Updated 1/28/2022

Topic 1: Data models and query languages
Unit 2: Logic &relational calculus
Lecture 04

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

https://northeastern-datalab.github.io/cs7240/sp22/

1/28/2022

Topic 1: Data Models and Query Languages

PRELIMINARY

- **Lecture 1 (Tue 1/18):** Course introduction, SQL, PostgreSQL setup, SQL Activities
- **Lecture 2 (Fri 1/21):** SQL
- **Lecture 3 (Tue 1/25):** SQL
- **Lecture 4 (Fri 1/28):** Logic & Relational Calculus
- **Lecture 5 (Tue 2/1):** Relational Algebra & Codd's Theorem
- **Lecture 6 (Fri 2/4):** Relational Algebra & Codd's Theorem
- **Lecture 7 (Tue 2/8):** Datalog & Recursion,
- **Lecture 8 (Fri 2/11):** Datalog & Recursion,
- **Lecture 9 (Tue 2/15):** Alternative Data Models

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]

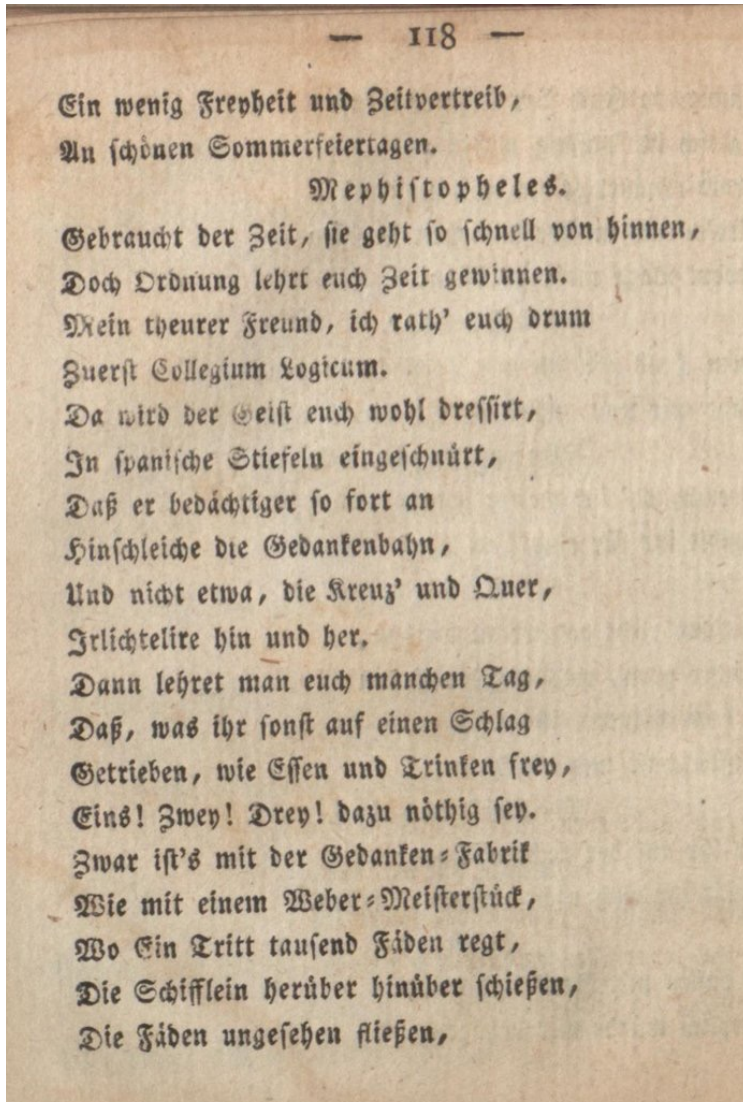
Based on material by Benny Kimelfeld for 236363 Database Management Systems, Technion, 2018.
See also: Halpern, Harper, Immerman, Kolaitis, Vardi, Vianu. "*On the unusual effectiveness of logic in computer science*", 2001. https://doi.org/10.2307/2687775
A play on: Wigner. "*The unreasonable effectiveness 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."

