ∧ r.child = p.parent ∧
w.parent = r.parent ∧ w.child ≠ r.child ∧ a.id = **q.aunt** ∧
(w.child = a.id ∨ ∃s [s ∈ Spouse ∧ s.person1 = w.child ∧ s.person2 = a.id])]]}



tuple variables like in SQL instead of domain variables: { q | COND(q)}

often used short forms:

∀x∈R[φ]	same as	∀x[x∈R ⇒ φ]
∃x∈R[φ]	same as	∃x[x∈R ∧ φ]

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018. However, notice I prefer and follow here the notation of [Ramakrishnan, Gehrke' 03] and [Elmasri, Navathe'15] of using **a.country = 'Canada'**, instead of the alternative notation **a[country]='Canada'** used by [Silberschatz, Korth, Sudarshan 2010]

Different TRC notations

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

331

Find persons who frequent <u>only</u> bars that serve <u>some</u> drinks they like.
(Find persons for whom there does not exist a bar they frequent that serves no drink they do not like.)

{q(person) | ∃f ∈ Frequents [f.person=q.person ∧ ¬(∃f2 ∈ Frequents [f2.person=f.person ∧
¬(∃l ∈ Likes, ∃s ∈ Serves [l.drink=s.drink ∧ f2.bar=s.bar ∧ f2.person=l.person])])]} *my preferred notation*

{Q.person | ∃F ∈ Frequents.(Q.person=F.person ∧ (∄F2 ∈ Frequents.(F2.person=F.person ∧
(∄L ∈ Likes, ∄S ∈ Serves.(L.drink=S.drink ∧ F2.bar=S.bar ∧ F2.person=L.person)))))} *my earlier pref. notation*

{t: Person | ∃f ∈ Frequents [t(Person)=f(Person) ∧ ¬∃f2 ∈ Frequents [F2(person)=F(person) ∧
¬(∃l ∈ Likes ∃s ∈ Serves) [l(Drink)=s(Drink) ∧ f2(Bar)=s(Bar) ∧ f2(Person)=l(Person)]]]} *[Deutsch 2019]*

{f.Person | Frequents(f) AND (NOT(∃f2)(Frequents(f2) AND f2.person=f.person ∧
(NOT(∃l)(∃s)(Likes(l) AND Serves(s) AND l.drink=s.drink AND f2.bar=s.bar AND f2.person=l.person))))} *[Elmasri 2015]*

$\{\mu^{(1)} \mid (\exists\rho^{(2)})\,(\text{Frequents}(\rho) \land \rho[1]=\mu[1] \land \neg((\exists\lambda^{(2)})(\text{Frequents}(\lambda) \land \lambda[1]=\rho[1] \land$
$\neg((\exists\nu^{(2)})(\exists\theta^{(2)})(\text{Likes}(\nu) \land \text{Serves}(\theta) \land \nu(2)=\theta(2) \land \lambda(2)=\theta(1) \land \lambda(1)=\nu(1))))))\}$ *[Ullman 1988]*

{P| ∃F ∈ Frequents (F.person=P.person ∧ ¬∃F2 ∈ Frequents(F2.person=F.person ∧
¬(∃L ∈ Likes ∃S ∈ Serves (L.drink=S.drink ∧ F2.bar=S.bar ∧ F2.person=L.person))))} *[Ramakrishnan 2003]*

Deutsch (based on Vianu), CSE132A: Database System Principles, fall 2019. https://cseweb.ucsd.edu/classes/fa19/cse132A-a/slides/relational-calculus.pdf , Elmasri , Navathe. Fundamentals of database systems (7 ed), 2015. https://dl.acm.org/doi/book/10.5555/2842853 , Ullman. Principles of Database and Knowledge-base Systems, Vol. 1, 1988. https://dl.acm.org/doi/book/10.5555/42790 , Ramakrishnan, Gehrke. Database Management Systems (3 ed), 2003. https://dl.acm.org/doi/book/10.5555/560733
SQL database available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

TRC vs. Relational Diagrams

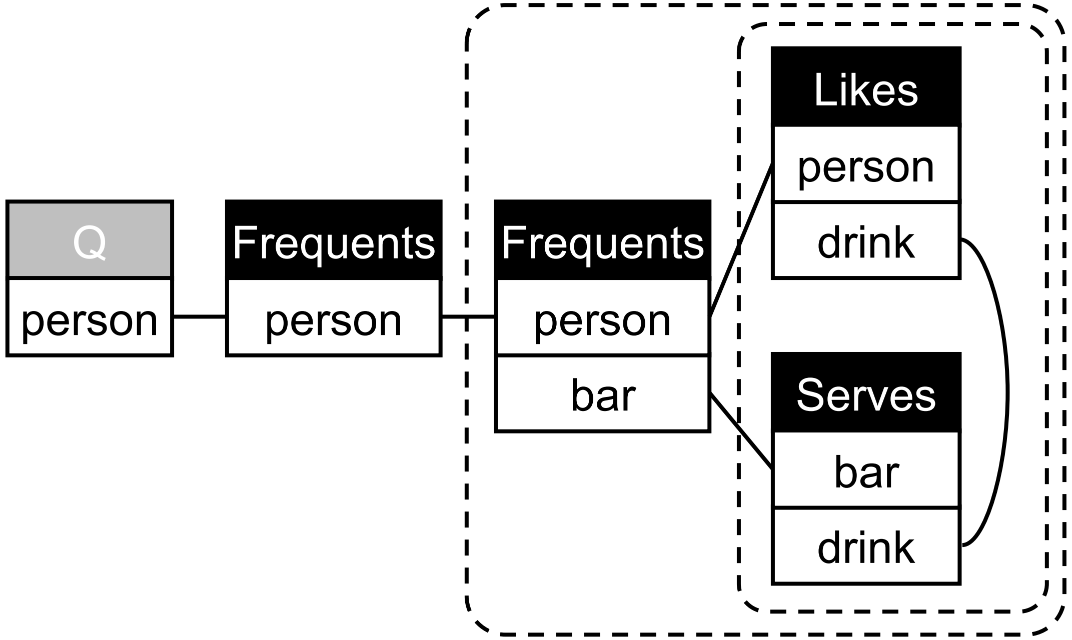
Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink)

331

Find persons who frequent only bars that serve some drinks they like.
(Find persons for whom there does not exist a bar they frequent that serves no drink they do not like.)

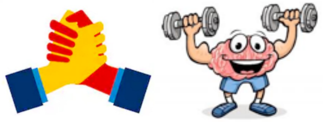
{q(person) | ∃f ∈ Frequents [f.person=q.person ∧ ¬(∃f2 ∈ Frequents [f2.person=f.person ∧ ¬(∃l ∈ Likes, ∃s ∈ Serves [l.drink=s.drink ∧ f2.bar=s.bar ∧ f2.person=l.person])])]}

my preferred notation



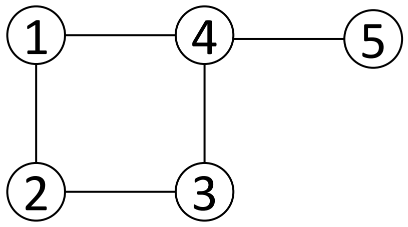

```
SELECT DISTINCT F.person
FROM Frequents F
WHERE not exists
  (SELECT *
  FROM Frequents F2
  WHERE F2.person=F.person
  AND not exists
    (SELECT *
    FROM Likes L, Serves S
    WHERE L.person=F2.person
    AND L.drink=S.drink
    AND S.bar=F2.bar))
```

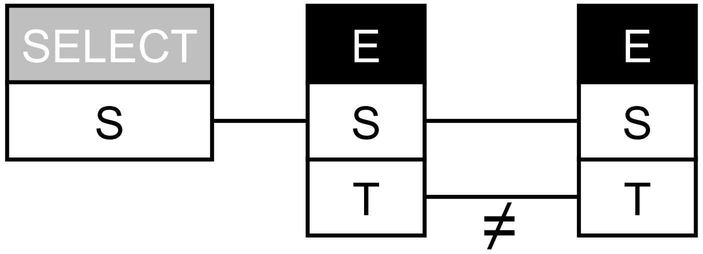
In DRC, SQL, RD, and now in TRC

E(S,T) 

501

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]\}$
  - in TRC ?

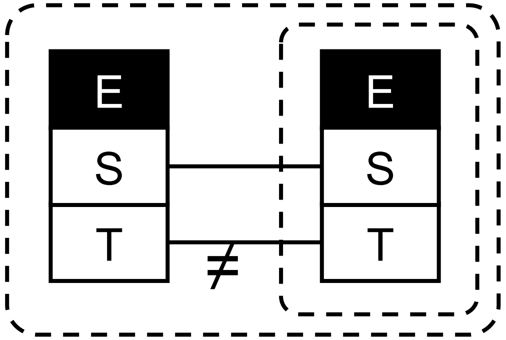





```
SELECT DISTINCT E1.S
FROM     E E1, E E2
WHERE    E1.S = E2.S
AND      E1.T <> E2.T
```

{1, 2, 3, 4}

- "each node has at least two distinct neighbors"
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]$
  - $\neg(\exists x\, \neg(\exists y \exists z[E(x,y) \wedge E(x,z) \wedge y \neq z]))$
  - in TRC ?




```
SELECT not exists
   (SELECT *
   FROM E E1
   WHERE not exists
      (SELECT *
      FROM E E2
      WHERE E1.S = E2.S
      AND E1.T <> E2.T))
```

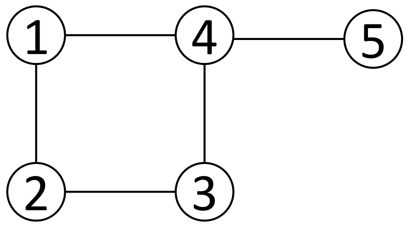
false

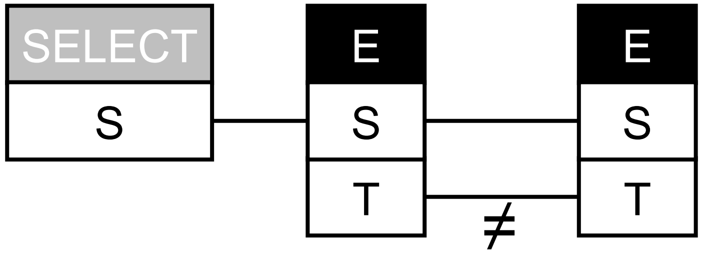
In DRC, SQL, RD, and now in TRC

E(S,T)

501

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]\}$
  - $\{q \mid \exists e_1 \in E, \exists e_2 \in E [q.S = e_1.S \wedge e_1.S = e_2.S \wedge e_1.T \neq e_2.T]\}$

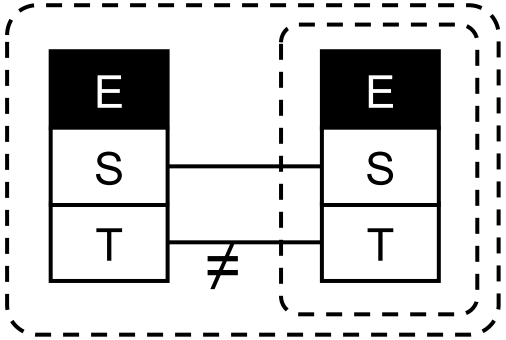





```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
```

{1, 2, 3, 4}

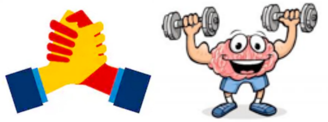
- "each node has at least two distinct neighbors"
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]$
  - $\neg(\exists x\, \neg(\exists y \exists z [E(x,y) \wedge E(x,z) \wedge y \neq z]))$
  - in TRC ?




```
SELECT not exists
   (SELECT *
   FROM E E1
   WHERE not exists
      (SELECT *
      FROM E E2
      WHERE E1.S = E2.S
      AND E1.T <> E2.T))
```

