Updated 1/20/2023

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

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp23)

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

1/20/2023

Topic 1: Data Models and Query Languages

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]

Queries and the connection to logic

- Why logic?
- A crash course in FOL
- Relational Calculus
  - Syntax and Semantics
  - Domain Independence and Safety

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

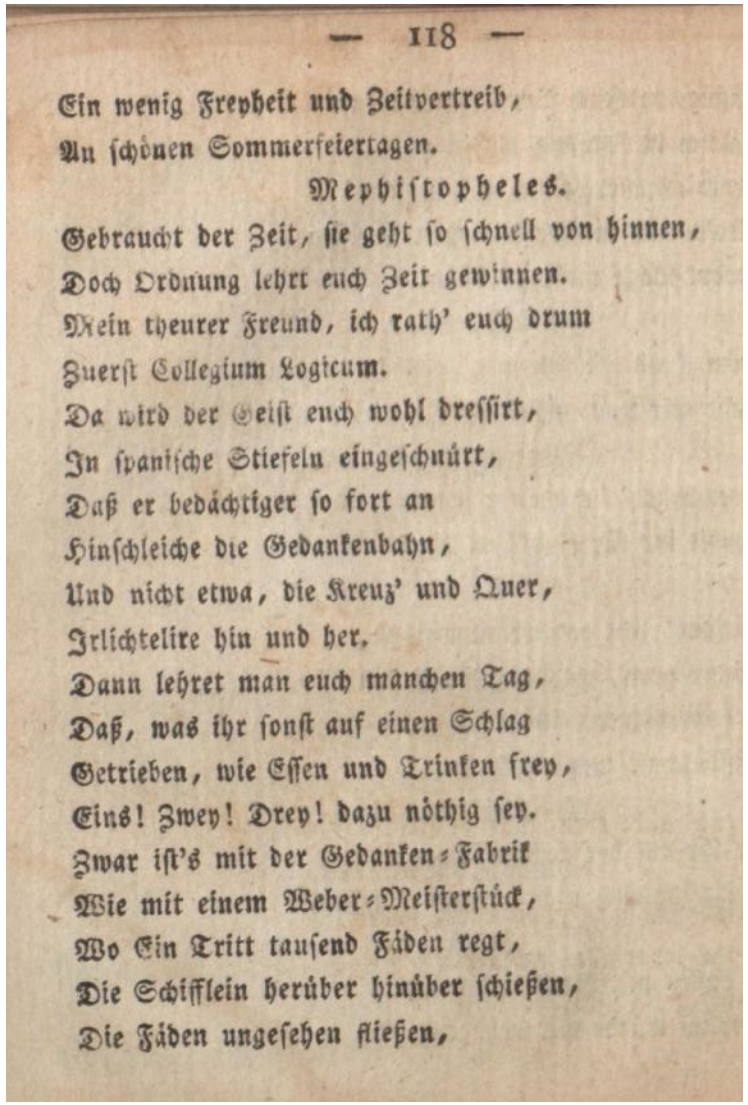
Why has Logic turned out to be so powerful?

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

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,

MEPHISTOPHELES

Gebraucht der Zeit, sie geht so schnell von hinnen,
Doch Ordnung lehrt Euch Zeit gewinnen.
Mein teurer Freund, ich rat Euch drum
Zuerst Collegium Logicum.
Da wird der Geist Euch wohl dressiert,
In spanische Stiefeln eingeschnürt,
Daß er bedächtiger so fortan
Hinschleiche die Gedankenbahn,
Und nicht etwa, die Kreuz und Quer,
Irrlichteliere hin und her.
...

MEPHISTOPHELES

Use your time well: it slips away so fast, yet
Discipline will teach you how to win it.
My dear friend, I'd advise, in sum,
First, the Collegium Logicum.
There your mind will be trained,
As if in Spanish boots, constrained,
So that painfully, as it ought,
It creeps along the way of thought,
Not flitting about all over,
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
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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

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

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

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"

?

Based on: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example: Aristotle' Syllogisms

- "All humans are mortal"
- "For all x, if x is a human, then x is mortal"
- ∀x [Human(x) → 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
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example: Aristotle' Syllogisms

- "All humans are mortal"
- "For all x, if x is a human, then x is mortal"
- ∀x [Human(x) → Mortal(x)]

Do you see the connection to referential integrity constraints

∀x [Product(_,_,_,x) → 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

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 a 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.