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,
Irlichtelire 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 men are mortal"

?

Example: Aristotle' Syllogisms

- "All men are mortal"

- "For all x, if x is a man, 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 men are mortal"
- "For all x, if x is a man, then x is mortal"
- ∀x [Man(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 men are mortal"
- "For all x, if x is a man, then x is mortal"
- ∀x [Man(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.

Source: Phokion Kolaitis
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.

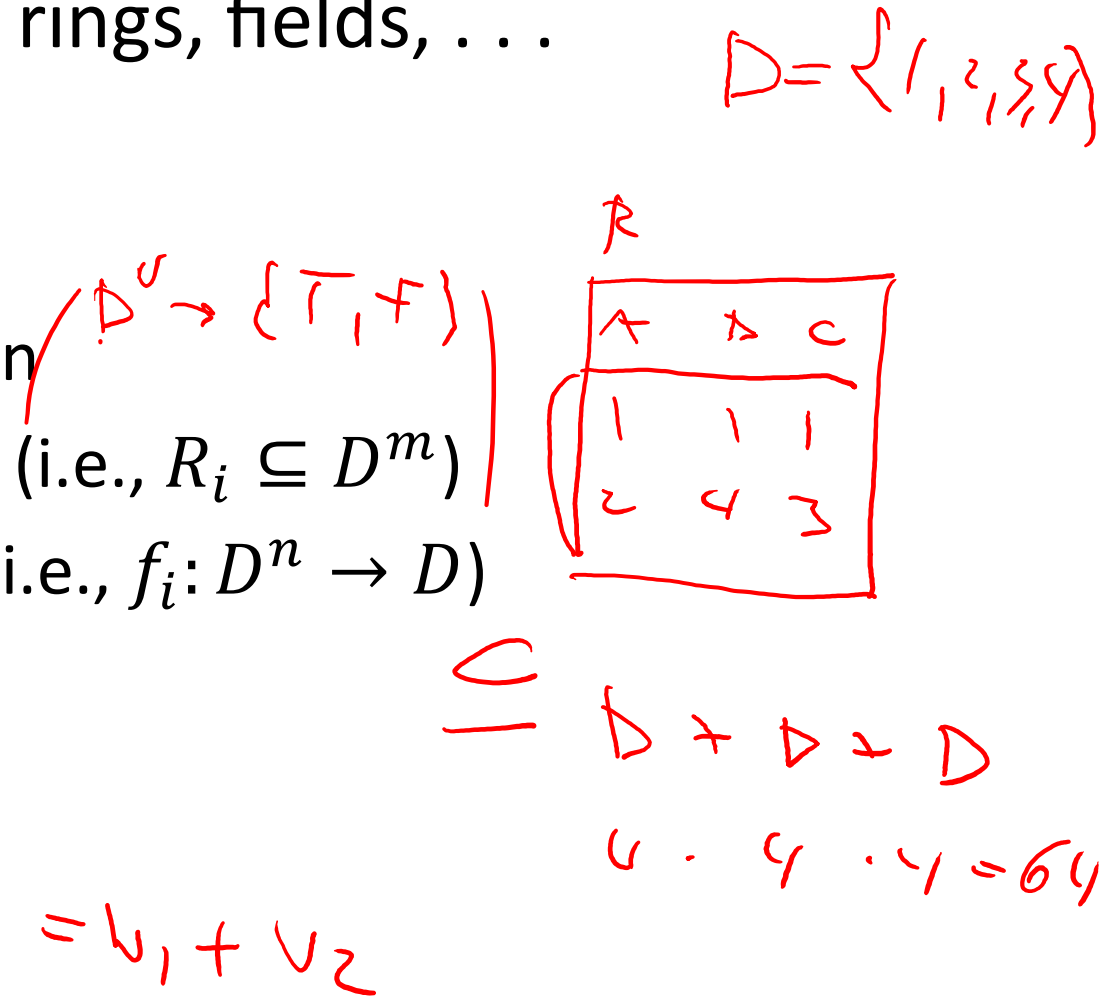
Source: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Queries and the connection to logic