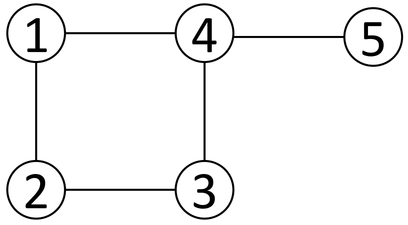
false

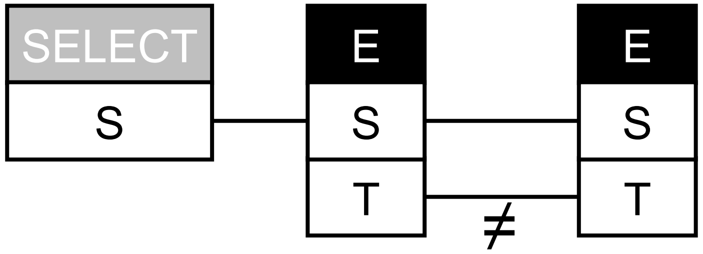
In DRC, SQL, RD, and now in TRC

E(S,T) 

501

- "Find nodes that have at least two distinct neighbors" (query)
  - $\{x \mid \exists y \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]\}$
  - $\{q \mid \exists e_1 \in E, \exists e_2 \in E[q.S = e_1.S \wedge e_1.S = e_2.S \wedge e_1.T \neq e_2.T]\}$

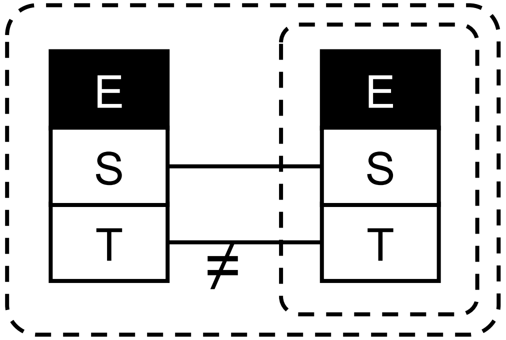





```sql
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
```

{1, 2, 3, 4}

- "each node has at least two distinct neighbors"
  - $\forall x\, \exists y\, \exists z\, [E(x,y) \wedge E(x,z) \wedge y \neq z]$
  - $\neg(\exists x\, \neg(\exists y \exists z[E(x,y) \wedge E(x,z) \wedge y \neq z]))$
  - $\neg(\exists e_1 \in E[\neg(\exists e_2 \in E[e_1.S = e_2.S \wedge e_1.T \neq e_2.T])])$




```sql
SELECT not exists
  (SELECT *
  FROM E E1
  WHERE not exists
    (SELECT *
    FROM E E2
    WHERE E1.S = E2.S
    AND E1.T <> E2.T))
```

false

Queries and the connection to logic

- Why logic?
- A crash course in FOL
- Relational Calculus (RC)
  - Syntax and Semantics
  - Domain RC (DRC) vs Tuple RC (TRC)
  - Domain Independence and Safety
- 4 categorical propositions

Intuition for what we are trying to avoid

Q_1: $\{\ (x) \mid \neg S(x)\ \}$

$S = \{3, 4\}$

1) What's the answer to Q_1?

Intuition for what we are trying to avoid

Q_1: { (x) | ¬S(x) }

$S = \{3, 4\}$

$\mathrm{Dom} = \mathbb{N}_1^{100}$

1) What's the answer to Q_1?

2) What now?

Intuition for what we are trying to avoid

Q_1: $\{ (x) \mid \neg S(x) \}$

$S = \{3, 4\}$ 1) What's the answer to Q_1?

$\text{Dom} = \mathbb{N}_1^{100}$ 2) What now?

Q_2: $\{ (x) \mid R(x) \wedge \neg S(x) \}$

$R = \{1, 2, 3\}$ 3) What's the answer to Q_2?

Intuition for what we are trying to avoid

Q_1: $\{ (x) \mid \neg S(x) \}$

$S = \{3, 4\}$ — 1) What's the answer to Q_1?

$\text{Dom} = \mathbb{N}_1^{100}$ — 2) What now?

Q_2: $\{ (x) \mid R(x) \land \neg S(x) \}$

$R = \{1, 2, 3\}$ — 3) What's the answer to Q_2?

$\text{Dom} = \mathbb{N}_1^{1000}$ — 4) What now?

Intuition for what we are trying to avoid

$\cancel{Q_1\text{: } \{ (x) \mid \neg S(x) \}}$

We "don't like this query" because we can't evaluate it by only looking at the database

$S = \{3, 4\}$

1) What's the answer to Q_1?

$\mathrm{Dom} = \mathbb{N}_1^{100}$

2) What now?

$Q_2\text{: } \{ (x) \mid R(x) \wedge \neg S(x) \}$

$R = \{1, 2, 3\}$

3) What's the answer to Q_2?

$\mathrm{Dom} = \mathbb{N}_1^{1000}$

4) What now?

That's easy to see, but it gets more complicated ☹

Q_2 is "domain-independent", i.e. we don't care whether Dom is $\mathbb{N}_1^{100}$ or $\mathbb{N}_1^{1000}$. We only care about the database D:

R

A
1
2
3

S

A
3
4

Bringing in the Domain

- Let **S** be a schema, D a database over **S**, and Q an RC query over **S**
- Then D gives an unambiguous interpretation for the underlying FOL
  - Predicates ⟶ relations; constants copied; no functions

Is this true ?

Bringing in the Domain

- Let **S** be a schema, D a database over **S**, and Q an RC query over **S**
- ~~Then D gives an unambiguous interpretation for the underlying FOL~~
  - Predicates ⟶ relations; constants copied; no functions
  - Not yet! We need to answer first: What is the domain?

- The active domain **ADom** (of D and Q) is the set of all the values that occur in either D or Q
- The query Q is evaluated over D with respect to a domain **Dom** that contains the active domain (**Dom** ⊇ **ADom**)
- Denote by $Q^{\mathbf{Dom}}(D)$ the result of evaluating Q over D relative to the domain **Dom**

Domain Independence

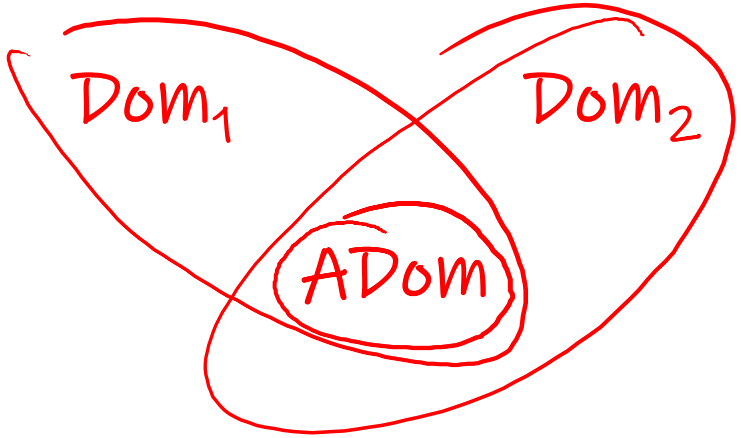
- Let **S** be a schema, and let Q be an RC query over **S**
- We say that Q is domain independent if for every database D over **S** and ...

How could we continue the definition?

Domain Independence

- Let **S** be a schema, and let Q be an RC query over **S**
- We say that Q is domain independent if for every database D over **S** and every two domains **Dom_1** and **Dom_2** that contain the active domain, we have:

$$Q^{Dom1}(D) = Q^{Dom2}(D) = Q^{ADom}(D)$$



First bad news ... and then good news ...

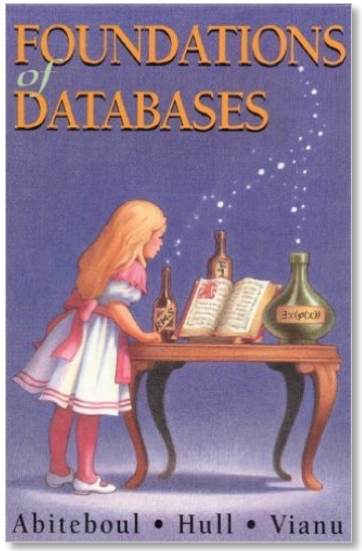
We would like be able to tell whether a given RC query is domain independent, and then reject “bad queries”

- Bad: This problem is undecidable ☹!
  - That is, there is no algorithm that takes as input an RC query and returns true iff the query is domain independent

- Good: Domain-independent RC has an "effective syntax", that is:
    - A syntactic restriction of RC in which every query is domain independent
    - Restricted queries are said to be safe
  - Safety can be tested automatically (and efficiently)
    - Most importantly, for every domain independent RC query there exists an equivalent safe RC query!

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
First observed in "Di Paola. The Recursive Unsolvability of the Decision Problem for the Class of Definite Formulas, JACM 1969. https://doi.org/10.1145/321510.321524"

Safety



- We don't cover the formal definition of the safe syntax
- Details on the safe syntax can be found e.g. in [Alice'95]
- Example:
  - Every variable x_i is guarded by $R(x_1,\ldots,x_k)$
  - In "$\exists x\ \varphi$", the variable x should be guarded by φ
  - In "$\psi \wedge (x=y)$", the variable x is guarded iff either x or y is guarded by ψ
  - ...

The basic idea of these definitions is to ensure that every free variable in the query is somehow bound to an element in the active domain of the database or, in the presence of nontrivial operations, to one of a finite number of domain elements. In the absence of operations, this is typically done by ensuring that every free or existentially quantified variable in a query occurs positively in its scope, every universally quantified variable occurs negatively in its scope, and that the same free variables occur in every component of a disjunction. For example, the query $\{x \mid P(x) \wedge \forall y(Q(x, y) \rightarrow R(x, y))\}$ is safe according to these ideas.

[Alice'95] Abiteboul, Hull, Vianu. Foundations of Databases, 1995. Chapter 5.4 Syntactic Restrictions for Domain Independence. http://webdam.inria.fr/Alice/
An more accessible overview of issues involving safety is: Topor, Safety and Domain Independence, Encyclopedia of Database Systems. https://doi.org/10.1007/978-0-387-39940-9_1255

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

ADom = {1, 2, 3, 'female', 'Canada'}

Dom = ADom ∪ {'elefant', 'car', 'lemon', π, ...}

$\{ (x) \mid \neg \text{Person}(x, \text{'female'}, \text{'Canada'}) \}$?

$\{ (x,y) \mid \exists z\, [\text{Spouse}(x,z) \land y=z] \}$?

$\{ (x,y) \mid \exists z\, [\text{Spouse}(x,z) \land y \neq z] \}$?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

What are example fixes: ?

$\{\ (x) \mid \neg\text{Person}(x, \text{'female'}, \text{'Canada'})\ \}$ Not DI

$\{\ (x,y) \mid \exists z\ [\text{Spouse}(x,z) \wedge y=z]\ \}$?