Based on Phokion Kolaitis' "Logic and Databases" series at Simons Institute, 2016. https://simons.berkeley.edu/talks/logic-and-databases
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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

Queries and the connection to logic

- Why logic?
- A crash course in FOL
- Relational Calculus
  - Syntax and Semantics
  - Domain Independence and Safety

First-Order Logic: some notions

- Objects, e.g., 2
- Predicates (relationships), e.g., 2 < 3
  - notice predicates are Boolean functions
  - e.g., Define f(x,y)=true iff x<y. Thus f(2,3)=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” (∧), “or” (∨), “implies” (→)
  - Both inputs and outputs are Boolean
- Quantifiers, e.g., “for all” (∀), “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
- 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., $f_i: D^n \to D$)

$D = \{1,2,3,4\}$

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

$\subseteq D \times D \times D$

$4 \cdot 4 \cdot 4 = 64$

R1

A	B	C
1	1	1
2	4	3

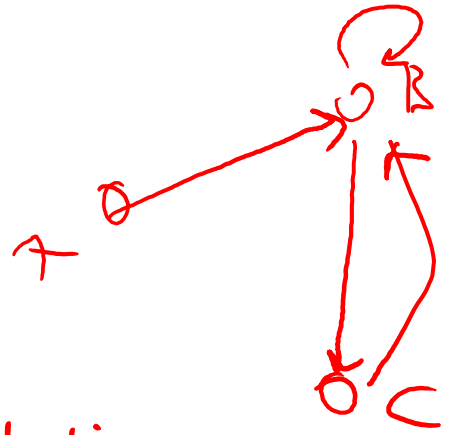
$$f(w_1, w_2) = w_1 + w_2$$

Source: Moshe Vardi: Database Queries - Logic and Complexity
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, . . . (range over nodes)
- Atomic formulas: E(x, y), x = y
- Formulas:
  - Atomic Formulas, and
  - Boolean Connectives (∨, ∧, ¬), and
  - First-Order Quantifiers (∃x, ∀x)



binary relation

Edge

from	to
A	B
B	B
C	B
B	C

Source: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example properties for graph

Assume schema is E(source, target), yet undirected. Thus for every edge E(x,y), we also have E(y,x). As convention later, we use A(x,y) for directed edges = arcs

- "node x has at least two distinct neighbors"

?

- "each node has at least two distinct neighbors"

?

Example adopted from: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example properties for graph

Assume schema is E(source, target), yet undirected. Thus for every edge E(x,y), we also have E(y,x). As convention later, we use A(x,y) for directed edges = arcs

- "node x has at least two distinct neighbors"
  - ∃y ∃z [E(x, y) ∧ E(x, z) ∧ y≠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 ...)

Alice Alice Alice

- "each node has at least two distinct neighbors"

?

Example adopted from: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example properties for graph

Assume schema is E(source, target), yet undirected. Thus for every edge E(x,y), we also have E(y,x). As convention later, we use A(x,y) for directed edges = arcs

- "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"
  - $\forall x \exists y \exists z [E(x, y) \wedge E(x, z) \wedge y \neq z]$
  - The above is a sentence, that is, a formula with no free variables; it expresses a property of graphs.

We will sometimes use $\exists x,y,z$ as short form for $\exists x \exists y \exists z$

Example adopted from: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

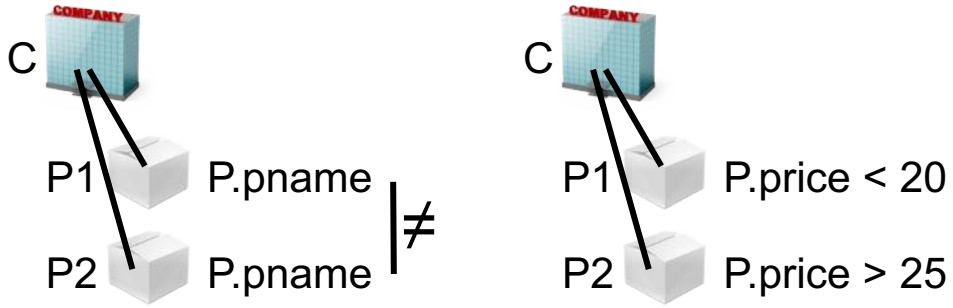
Now in SQL

E(S,T)

501

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

?