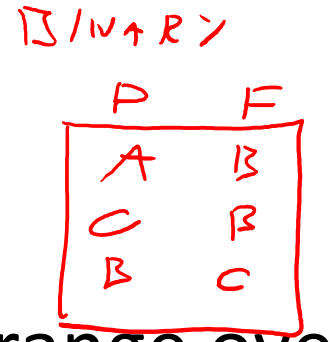
- Why logic?
- A crash course in FOL
- Relational Calculus
  - Syntax and Semantics
  - Domain Independence and Safety

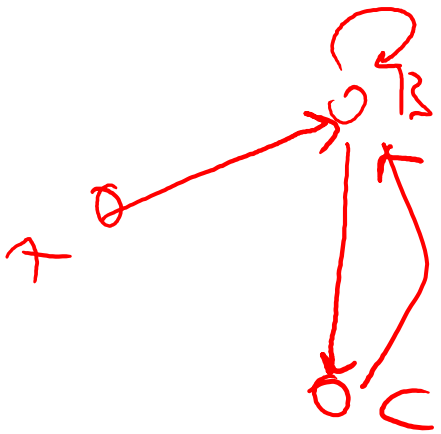
First-Order Logic

- A formalism for specifying properties of mathematical structures, such as graphs, partial orders, groups, rings, fields, . . .
- 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., $f_i: D^n \to D$)



First-Order Logic on Graphs





EDGES

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)

Examples

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

Examples

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

Examples

Assume schema is E(source, target), yet undirected. Thus for every edge E(x,y), we also have E(y,x). As convention, 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 $\exists x \exists y \exists z$

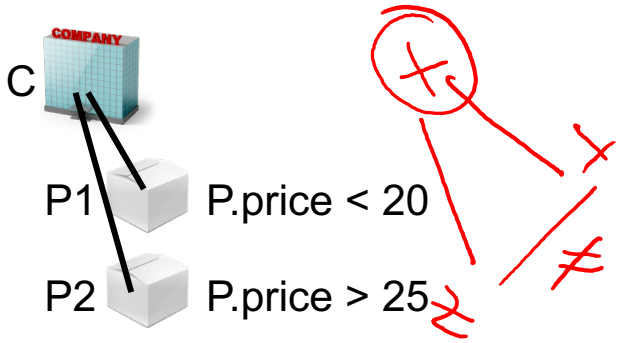
Example adopted from: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Now in SQL

E(S,T)

501

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



?

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

?

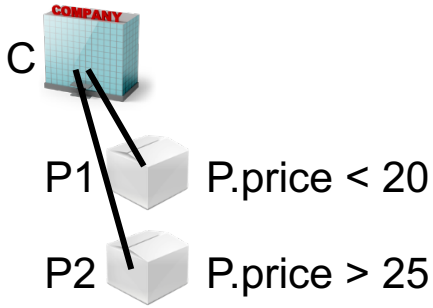
Example adopted from: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

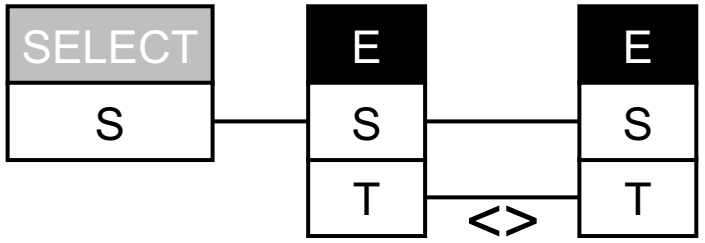
Now in SQL

E(S,T)