$\{\ (x,y) \mid \exists z\ [\text{Spouse}(x,z) \wedge y\neq z]\ \}$?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

What are example fixes:

$\wedge \exists y,z.\text{Person}(x,y,z)$
$\wedge$ Person(x,_,_)
$\wedge$ Person(x,_,'Canada')
$\wedge$ x='1' or x='2'

$$\{ (x) \mid \neg\text{Person}(x, \text{'female'}, \text{'Canada'}) \}$$

Not DI

$$\{ (x,y) \mid \exists z\, [\text{Spouse}(x,z) \wedge y=z] \}$$

?

$$\{ (x,y) \mid \exists z\, [\text{Spouse}(x,z) \wedge y\neq z] \}$$

?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

What are example fixes:

$\wedge\ \exists y,z.\text{Person}(x,y,z)$
$\wedge\ \text{Person}(x,\_,\_)$
$\wedge\ \text{Person}(x,\_,\text{'Canada'})$
$\wedge\ x=\text{'1'} \text{ or } x=\text{'2'}$

$\{\ (x) \mid \neg\text{Person}(x, \text{'female'}, \text{'Canada'})\ \}$

Not DI

$\{\ (x,y) \mid \exists z\, [\text{Spouse}(x,z) \wedge y=z]\ \}$

DI

same as $\{(x,y) \mid \text{Spouse}(x,y)\} = \text{Spouse}(x,y)$

$\{\ (x,y) \mid \exists z\, [\text{Spouse}(x,z) \wedge y\neq z]\ \}$

?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

What are example fixes:

$\land\ \exists y,z.\text{Person}(x,y,z)$
$\land\ \text{Person}(x,\_,\_)$
$\land\ \text{Person}(x,\_,\text{'Canada'})$
$\land\ x=\text{'1'}$ or $x=\text{'2'}$

$\{ (x) \mid \neg\text{Person}(x, \text{'female'}, \text{'Canada'}) \}$

Not DI

$\{ (x,y) \mid \exists z\, [\text{Spouse}(x,z) \land y=z] \}$

DI

same as $\{(x,y) \mid \text{Spouse}(x,y)\} = \text{Spouse}(x,y)$

$\{ (x,y) \mid \exists z\, [\text{Spouse}(x,z) \land y \neq z] \}$

?

D: Spouse('Alice','Bob')

ADom={'Alice','Bob'} → ?

Dom ⊇ ADom Dom={'Alice','Bob','Charly'} → ?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

What are example fixes:

$\wedge\ \exists y,z.\text{Person}(x,y,z)$

$\wedge$ Person(x,_,_)

$\wedge$ Person(x,_,'Canada')

$\wedge$ x='1' or x='2'

$$\{\ (x) \mid \neg\text{Person}(x, \text{'female'}, \text{'Canada'})\ \}$$

Not DI

$$\{\ (x,y) \mid \exists z\ [\text{Spouse}(x,z) \wedge y=z]\ \}$$

DI

same as $\{(x,y) \mid \text{Spouse}(x,y)\} = \text{Spouse}(x,y)$

$$\{\ (x,y) \mid \exists z\ [\text{Spouse}(x,z) \wedge y\neq z]\ \}$$

?

D: Spouse('Alice','Bob')

ADom={'Alice','Bob'} $\rightarrow$ {('Alice','Alice')}

Dom $\supseteq$ ADom   Dom={'Alice','Bob','Charly'} $\rightarrow$?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

What are example fixes:

$\wedge\ \exists y,z.\text{Person}(x,y,z)$
$\wedge\ \text{Person}(x,\_,\_)$
$\wedge\ \text{Person}(x,\_,\text{'Canada'})$
$\wedge\ x=\text{'1'}$ or $x=\text{'2'}$

$$\{\ (x) \mid \neg\text{Person}(x, \text{'female'}, \text{'Canada'})\ \}$$

Not DI

$$\{\ (x,y) \mid \exists z\ [\text{Spouse}(x,z) \wedge y{=}z]\ \}$$

DI

same as $\{(x,y) \mid \text{Spouse}(x,y)\} = \text{Spouse}(x,y)$

$$\{\ (x,y) \mid \exists z\ [\text{Spouse}(x,z) \wedge y{\neq}z]\ \}$$

Not DI

D: Spouse('Alice','Bob')

ADom={'Alice','Bob'} $\rightarrow$ {('Alice','Alice')}

Dom $\supseteq$ ADom   Dom={'Alice','Bob','Charly'} $\rightarrow$ {('Alice','Alice'), ('Alice','Charly')}

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \land \exists y\ [\neg \text{Likes}(x,y)] \}$

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \land \forall y\ [\neg \text{Likes}(x,y)] \}$

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \land \forall y\ [\neg \text{Likes}(x,y)] \land \exists y\ [\neg \text{Likes}(x,y)] \}$

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

D

Person('Alice', 'female', 'Canada')	Likes('Alice', 'Beate')
Person('Beate', 'female', 'Canada')	Likes('Alice', 'Cecile')
Person('Cecile', 'female', 'Canada')	Likes('Alice', 'Alice')

ADom = ?

$$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \exists y\ [\neg \text{Likes}(x,y)] \}$$

$$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\ [\neg \text{Likes}(x,y)] \}$$

$$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\ [\neg \text{Likes}(x,y)] \wedge \exists y\ [\neg \text{Likes}(x,y)] \}$$

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

D

Person('Alice', 'female', 'Canada')	Likes('Alice', 'Beate')
Person('Beate', 'female', 'Canada')	Likes('Alice', 'Cecile')
Person('Cecile', 'female', 'Canada')	Likes('Alice', 'Alice')

ADom = {'Alice', 'Beate', 'Cecile', 'female', 'Canada'}

$$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \land \exists y\ [\neg \text{Likes}(x,y)] \}$$

$$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \land \forall y\ [\neg \text{Likes}(x,y)] \}$$

$$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \land \forall y\ [\neg \text{Likes}(x,y)] \land \exists y\ [\neg \text{Likes}(x,y)] \}$$

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

D

Person('Alice', 'Alice', 'Alice')	Likes('Alice', 'Beate')
Person('Beate', 'Beate', 'Beate')	Likes('Alice', 'Cecile')
Person('Cecile', 'Cecile', 'Cecile')	Likes('Alice', 'Alice')

... for the sake of the exercise

ADom = {'Alice', 'Beate', 'Cecile'}

Dom = {'Alice', 'Beate', 'Cecile', 'Dora'}

Exercise for next class ☺

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \exists y\ [\neg \text{Likes}(x,y)] \}$?

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\ [\neg \text{Likes}(x,y)] \}$?

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\ [\neg \text{Likes}(x,y)] \wedge \exists y\ [\neg \text{Likes}(x,y)] \}$?

Updated 1/30/2024

Topic 1: Data models and query languages
Unit 2: Logic & relational calculus
Lecture 6

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

1/30/2024

Pre-class conversations

- Last class recapitulation
- Thanks Haoen for finding a mistake in the slides ☺

- today:
  - we continue with logic (RC) & start with relational algebra (RA)
  - (next week: equivalence of the two)

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

D

Person('Alice', 'Alice', 'Alice')	Likes('Alice', 'Beate')
Person('Beate', 'Beate', 'Beate')	Likes('Alice', 'Cecile')
Person('Cecile', 'Cecile', 'Cecile')	Likes('Alice', 'Alice')

ADom = {'Alice', 'Beate', 'Cecile'}

Dom = {'Alice', 'Beate', 'Cecile', 'Dora'}

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \land \exists y\ [\neg \text{Likes}(x,y)] \}$?

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \land \forall y\ [\neg \text{Likes}(x,y)] \}$?

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \land \forall y\ [\neg \text{Likes}(x,y)] \land \exists y\ [\neg \text{Likes}(x,y)] \}$?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

D

Person('Alice', 'Alice', 'Alice')	Likes('Alice', 'Beate')
Person('Beate', 'Beate', 'Beate')	Likes('Alice', 'Cecile')
Person('Cecile', 'Cecile', 'Cecile')	Likes('Alice', 'Alice')

ADom = {'Alice', 'Beate', 'Cecile')

Dom = {'Alice', 'Beate', 'Cecile', 'Dora')

Example fix:

?

$\{\, (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \exists y\, [\neg \text{Likes}(x,y)] \,\}$ answer$_{\text{ADom}}$: Beate, Cecile **Not DI**

Alice is in the output only if Dom ⊃ ADom (e.g., Dora ∈Dom)

$\{\, (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\, [\neg \text{Likes}(x,y)] \,\}$?

$\{\, (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\, [\neg \text{Likes}(x,y)] \wedge \exists y\, [\neg \text{Likes}(x,y)] \,\}$?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

D

Person('Alice', 'Alice', 'Alice')	Likes('Alice', 'Beate')
Person('Beate', 'Beate', 'Beate')	Likes('Alice', 'Cecile')
Person('Cecile', 'Cecile', 'Cecile')	Likes('Alice', 'Alice')

ADom = {'Alice', 'Beate', 'Cecile'}

Dom = {'Alice', 'Beate', 'Cecile', 'Dora'}

Person(y,_,_)

Example fix: ... $\wedge\, \exists u,v\, [\text{Person}(y,u,v)]$

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \exists y\, [\neg \text{Likes}(x,y)] \}$ answer$_{ADom}$: Beate, Cecile **Not DI**

Alice is in the output only if Dom ⊃ ADom (e.g., Dora ∈Dom)

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\, [\neg \text{Likes}(x,y)] \}$?

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\, [\neg \text{Likes}(x,y)] \wedge \exists y\, [\neg \text{Likes}(x,y)] \}$?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

D

Person('Alice', 'Alice', 'Alice')	Likes('Alice', 'Beate')
Person('Beate', 'Beate', 'Beate')	Likes('Alice', 'Cecile')
Person('Cecile', 'Cecile', 'Cecile')	Likes('Alice', 'Alice')

ADom = {'Alice', 'Beate', 'Cecile'}

Dom = {'Alice', 'Beate', 'Cecile', 'Dora'}

Person(y,_,_)

Example fix: … $\wedge\ \exists u,v$ [Person(y,u,v)]

{ (x) |∃z,w Person(x,z,w) ∧ ∃y [¬Likes(x,y)] } answer$_{ADom}$: Beate, Cecile

Not DI

Alice is in the output only if Dom ⊃ ADom (e.g., Dora ∈Dom)

{ (x) |∃z,w Person(x,z,w) ∧ ∀y [¬Likes(x,y)] }

DI

x never occurs in Likes(x,_): Beate, Cecile

{ (x) |∃z,w Person(x,z,w) ∧ ∀y [¬Likes(x,y)] ∧ ∃y [¬Likes(x,y)] }

?

Which One is Domain Independent?

Person(id, gender, country)
Likes(person1, person2)
Spouse(person1, person2)

D

Person('Alice', 'Alice', 'Alice')	Likes('Alice', 'Beate')
Person('Beate', 'Beate', 'Beate')	Likes('Alice', 'Cecile')
Person('Cecile', 'Cecile', 'Cecile')	Likes('Alice', 'Alice')

ADom = {'Alice', 'Beate', 'Cecile'}

Dom = {'Alice', 'Beate', 'Cecile', 'Dora'}

Person(y,_,_)

Example fix: … $\wedge$ $\exists$u,v [Person(y,u,v)]

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \exists y\ [\neg \text{Likes}(x,y)] \}$ answer$_{ADom}$: Beate, Cecile **Not DI**

Alice is in the output only if Dom ⊃ ADom (e.g., Dora ∈Dom)

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\ [\neg \text{Likes}(x,y)] \}$ **DI**

x never occurs in Likes(x,_): Beate, Cecile

$\{ (x) \mid \exists z,w\ \text{Person}(x,z,w) \wedge \forall y\ [\neg \text{Likes}(x,y)] \wedge \cancel{\exists y\ [\neg \text{Likes}(x,y)]} \}$ **DI**

implication (absorption) if Dom ≠ ∅, which is necessary for there to be Person(x,_,_)

What is the meaning of following unsafe expressions?

$\{\, x \mid \exists y.\ R(x) \,\}$?

$\{\, x \mid x \geq 10 \,\}$?

$\{\, x \mid \forall y\ R(x,y) \,\}$?

What is the meaning of following unsafe expressions?

$\{\, x \mid \exists y.\ R(x) \,\}$ logically equivalent to $\{ x \mid R(x) \} = R(x)$

$\{\, x \mid x \geq 10 \,\}$?

$\{\, x \mid \forall y\ R(x,y) \,\}$?

What is the meaning of following unsafe expressions?

$\{\, x \mid \exists y.\ R(x) \,\}$ logically equivalent to $\{x \mid R(x)\} = R(x)$

$\{\, x \mid x \geq 10 \,\}$ What if Dom=$\mathbb{N}$? DI: $\{\, x \mid R(x) \land x \geq 10 \,\}$

$\{\, x \mid \forall y\ R(x,y) \,\}$?

What is the meaning of following unsafe expressions?