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

?

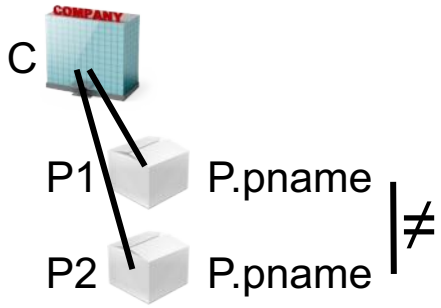
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

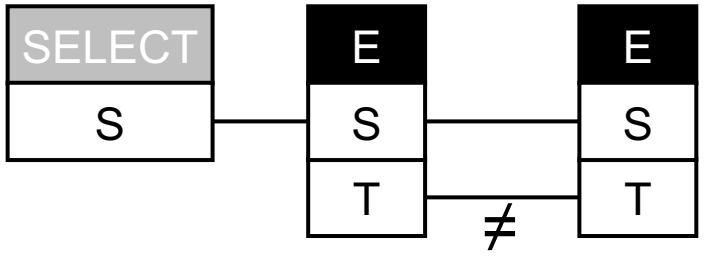
Now in SQL

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" (statement = Boolean query)
  - $\forall x \exists y \exists z [E(x, y) \wedge E(x, z) \wedge y \neq z]$
  - $\neg(\exists x \neg(\exists y \exists z [E(x, y) \wedge E(x, z) \wedge y \neq z]))$

?

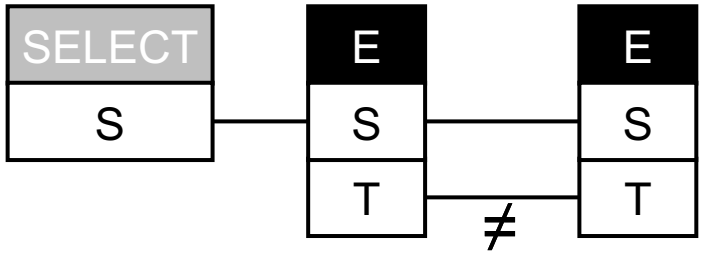
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Now in SQL

E(S,T)

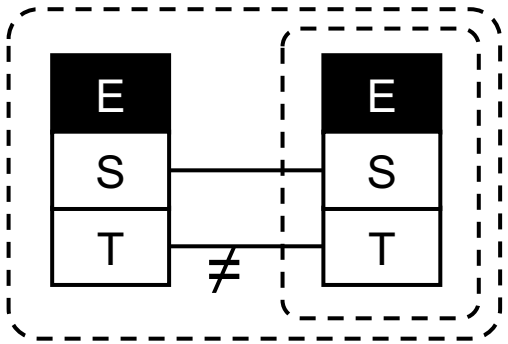
501

- "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))
```

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/

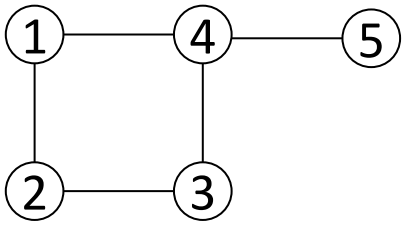
Now in SQL

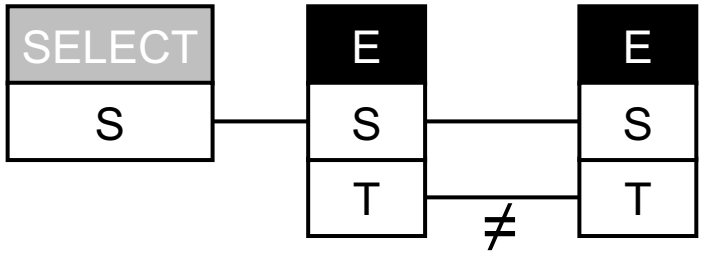
what do the queries return over the shown "graph database" instance

E(S,T)

501

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

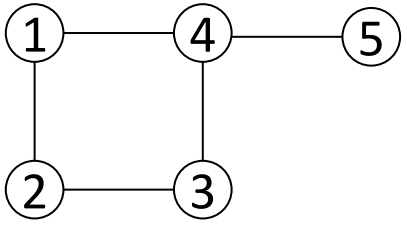


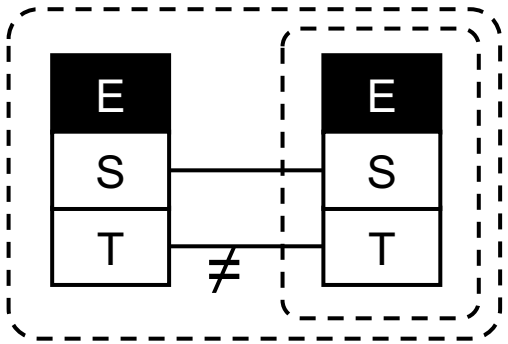