501

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






```
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"
  - $\forall x \exists y \exists z [E(x, y) \wedge E(x, z) \wedge y \neq z]$
  - $\neg(\exists x \neg(\exists y \exists z [E(x, y) \wedge E(x, z) \wedge y \neq z]))$

?

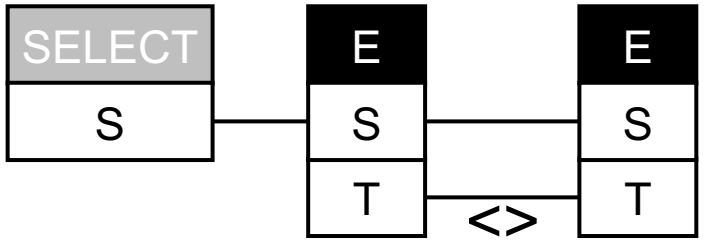
Example adopted from: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Now in SQL

E(S,T)

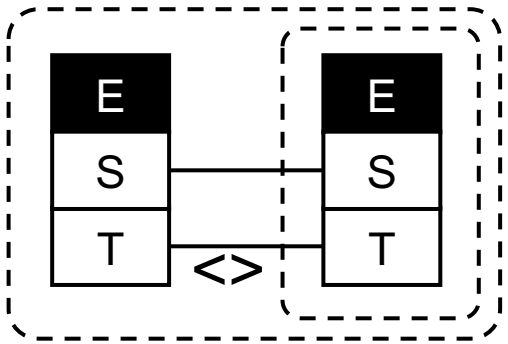
501

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




```
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))
```

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

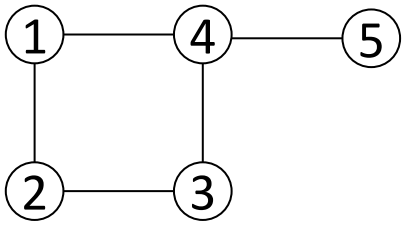
Now in SQL

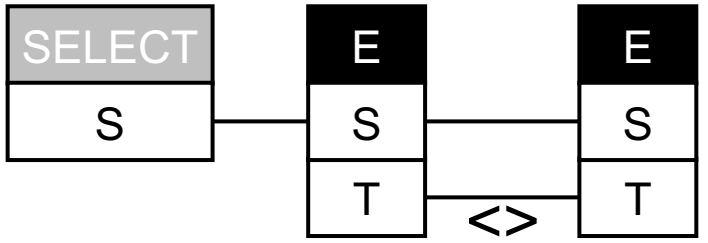
what do the queries return over the shown "graph database" instance

E(S,T)

501

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

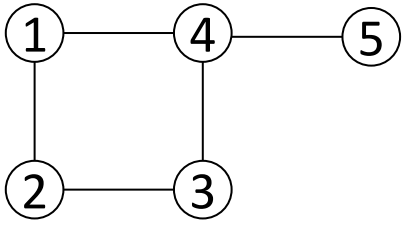


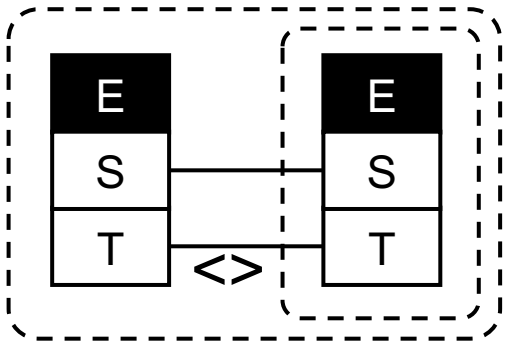



```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T != E2.T
```

?

- "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))
```

?