$\{\, x \mid \exists y.\, R(x) \,\}$ logically equivalent to $\{ x \mid R(x) \} = R(x)$

$\{\, x \mid x \geq 10 \,\}$ What if Dom=$\mathbb{N}$? DI: $\{\, x \mid R(x) \wedge x \geq 10 \,\}$

$\{\, x \mid \forall y\, R(x,y) \,\}$ D: R('a','a')
ADom={'a'}
Dom={'a','Chile'}

DI ?: $\{\, x \mid \forall y\, [S(y) \rightarrow R(x,y)] \,\}$

 ?

What is the meaning of following unsafe expressions?

$\{ x \mid \exists y.\, R(x) \}$ logically equivalent to $\{ x \mid R(x) \} = R(x)$

$\{ x \mid x \geq 10 \}$ What if Dom=$\mathbb{N}$? DI: $\{ x \mid R(x) \wedge x \geq 10 \}$

$\{ x \mid \forall y\, R(x,y) \}$

D: R('a','a')
ADom={'a'}
Dom={'a','Chile'}

DI ?: $\{ x \mid \forall y\, [S(y) \rightarrow R(x,y)] \}$

what if relation S is empty?

1. always true for S=Ø

$\{ x \mid \forall y\, [\neg S(y) \vee R(x,y)] \}$

See https://en.wikipedia.org/wiki/Vacuous_truth: A conditional statement is vacuously true when the antecedent is false regardless of whether the conclusion or consequent is true or false.

What is the meaning of following unsafe expressions?

$\{\, x \mid \exists y.\ R(x) \,\}$ logically equivalent to $\{\, x \mid R(x) \,\} = R(x)$

$\{\, x \mid x \geq 10 \,\}$ What if Dom=ℕ? DI: $\{\, x \mid R(x) \wedge x \geq 10 \,\}$

$\{\, x \mid \forall y\ R(x,y) \,\}$

D: R('a','a')
ADom={'a'}
Dom={'a','Chile'}

not DI: $\{\, x \mid \forall y\ [S(y) \rightarrow R(x,y)] \,\}$

what if relation S is empty?

1. always true for S=∅

$\{\, x \mid \forall y\ [\neg S(y) \vee R(x,y)] \,\}$

2. alternative way to see that

Neutral element (identity) for ∀ is TRUE

Σ:

Π:

$\vee$:

$\wedge$:

MIN:

What are the neutral elements of these operations ?

See https://en.wikipedia.org/wiki/Vacuous_truth: A conditional statement is vacuously true when the antecedent is false regardless of whether the conclusion or consequent is true or false.

What is the meaning of following unsafe expressions?

$\{ x \mid \exists y.\, R(x) \}$ logically equivalent to $\{ x \mid R(x) \} = R(x)$

$\{ x \mid x \geq 10 \}$ What if Dom=$\mathbb{N}$? DI: $\{ x \mid R(x) \wedge x \geq 10 \}$

$\{ x \mid \forall y\, R(x,y) \}$ D: R('a','a')
ADom={'a'}
Dom={'a','Chile'}

not DI: $\{ x \mid \forall y\, [S(y) \rightarrow R(x,y)] \}$

what if relation S is empty?

Neutral element (identity) for ∀ is TRUE

$\sum$: $0 + x = x$

$\prod$: $1 \cdot x = x$

$\vee$: $\text{FALSE} \vee x = x$ $\exists$: ?

$\wedge$: $\text{TRUE} \wedge x = x$ $\forall$: ?

MIN: $\text{MIN}(\infty, x) = x$

1. always true for S=∅

$\{ x \mid \forall y\, [\neg S(y) \vee R(x,y)] \}$

2. alternative way to see that

See https://en.wikipedia.org/wiki/Vacuous_truth: A conditional statement is vacuously true when the antecedent is false regardless of whether the conclusion or consequent is true or false.

What is the meaning of following unsafe expressions?

$\{ x \mid \exists y.\ R(x) \}$ logically equivalent to $\{x \mid R(x)\} = R(x)$

$\{ x \mid x \geq 10 \}$ What if Dom=$\mathbb{N}$? DI: $\{ x \mid R(x) \land x \geq 10 \}$

$\{ x \mid \forall y\ R(x,y) \}$

D: R('a','a')
ADom={'a'}
Dom={'a','Chile'}

not DI: $\{ x \mid \forall y\ [S(y) \rightarrow R(x,y)] \}$

what if relation S is empty?

Neutral element (identity) for ∀ is TRUE

Σ: $0 + x = x$

$\prod$: $1 \cdot x = x$

$\lor$: $\text{FALSE} \lor x = x$ $\quad \exists$: $x_1 \lor x_2 \lor \ldots \lor \text{FALSE}$

$\land$: $\text{TRUE} \land x = x$ $\quad \forall$: $x_1 \land x_2 \land \ldots \land \text{TRUE}$

MIN: $\text{MIN}(\infty, x) = x$

1. always true for S=∅

$\{ x \mid \forall y\ [\neg S(y) \lor R(x,y)] \}$

2. alternative way to see that

See https://en.wikipedia.org/wiki/Vacuous_truth: A conditional statement is vacuously true when the antecedent is false regardless of whether the conclusion or consequent is true or false.

What is the meaning of following unsafe expressions?

$\{ x \mid \exists y.\, R(x) \}$ logically equivalent to $\{ x \mid R(x) \} = R(x)$

$\{ x \mid x \geq 10 \}$ What if Dom=$\mathbb{N}$? DI: $\{ x \mid R(x) \wedge x \geq 10 \}$

$\{ x \mid \forall y\, R(x,y) \}$ D: R('a','a')
ADom={'a'}
Dom={'a','Chile'}

not DI: $\{ x \mid \forall y\, [S(y) \rightarrow R(x,y)] \}$

what if relation S is empty?

Neutral element (identity) for ∀ is TRUE

another way to see it: The following sentence

$\forall y\, [R(y)]$

is vacuously true if the domain for y is empty set:

$\forall y\, [y \in Dom \rightarrow R(y)]$

1. always true for S=∅

$\{ x \mid \forall y\, [\neg S(y) \vee R(x,y)] \}$

2. alternative way to see that

What is the meaning of following unsafe expressions?

$\{ x \mid \exists y.\, R(x) \}$ logically equivalent to $\{ x \mid R(x) \} = R(x)$

$\{ x \mid x \geq 10 \}$ What if Dom=$\mathbb{N}$? DI: $\{ x \mid R(x) \land x \geq 10 \}$

$\{ x \mid \forall y\, R(x,y) \}$ not DI: $\{ x \mid \forall y\, [S(y) \rightarrow R(x,y)] \}$

DI: ?

What is the meaning of following unsafe expressions?

$\{ x \mid \exists y.\ R(x) \}$ logically equivalent to $\{ x \mid R(x) \} = R(x)$

$\{ x \mid x \geq 10 \}$ What if Dom=ℕ? DI: $\{ x \mid R(x) \wedge x \geq 10 \}$

$\{ x \mid \forall y\ R(x,y) \}$ not DI: $\{ x \mid \forall y\ [S(y) \rightarrow R(x,y)] \}$

$\exists z [R(x,z) \wedge \ldots]$

DI: $\{ x \mid R(x,\_) \wedge \forall y\ [S(y) \rightarrow R(x,y)] \}$

$\{ x \mid R(x,\_) \wedge \nexists y\ [S(y) \wedge \neg R(x,y)] \}$

We will see this last expression again in a future class ☺
In the meantime, try for yourself. How to write in TRC?

Another example on domain-independence

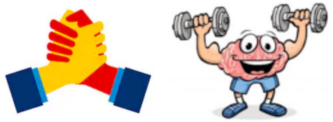
More interestingly, if the domain is the set of natural numbers and the only operation on the domain is linear order, then the query

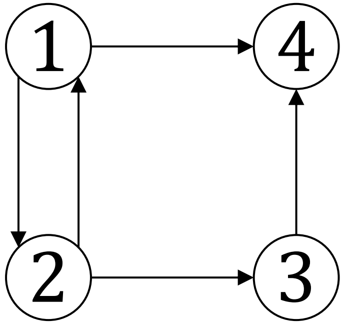
$$Q_4 = \{x \mid \forall y(\Delta(y) \rightarrow x > y) \\ \wedge \forall y(y < x \rightarrow \exists z(\Delta(z) \wedge z \geq y)),$$

where $\Delta(y)$ is true if and only if y is in the active domain of the database, defines the smallest integer greater than all the active domain elements, and is hence finite but not domain independent [7].

Source: Topor, Safety and Domain Independence, Encyclopedia of Database Systems. 2009. https://doi.org/10.1007/978-0-387-39940-9_1255

Example: Querying a Graph





What do these queries return ?

E:

1	2
2	1
2	3
1	4
3	4

$$\{\, x \mid \exists y.\ E(x,y) \,\}$$

?

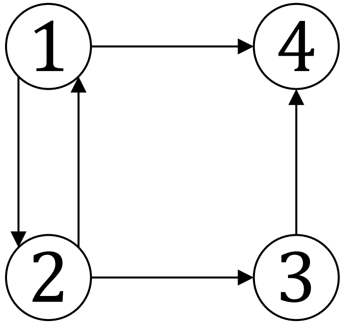
$$\{\, x \mid \exists y,z,u.[E(x,y) \wedge E(y,z) \wedge E(z,u)]\}$$

?

$$\{\, (x,y) \mid \forall z.[E(x,z) \rightarrow E(y,z)]\}$$

?

Example: Querying a Graph



What do these queries return ?

E:

1	2
2	1
2	3
1	4
3	4

$$\{\, x \mid \exists y.\ E(x,y) \,\}$$

Nodes that have at least one child: ?

$$\{\, x \mid \exists y,z,u.[E(x,y) \wedge E(y,z) \wedge E(z,u)] \}$$

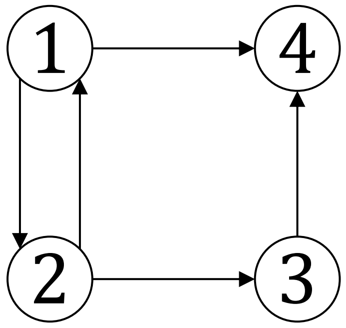
?

$$\{\, (x,y) \mid \forall z.[E(x,z) \rightarrow E(y,z)] \}$$

?

Based on an example by Dan Suciu from CSE 554, 2011.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example: Querying a Graph



What do these queries return ?

E:

1	2
2	1
2	3
1	4
3	4

$$\{\, x \mid \exists y.\ E(x,y) \,\}$$

Nodes that have at least one child: {1,2,3}

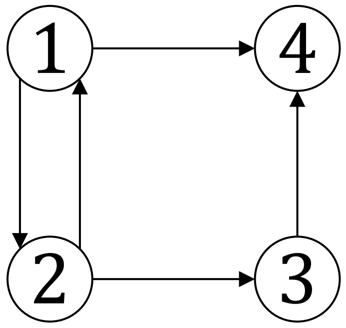
$$\{\, x \mid \exists y,z,u.[E(x,y) \land E(y,z) \land E(z,u)] \}$$

?

$$\{\, (x,y) \mid \forall z.[E(x,z) \rightarrow E(y,z)] \}$$

?

Example: Querying a Graph



E:

1	2
2	1
2	3
1	4
3	4

What do these queries return ?

$$\{\, x \mid \exists y.\ E(x,y) \,\}$$

Nodes that have at least one child: {1,2,3}

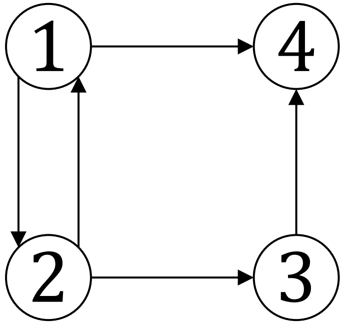
$$\{\, x \mid \exists y,z,u.[E(x,y) \land E(y,z) \land E(z,u)] \}$$

Nodes that have a great-grand-child: ?

$$\{\, (x,y) \mid \forall z.[E(x,z) \rightarrow E(y,z)] \}$$

?