```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
```

?

- "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))
```

?

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

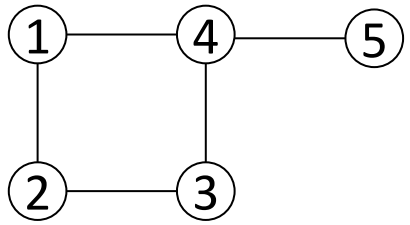
Now in SQL

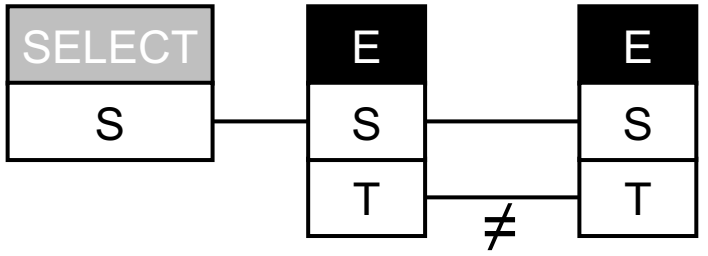
What is a minimal change to the two queries to evaluate them only over nodes 1-4 ?

E(S,T)

501

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

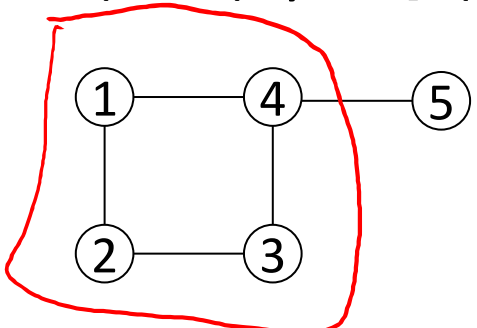


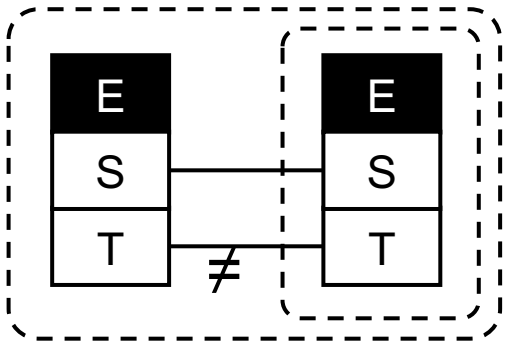



```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
```

{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

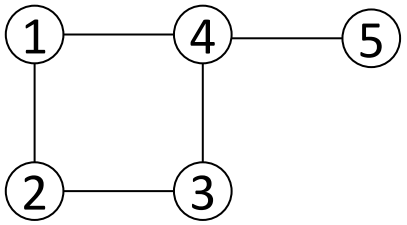
Now in SQL

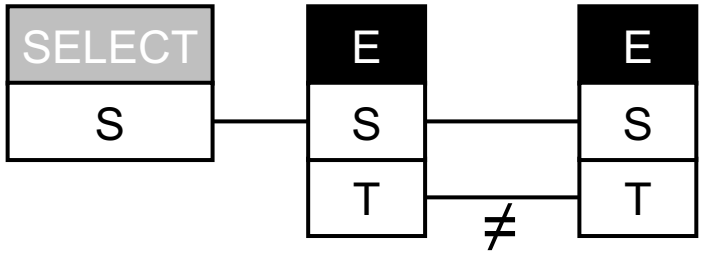
A minimal change to the two queries to evaluate them only over nodes 1-4:

E(S,T)

501

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

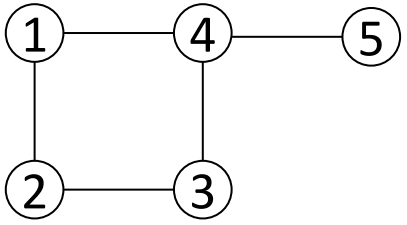


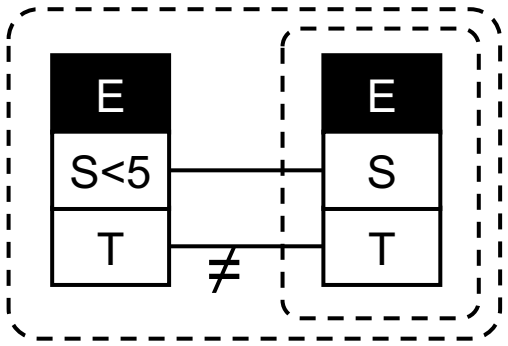



```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T <> E2.T
AND    E1.S<5
```

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

TRUE

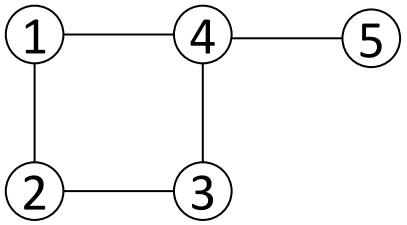
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Now in SQL with grouping

E(S,T)

501

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

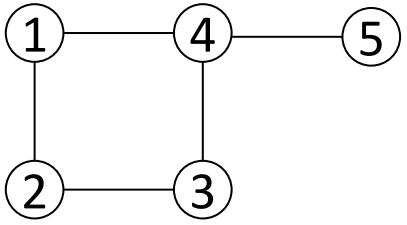


?

```
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

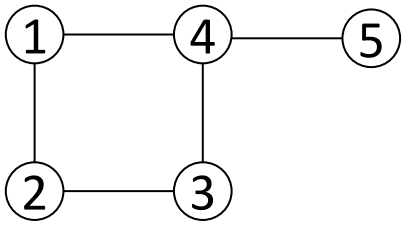
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Now in SQL with grouping

E(S,T)

501

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

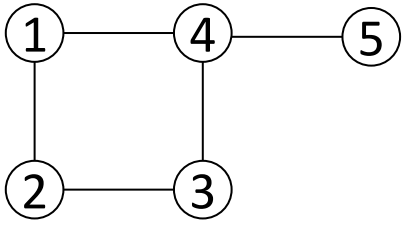



```
SELECT DISTINCT S
FROM    E
GROUP BY S
HAVING COUNT(T)>=2
```

```
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

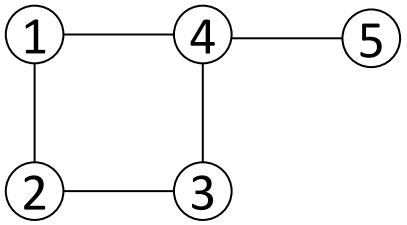
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Now in SQL with grouping

E(S,T)

501

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

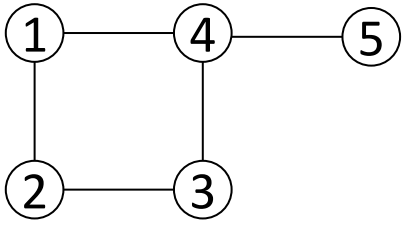



```
SELECT DISTINCT S
FROM    E
GROUP BY S
HAVING COUNT(T)>=2
```

```
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

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/

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."

?

Example adopted from Barker-Plummer, Barwise, Etchemendy - Language, Proof, And Logic (book, 2nd ed), 2011. https://www.gradegrinder.net/
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

One more example

- "There are infinitely many prime numbers"

?

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/

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) ∧ ∃ ( )

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
  - 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:

x = 'Alice'

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

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