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

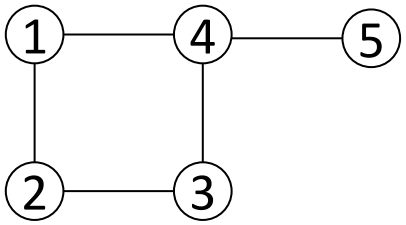
Now in SQL

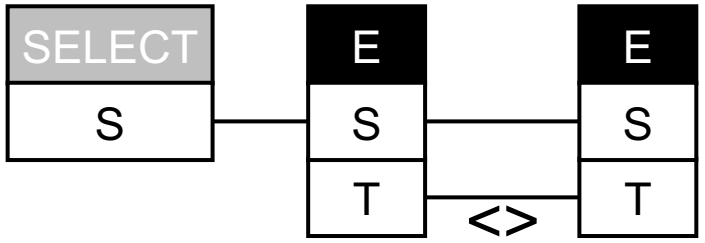
What is a minimal change to the two queries to evaluate them only over nodes 1-4

E(S,T)

501

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

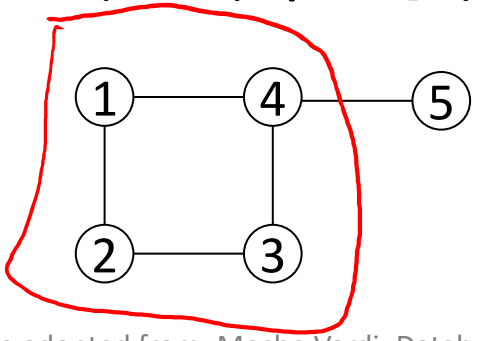


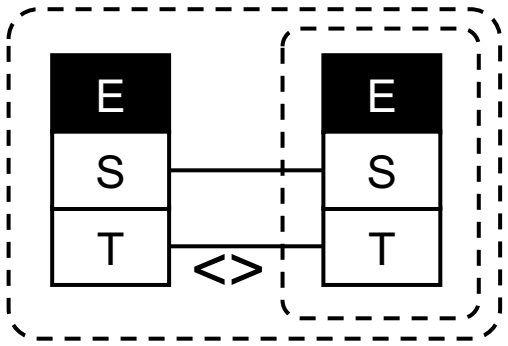



```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T != E2.T
```

{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

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

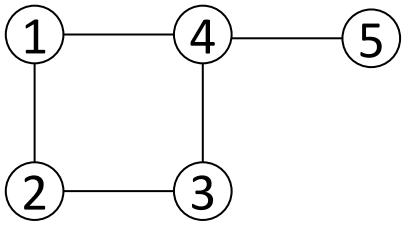
Now in SQL

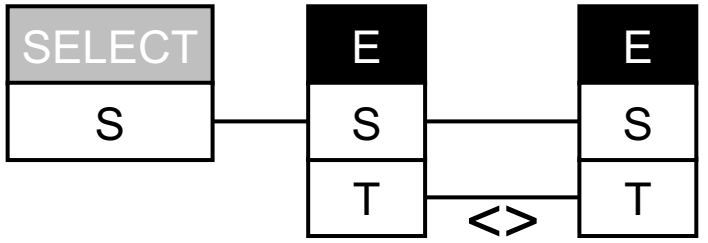
A minimal change to the two queries to evaluate them only over nodes 1-4:

E(S,T)

501

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

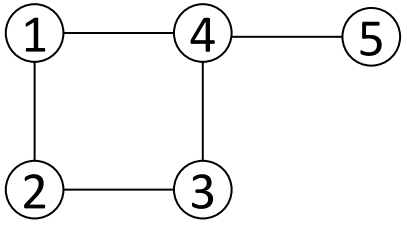


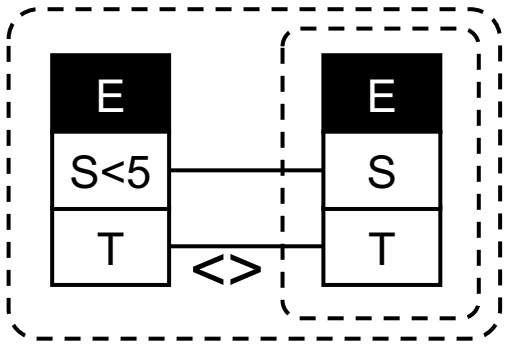



```
SELECT DISTINCT E1.S
FROM   E E1, E E2
WHERE  E1.S = E2.S
AND    E1.T != E2.T
AND    E1.S<5
```