Example: Querying a Graph



What do these queries return ?

E:

1	2
2	1
2	3
1	4
3	4

$$\{\, x \mid \exists y.\, E(x,y) \,\}$$

Nodes that have at least one child: {1,2,3}

$$\{\, x \mid \exists y,z,u.[E(x,y) \land E(y,z) \land E(z,u)]\}$$

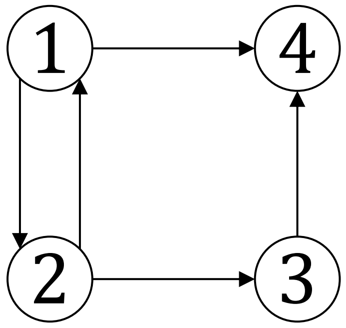
Nodes that have a great-grand-child: {1,2}

y≠u not necessary!
Contrast homomorphism vs. isomorphism ("Hamiltonian Path")

$$\{\, (x,y) \mid \forall z.[E(x,z) \rightarrow E(y,z)]\}$$

?

Example: Querying a Graph



E:

1	2
2	1
2	3
1	4
3	4

What do these queries return ?

$$\{\, x \mid \exists y.\, E(x,y) \,\}$$

Nodes that have at least one child: {1,2,3}

$$\{\, x \mid \exists y,z,u.[E(x,y) \land E(y,z) \land E(z,u)] \,\}$$

Nodes that have a great-grand-child: {1,2}

$\nexists z.[E(x,z) \land \neg E(y,z)]$

$$\{\, (x,y) \mid \forall z.[E(x,z) \rightarrow E(y,z)] \,\}$$

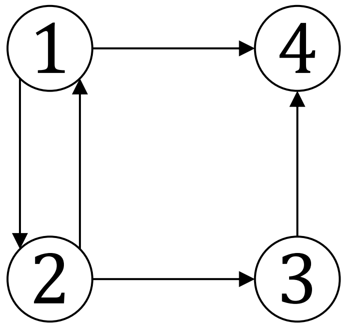
Every child of x is a child of y.

Which of the following tuples fulfill the condition? ?

(1,3) (3,1)

Based on an example by Dan Suciu from CSE 554, 2011.

Example: Querying a Graph



E:

1	2
2	1
2	3
1	4
3	4

What do these queries return ?

$$\{\ x \mid \exists y.\ E(x,y)\ \}$$

Nodes that have at least one child: $\{1,2,3\}$

$$\{\ x \mid \exists y,z,u.[E(x,y) \land E(y,z) \land E(z,u)]\}$$

Nodes that have a great-grand-child: $\{1,2\}$

$\nexists z.[E(x,z) \land \neg E(y,z)]$

$$\{\ (x,y) \mid \forall z.[E(x,z) \rightarrow E(y,z)]\}$$

Every child of x is a child of y.

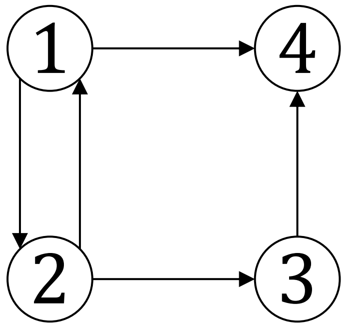
Which of the following tuples fulfill the condition?

~~(1,3)~~ (3,1)

$\{(1,1),(2,2),(3,1),(3,3),(4,1),(4,2),(4,3),(4,4)\}$

if domain is set of nodes!

Example: Querying a Graph



What do these queries return ?

E:

1	2
2	1
2	3
1	4
3	4

$$\{\, x \mid \exists y.\ E(x,y) \,\}$$

Nodes that have at least one child: {1,2,3}

$$\{\, x \mid \exists y,z,u.[E(x,y) \land E(y,z) \land E(z,u)] \,\}$$

Nodes that have a great-grand-child: {1,2}

$\nexists z.[E(x,z) \land \neg E(y,z)]$

$$\{\, (x,y) \mid \forall z.[E(x,z) \rightarrow E(y,z)] \,\}$$

Every child of x is a child of y.

Which of the following tuples fulfill the condition?

$\{\, (x,y) \mid V(x) \land V(y) \land \forall z.[E(x,z) \rightarrow E(y,z)] \,\}$

{(1,1),(2,2),(3,1),(3,3),(4,1), (4,2), (4,3), (4,4)}

if domain is set of nodes!

The person/bar/drinks schema

331

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

What does the following query return?

$$\{\, x \mid \forall y.[\text{Frequents}(x,y) \rightarrow \exists z.[\text{Serves}(y,z) \wedge \text{Likes}(x,z)]]\}$$

?

Schema adapted from Jeff Ullman's drinkers/bars/beers example to avoid attributes with same first letters. https://dl.acm.org/doi/book/10.5555/42790

The person/bar/drinks schema

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

What does the following query return?

$$\{\, x \mid \forall y.[\text{Frequents}(x,y) \rightarrow \exists z.[\text{Serves}(y,z) \wedge \text{Likes}(x,z)]] \,\}$$

Find drinkers who frequent only bars
that serve some drink they like.

Is this query domain independent? ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Schema adapted from Jeff Ullman's drinkers/bars/beers example to avoid attributes with same first letters. https://dl.acm.org/doi/book/10.5555/42790
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

The person/bar/drinks schema

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

What does the following query return?

$$\{\, x \mid \forall y.[\text{Frequents}(x,y) \rightarrow \exists z.[\text{Serves}(y,z) \land \text{Likes}(x,z)]]\,\}$$

Find drinkers who frequent <u>only</u> bars
that serve <u>some</u> drink they like.

This query is not domain independent.
How to fix?

?

Its output would include all values from the domain that do not appear in the Frequents(x,_)

The person/bar/drinks schema

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

What does the following query return?

Frequents(x,_) ∧ …
Likes(x,_) ∧ …

Are those two options to make it safe identical ?

$$\{\, x \mid \forall y.[\text{Frequents}(x,y) \rightarrow \exists z.[\text{Serves}(y,z) \wedge \text{Likes}(x,z)]]\}$$

Find drinkers who frequent <u>only</u> bars that serve <u>some</u> drink they like.

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Schema adapted from Jeff Ullman's drinkers/bars/beers example to avoid attributes with same first letters. https://dl.acm.org/doi/book/10.5555/42790
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

The person/bar/drinks schema

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

331

What does the following query return?

Frequents(x,_) ∧ ...
Likes(x,_) ∧ ...

Both safe, but not identical. Tip: Should a drinker who likes a drink but does not frequent any bar be returned?

$$\{\, x \mid \forall y.[\text{Frequents}(x,y) \rightarrow \exists z.[\text{Serves}(y,z) \wedge \text{Likes}(x,z)]] \,\}$$

Find drinkers who frequent <u>only</u> bars
that serve <u>some</u> drink they like.

Challenge: write this query without the ∀ quantifier!
And then in SQL

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Schema adapted from Jeff Ullman's drinkers/bars/beers example to avoid attributes with same first letters. https://dl.acm.org/doi/book/10.5555/42790

The person/bar/drinks example

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

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

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

?

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

$$\{ x \mid \exists w.[\text{Frequents}(x,w) \land \forall y.[\text{Frequents}(x,y) \rightarrow \exists z.[\text{Serves}(y,z) \land \text{Likes}(x,z)]]\}$$

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

?

Find persons who 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.)

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Schema adapted from Jeff Ullman's drinkers/bars/beers example to avoid attributes with same first letters. https://dl.acm.org/doi/book/10.5555/42790

Queries and the connection to logic

- Why logic?
- A crash course in FOL
- Relational Calculus (RC)
  - Syntax and Semantics
  - Domain RC (DRC) vs Tuple RC (TRC)
  - Domain Independence and Safety
- 4 categorical propositions

4 categorical propositions

S… subject
P… predicate

All S are P

No S is P

Some S is P

Not all S are P (= **Some** S **is not** P)

4 categorical propositions / square of opposition

S... subject
P... predicate

All S are P

No S is P

contradiction

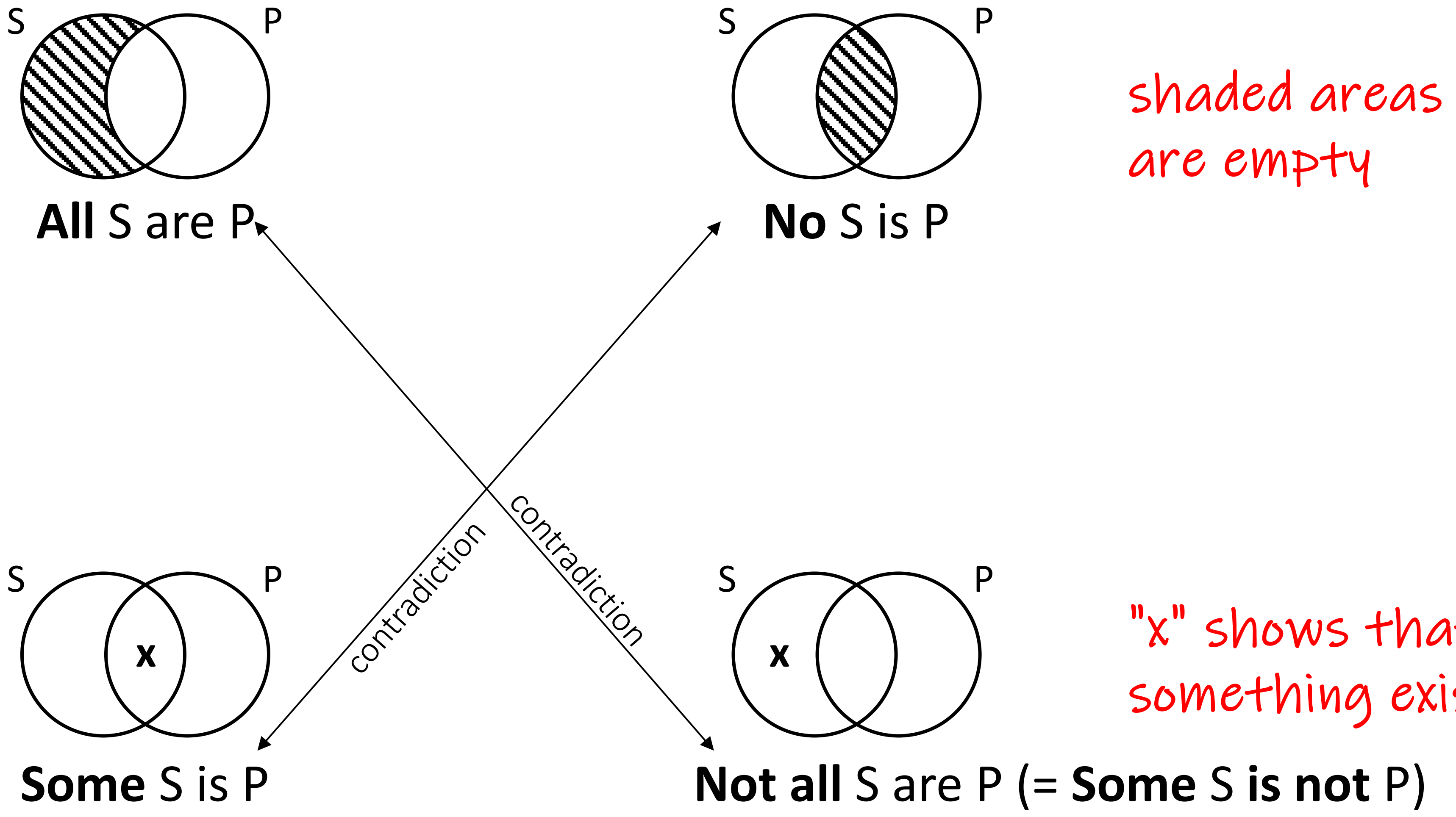
contradiction

Some S is P

Not all S are P (= **Some** S **is not** P)

4 categorical propositions / square of opposition

S... subject
P... predicate



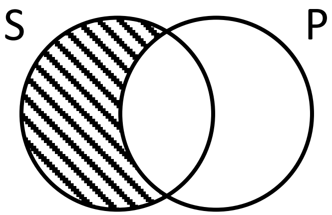
shaded areas are empty

"x" shows that something exists

Notation follows "Venn-I" diagrams by Shin (1994), itself a variant of the extension by Peirce (~1896) of Venn diagrams (1880)

4 categorical propositions / square of opposition

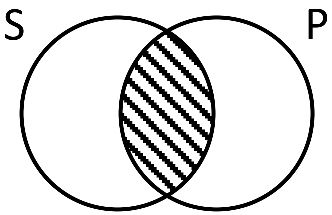
S... subject
P... predicate



All S are P

$\forall x[S(x) \Rightarrow P(x)]$

$\neg\exists x[S(x) \land \neg P(x)]$



No S is P

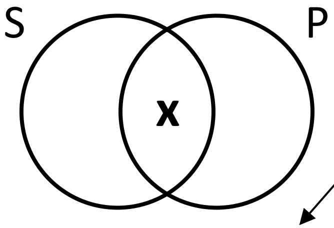
$\forall x[S(x) \Rightarrow \neg P(x)]$

$\neg\exists x[S(x) \land P(x)]$

shaded areas are empty

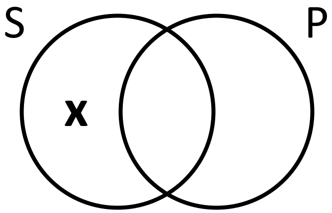
contradiction

contradiction



Some S is P

$\exists x[S(x) \land P(x)]$



Not all S are P (= **Some** S **is not** P)

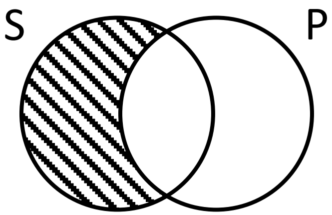
$\exists x[S(x) \land \neg P(x)]$

"x" shows that something exists

4 categorical propositions / square of opposition

S… subject
P… predicate

Universal propositions:

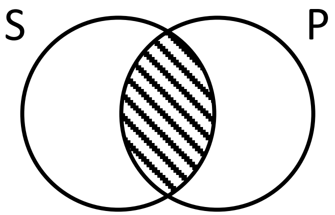


All S are P

$\forall x[S(x) \Rightarrow P(x)]$

$\neg\exists x[S(x) \wedge \neg P(x)]$

A ("Affirmo" = I affirm)



No S is P

$\forall x[S(x) \Rightarrow \neg P(x)]$

$\neg\exists x[S(x) \wedge P(x)]$

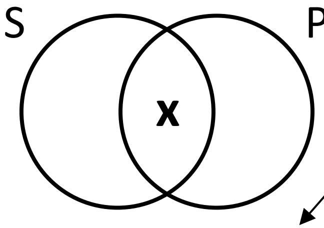
E ("nEgo" = I deny)

shaded areas are empty

contradiction

contradiction

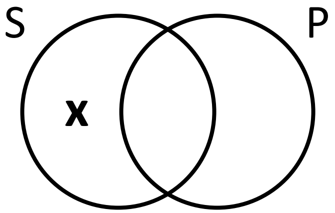
Particular propositions



Some S is P

$\exists x[S(x) \wedge P(x)]$

I ("affIrmo" = I affirm)



Not all S are P (= **Some** S **is not** P)

$\exists x[S(x) \wedge \neg P(x)]$

O ("negO" = I deny)

"x" shows that something exists

For more details see: https://en.wikipedia.org/wiki/Square_of_opposition

4 categorical propositions with Sailors

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

340

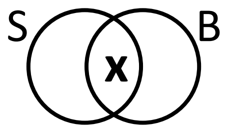
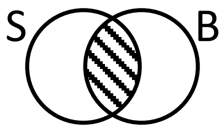
NL	Sailors who reserved **some** red boat	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats

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

4 categorical propositions with Sailors

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

340

Cat.	 Some S is B. $\exists x[S(x) \land B(x)]$	 No S is B. (All S are not B) $\neg\exists x[S(x) \land B(x)]$		
NL	Sailors who reserved **some** red boat	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats

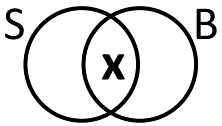
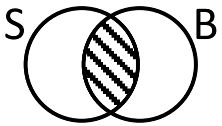
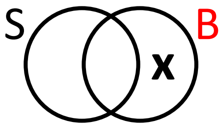
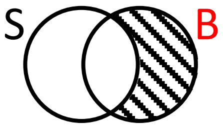
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

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

4 categorical propositions with Sailors

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

340

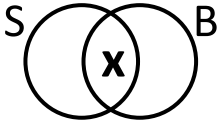
Cat.	 Some S is B. $\exists x[S(x) \wedge B(x)]$	 No S is B. (All S are not B) $\neg\exists x[S(x) \wedge B(x)]$	 Some B is not S. $\exists x[B(x) \wedge \neg S(x)]$	 All B are S. $\neg\exists x[B(x) \wedge \neg S(x)]$
NL	Sailors who reserved **some** red boat	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats

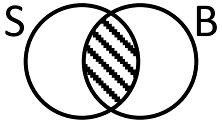
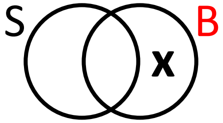
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

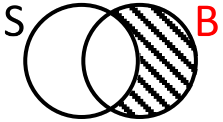
4 categorical propositions with Sailors

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

340

<table>
<tr><td>Cat.</td>
<td>S  B

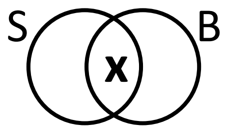
Some S is B.
$\exists x[S(x) \land B(x)]$</td>
<td>S  B
No S is B. (All S are not B)
$\neg\exists x[S(x) \land B(x)]$</td>
<td>S  B

Some B is not S.
$\exists x[B(x) \land \neg S(x)]$</td>
<td>S  B
All B are S.
$\neg\exists x[B(x) \land \neg S(x)]$</td></tr>
<tr><td>NL</td>
<td>Sailors who reserved
some red boat</td>
<td>Sailors who did not reserve
any red boat</td>
<td>Sailors who did not reserve
all red boats</td>
<td>Sailors who reserved
all red boats</td></tr>
<tr><td>SQL</td>
<td>SELECT S.sname
FROM Sailor S
WHERE EXISTS(
  SELECT *
  FROM Reserves R
  WHERE R.sid = S.sid
  AND EXISTS(
    SELECT *
    FROM Boat B
    WHERE B.color = 'red'
    AND B.bid = R.bid))</td>
<td>SELECT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Reserves R
  WHERE R.sid = S.sid
  AND EXISTS(
    SELECT *
    FROM Boat B
    WHERE B.color = 'red'
    AND B.bid = R.bid))</td>
<td>SELECT S.sname
FROM Sailor S
WHERE 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))</td>
<td>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))</td></tr>
</table>

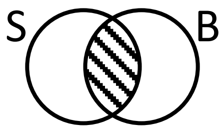
4 categorical propositions with Sailors

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

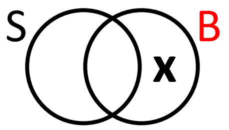
Cat.

1. 

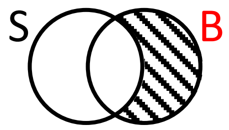
Some S is B.
$\exists x[S(x) \land B(x)]$

2. 

No S is B. (All S are not B)
$\neg\exists x[S(x) \land B(x)]$

3. 

Some B is not S.
$\exists x[B(x) \land \neg S(x)]$

4. 

All B are S.
$\neg\exists x[B(x) \land \neg S(x)]$

NL

Sailors who reserved **some** red boat	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats

SQL

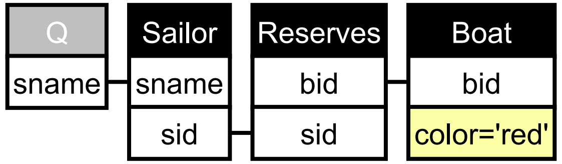
```
SELECT S.sname
FROM Sailor S
WHERE EXISTS(
  SELECT *
  FROM Reserves R
  WHERE R.sid = S.sid
  AND EXISTS(
    SELECT *
    FROM Boat B
    WHERE B.color = 'red'
    AND B.bid = R.bid))
```

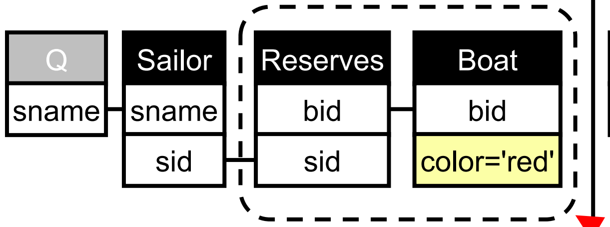
```
SELECT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Reserves R
  WHERE R.sid = S.sid
  AND EXISTS(
    SELECT *
    FROM Boat B
    WHERE B.color = 'red'
    AND B.bid = R.bid))
```

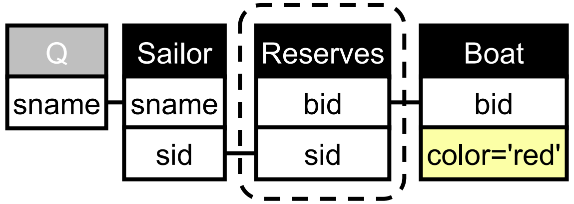
```
SELECT S.sname
FROM Sailor S
WHERE EXISTS(
  SELECT *
  FROM Boat B
  WHERE B.color = 'red'
  AND NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

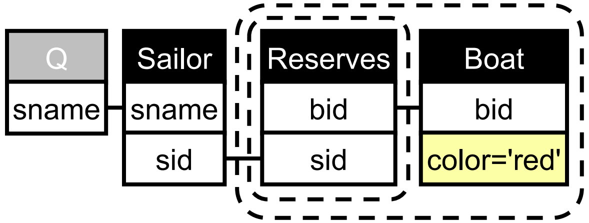
```
SELECT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE B.color = 'red'
  AND NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

RD







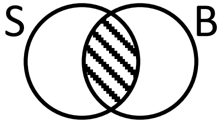
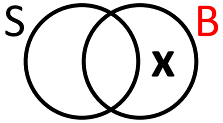
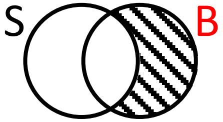
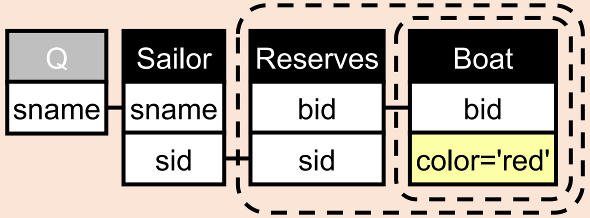
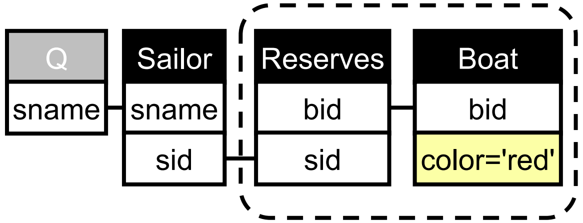
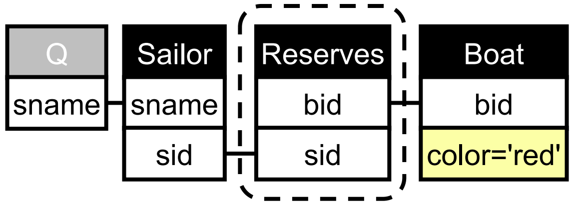
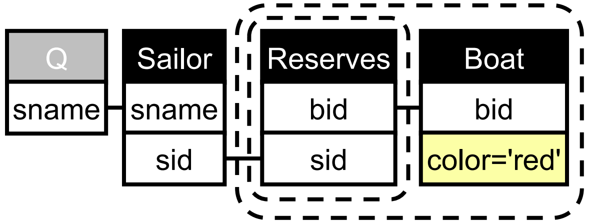


dashed box = not exists

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

A 5th proposition?