{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))
```

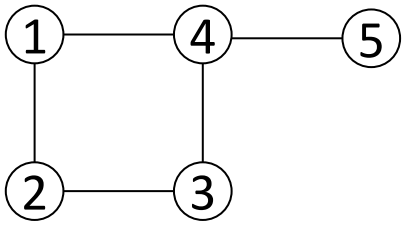
TRUE

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 with grouping

E(S,T)

- "Find nodes that have at least two distinct neighbors"

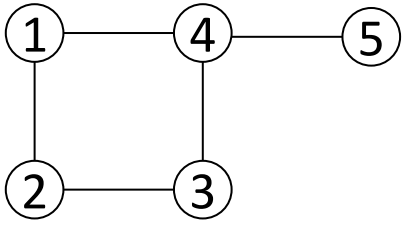


?

```
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

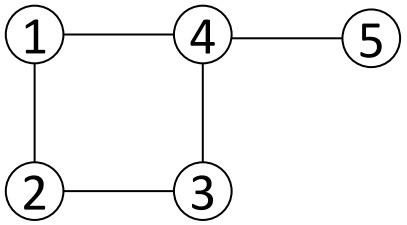
Example adopted from: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Now in SQL with grouping

E(S,T)

501

- "Find nodes that have at least two distinct neighbors"

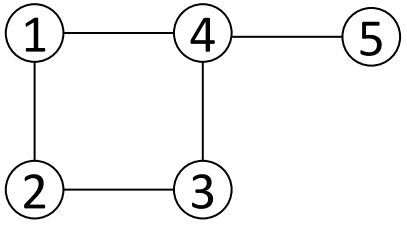



```
SELECT DISTINCT S
FROM   E
GROUP BY S
HAVING COUNT(T)>=2
```

```
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

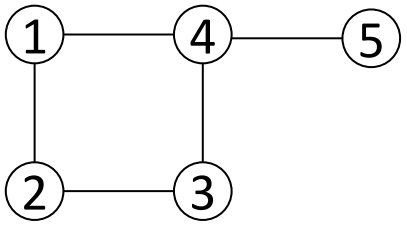
Example adopted from: Moshe Vardi: Database Queries - Logic and Complexity
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Now in SQL with grouping

E(S,T)

501

- "Find nodes that have at least two distinct neighbors"

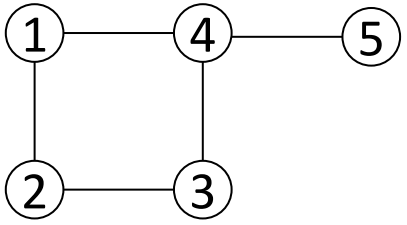



```
SELECT DISTINCT S
FROM   E
GROUP BY S
HAVING COUNT(T)>=2
```

```
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

Example adopted from: Moshe Vardi: Database Queries - Logic and Complexity
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/

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."

?

- "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/

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."

?

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/

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"
  - $\exists x\,[(\text{Small}(x) \wedge \text{Happy}(x) \wedge \text{Dog}(x)) \wedge \text{Owns}(\text{'Jiahui'}, x)]$

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

- "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/

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"
  - $\exists x\,[(\text{Small}(x) \wedge \text{Happy}(x) \wedge \text{Dog}(x)) \wedge \text{Owns}(\text{'Jiahui'}, x)]$

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

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

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/

One more example

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

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/

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.

Semantics of First-Order Logic on Graphs

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

Semantics:

- First-order variables range over (can be " bound to") elements of the universe 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: $G \vDash_{\alpha} \varphi(x_1, \ldots, x_k)$

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

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.

Separation of concerns: WHAT from HOW

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.

Source: Ray Dalio. "Principles", 2017. https://www.principles.com/
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Topic 4: Normalization, Information theory & Axioms of Uncertainty

PRELIMINARY

- Unit 1: Normal Forms & Information Theory
  - [Cow'03] Ramakrishnan, Gehrke. Database Management Systems. 3rd ed 2003. Ch 19: Normalization.
  - [Complete'08] Garcia-Molina, Ullman, Widom. Database Systems. 2nd ed. 2008. Ch 3: Design theory.
  - [Elmasri, Navathe'15] Fundamentals of Database Systems. 7th ed 2015. Ch 14 & 15: Normal forms,
  - [Silberschatz+'20] Silberschatz, Korth, Sudarshan. Database system concepts. 7th ed 2020. Ch 7.5 & 7.6: Relational design with decomposition.
  - [Arenas, Libkin'05] An information-theoretic approach to normal forms for relational and XML data. JACM 2005.
  - [Lee'87] An Information-Theoretic Analysis of Relational Databases-Part I: Data Dependencies and Information Metric. IEEE Transactions on Software Engineering 1987.
  - [Olah'15] Visual Information Theory. Beautiful blog post on the intuition behind entropy.
- Unit 2: Axioms for Uncertainty
  - [Cox'46] Probability, frequency and reasonable expectation, American Journal of Physics, 1946.
  - [Shannon'48] A Mathematical Theory of Communication, The Bell System Technical Journal, 1948.
  - [Van Horn'03] Constructing a logic of plausible inference: a guide to Cox's theorem, International Journal of Approximate Reasoning, 2003.

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

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

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 commonly write constants with quotation marks 'Alice')
  - An assignment of a function $\mathbf{Dom}^n \longrightarrow \mathbf{Dom}$ to each n-ary function symbol
    - Alice = MotherOf(Bob)
  - An assignment of a function $\mathbf{Dom}^n \longrightarrow$ {true, false} (i.e., a relation) to each n-ary predicate symbol
    - Friends(Bob, Charly) = TRUE

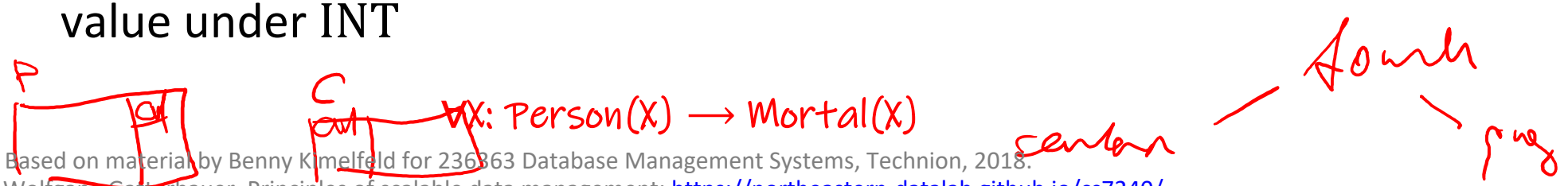
Components of FOL: (3) Semantics

- A variable assignment 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
  - $F \wedge G$ $F \vee G$ $\neg F$ $F \longrightarrow G$ $F \longleftrightarrow G$: 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



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 ⇔.

¬
∧
∨
⇒ ⇔

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.

$\neg p \wedge q$	$((\neg p) \wedge q)$
$p \wedge \neg q$	$(p \wedge (\neg q))$
$p \wedge q \vee r$	$((p \wedge q) \vee r)$
$p \vee q \wedge r$	$(p \vee (q \wedge r)$
$p \Rightarrow q \Rightarrow r$	$(p \Rightarrow (q \Rightarrow r))$
$p \Rightarrow q \Leftrightarrow r$	$(p \Rightarrow (q \Leftrightarrow r))$

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