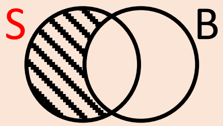
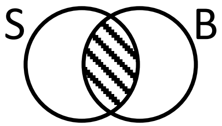
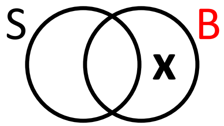
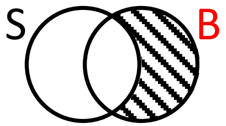
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

Cat.	?	 No S is B. (All S are not B) $\neg\exists x[S(x) \wedge B(x)]$	 Some B is not S. $\exists x[B(x) \wedge \neg S(x)]$	 All B are S. $\neg\exists x[B(x) \wedge \neg S(x)]$
NL	Sailors who reserved **only** red boats	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats
SQL	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 Reserves R WHERE R.sid = S.sid AND EXISTS(SELECT * FROM Boat B WHERE B.color = 'red' AND B.bid = R.bid))	SELECT S.sname FROM Sailor S WHERE 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))	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))
RD				

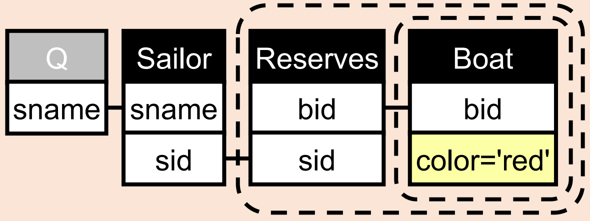
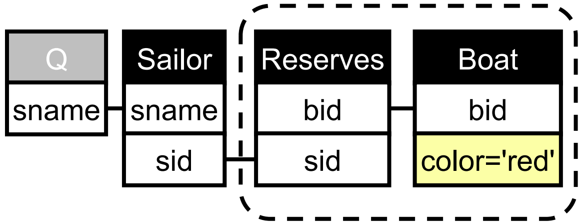
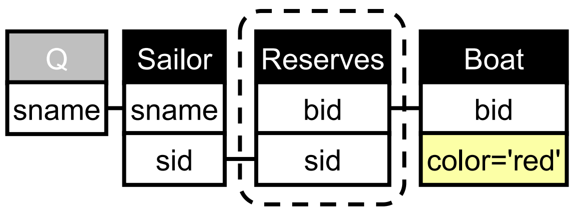
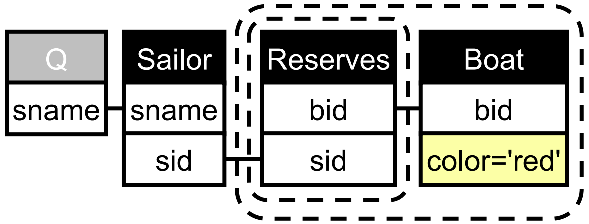
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

"Just" the other direction...

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

Cat.	 All S is B. $\neg\exists x[S(x) \wedge \neg B(x)]$	 No S is B. (All S are not B) $\neg\exists x[S(x) \wedge B(x)]$	 Some B is not S. $\exists x[B(x) \wedge \neg S(x)]$	 All B are S. $\neg\exists x[B(x) \wedge \neg S(x)]$
NL	Sailors who reserved **only** red boats	Sailors who did **not** reserve **any** red boat	Sailors who did **not** reserve **all** red boats	Sailors who reserved **all** red boats

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

RD				

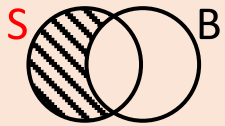
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Limits of Monadic FOL...

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

340

Cat.



All S is B.

$\neg\exists x[S(x) \wedge \neg B(x)]$

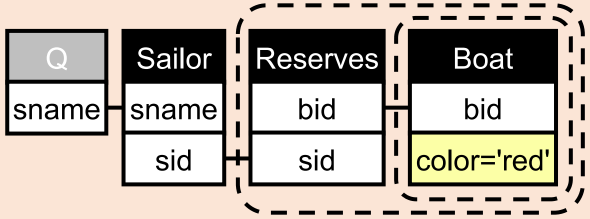
NL

Sailors who reserved **only** red boats

SQL

```
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))
```

RD



Cat.

?

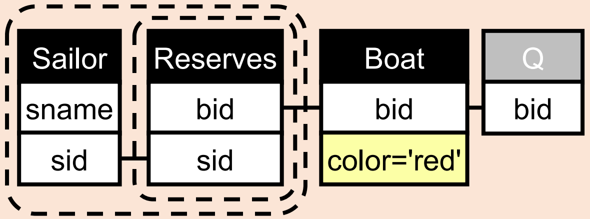
NL

Red boats that were reserved by **all** sailors

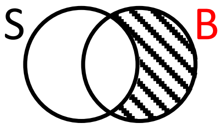
SQL

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

RD



Cat.



All B are S.

$\neg\exists x[B(x) \wedge \neg S(x)]$

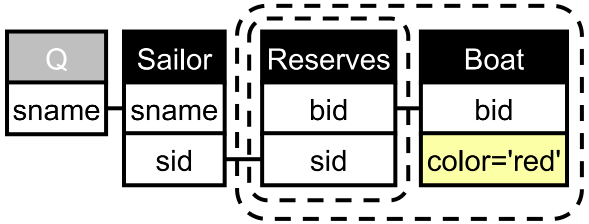
NL

Sailors who reserved **all** red boats

SQL

```
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))
```

RD

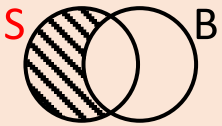


Limits of Monadic FOL...

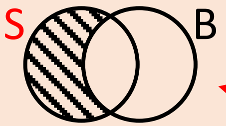
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

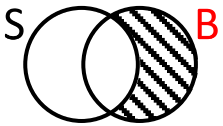
Cat.



All S is B.
$\neg\exists x[S(x) \land \neg B(x)]$



All S is B.
$\neg\exists x[S(x) \land \neg B(x)]$



All B are S.
$\neg\exists x[B(x) \land \neg S(x)]$

NL

Sailors who reserved **only** red boats

Red boats that were reserved by **all** sailors

Sailors who reserved **all** red boats

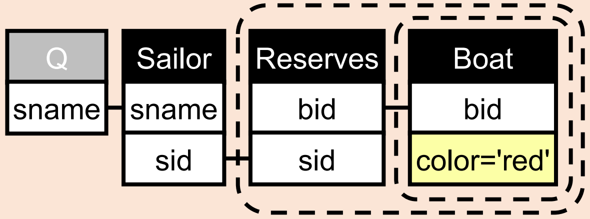
SQL

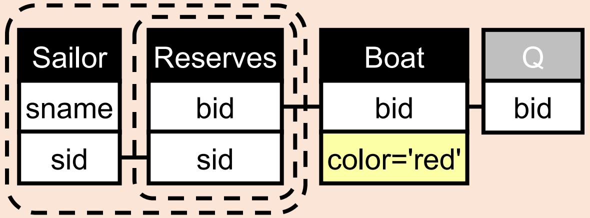
```
SELECT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Reserves R
  WHERE R.sid = S.sid
  AND NOT EXISTS(
    SELECT *
    FROM Boat B
    WHERE B.color = 'red'
    AND B.bid = R.bid))
```

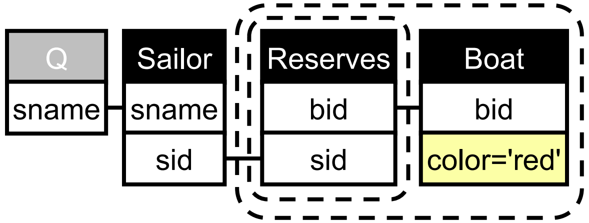
```
SELECT B.bid
FROM Boat B
WHERE B.color = 'red'
AND NOT EXISTS(
  SELECT *
  FROM Sailor S
  WHERE NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

```
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))
```

RD







monadic FOL (which only allows unary predicates, and slightly generalizes syllogistic logic) cannot distinguish between these two queries on the left

See also: https://en.wikipedia.org/wiki/Monadic_predicate_calculus#Relationship_with_term_logic

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

	some	not any	not all	all
Sailors renting boats	have reserved some red boat	have not reserved any red boat	reserved not all red boats	reserved all red boats

Example taken from: https://queryvis.com/example.html

Student (sid, sname)	Sailor (sid, sname, rating, bdate)
Takes (sid, cid, semester)	Reserves (sid, bid, day)
Course (cid, cname, depart)	Boat (bid, bname, color, pdate)

	some	**not any**	**not all**	**all**
Sailors renting boats	have reserved some red boat	have not reserved any red boat	reserved not all red boats	reserved all red boats
Students taking classes	took some art class	took no art class	took not all art classes	took all art classes

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

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

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

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

Example taken from: https://queryvis.com/example.html
Wolfgang Gatterbauer. A Tutorial on Visual Representations of Relational Queries, VLDB tutorial 2023. https://northeastern-datalab.github.io/visual-query-representation-tutorial/

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

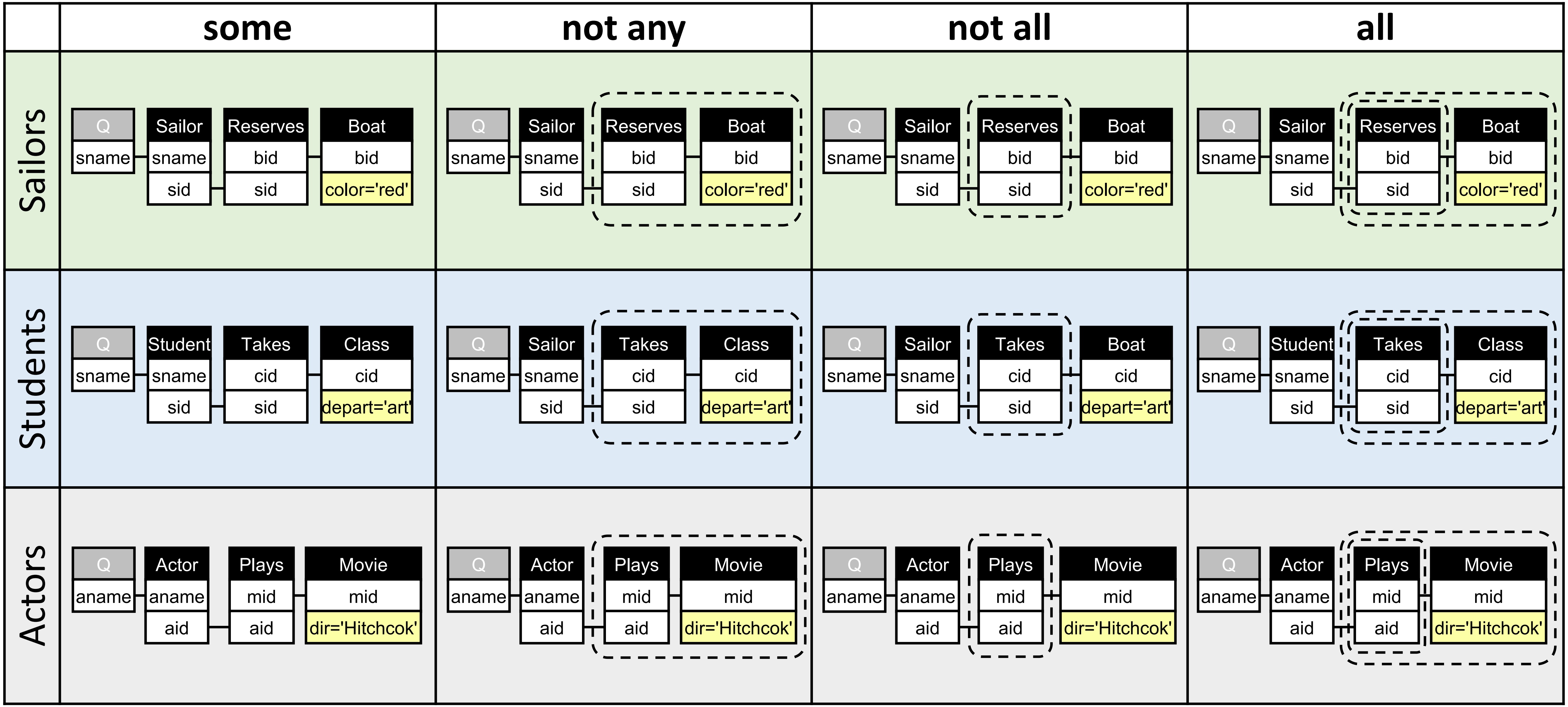
Student (sid, sname)
Takes (sid, cid, semester)
Course (cid, cname, depart)

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

	some	**not any**	**not all**	**all**
Sailors	SELECT S.sname FROM Sailor S WHERE EXISTS(SELECT * FROM Reserves R AND R.sid = S.sid WHERE EXISTS(SELECT * FROM Boat B WHERE B.color = 'red' AND B.bid = R.bid))	SELECT S.sname FROM Sailor S WHERE 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))	SELECT S.sname FROM Sailor S WHERE NOT EXISTS(SELECT * FROM Reserves R AND R.sid = S.sid WHERE 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 EXISTS(SELECT * FROM Takes T AND T.sid = S.sid WHERE EXISTS(SELECT * FROM Class C WHERE C.depart = 'art' AND C.cid = T.cid))	SELECT S.sname FROM Student S WHERE EXISTS(SELECT * FROM Class C WHERE C.depart = 'art' AND NOT EXISTS(SELECT * FROM Takes T WHERE T.cid = C.cid AND T.sid = S.sid))	SELECT S.sname FROM Student S WHERE NOT EXISTS(SELECT * FROM Takes T AND T.sid = S.sid WHERE EXISTS(SELECT * FROM Class C WHERE C.depart = 'art' AND C.cid = T.cid))	SELECT S.sname FROM Student S WHERE NOT EXISTS(SELECT * FROM Class C WHERE C.depart = '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 EXISTS(SELECT * FROM Plays P AND P.aid = A.aid WHERE EXISTS(SELECT * FROM Movie M WHERE M.dir = 'Hitchcock' AND M.mid = P.mid))	SELECT A.aname FROM Actor A WHERE EXISTS(SELECT * FROM Movie M WHERE M.dir = 'Hitchcock' AND NOT EXISTS(SELECT * FROM Plays P WHERE P.mid = M.mid AND P.aid = A.aid))	SELECT A.aname FROM Actor A WHERE NOT EXISTS(SELECT * FROM Plays P AND P.aid = A.aid WHERE EXISTS(SELECT * FROM Movie M WHERE M.dir = 'Hitchcock' AND M.mid = P.mid))	SELECT A.aname FROM Actor A WHERE NOT EXISTS(SELECT * FROM Movie M WHERE M.dir = 'Hitchcock' AND NOT EXISTS(SELECT * FROM Plays P WHERE P.mid = M.mid AND P.aid = A.aid))

Example taken from: https://queryvis.com/example.html
Wolfgang Gatterbauer. A Tutorial on Visual Representations of Relational Queries, VLDB tutorial 2023. https://northeastern-datalab.github.io/visual-query-representation-tutorial/

Actor (<u>aid</u>, aname) Plays (<u>aid, mid, role</u>) Movie (<u>mid</u>, mname, dir)	Student (<u>sid</u>, sname) Takes (<u>sid, cid, semester</u>) Course (<u>cid</u>, cname, depart)	Sailor (<u>sid</u>, sname, rating, bdate) Reserves (<u>sid, bid, day</u>) Boat (<u>bid</u>, bname, color, pdate)



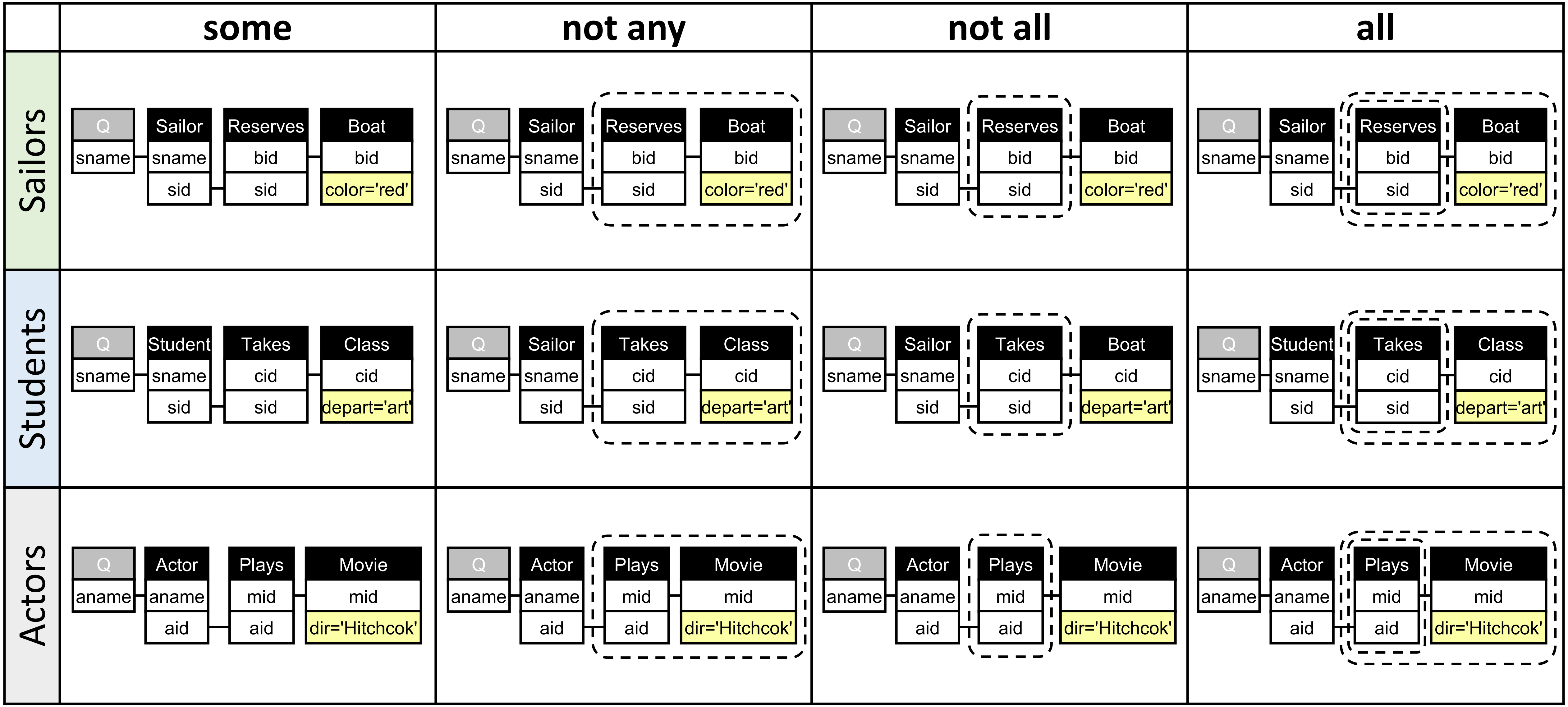
Example taken from: https://queryvis.com/example.html

Wolfgang Gatterbauer. A Tutorial on Visual Representations of Relational Queries, VLDB tutorial 2023. https://northeastern-datalab.github.io/visual-query-representation-tutorial/

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

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

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



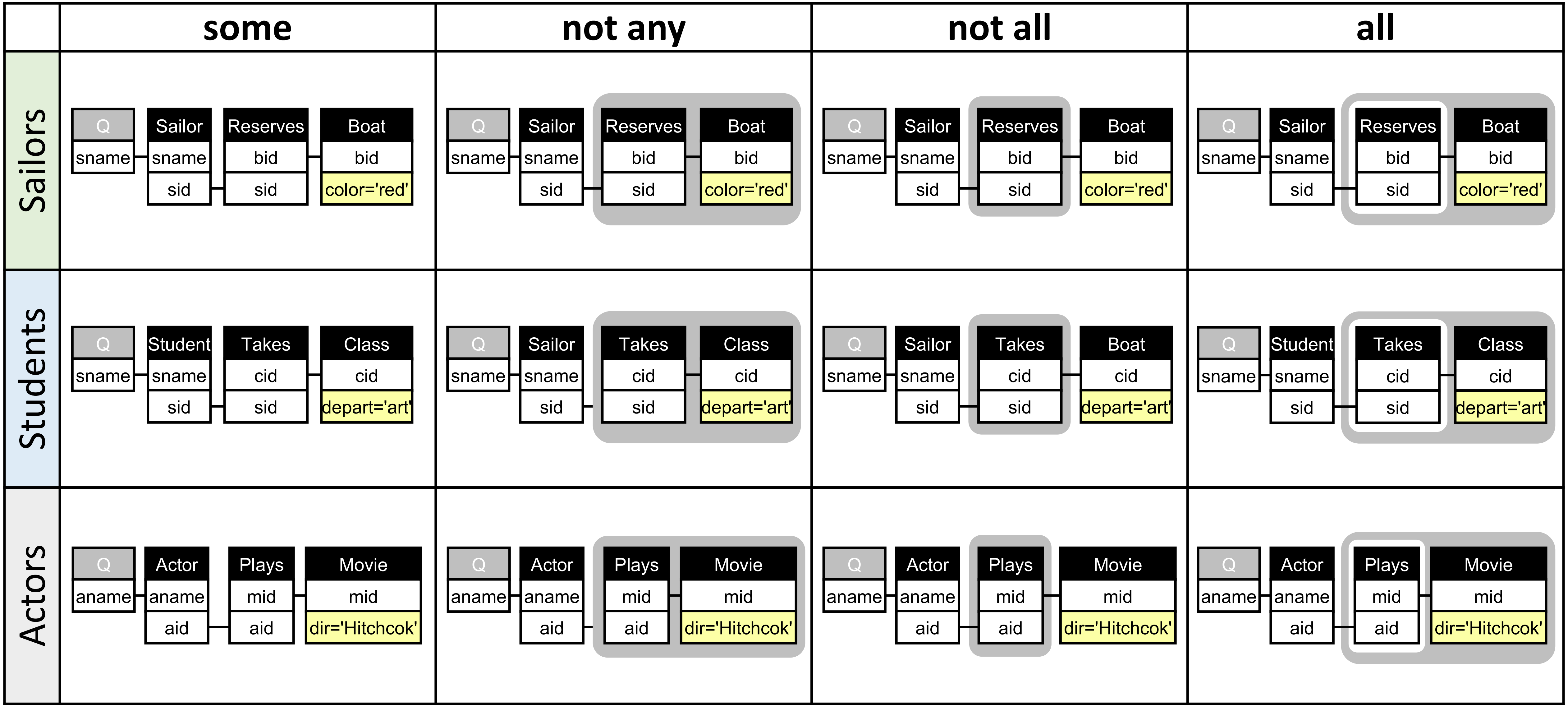
Example taken from: https://queryvis.com/example.html

Wolfgang Gatterbauer. A Tutorial on Visual Representations of Relational Queries, VLDB tutorial 2023. https://northeastern-datalab.github.io/visual-query-representation-tutorial/

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

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

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



Example taken from: https://queryvis.com/example.html
Wolfgang Gatterbauer. A Tutorial on Visual Representations of Relational Queries, VLDB tutorial 2023. https://northeastern-datalab.github.io/visual-query-representation-tutorial/