The Relational Model

Spring 2018

School of Computer Science
University of Waterloo

Databases CS348

How do we ask Questions (and understand Answers)?

In the beginning ...

Set comprehension syntax for questions:

$$\{(x_1, \ldots, x_k) \mid \langle \mathit{condition} \rangle\}$$

Answers:

All *k-tuples* of *values* that satisfy $\langle \mathit{condition} \rangle$.

How do we ask Questions (and understand Answers)?

Find *all pairs of (natural) numbers that add to 5*!

Question: $\{(x, y) \mid x + y = 5\mathrm{PLUS}(x, y, 5)\}$

Answer: $\{(0, 5), (1, 4), (2, 3), (3, 2), (4, 1), (5, 0)\}$

... but but but why? (explain this to a 6 year old!)

because $(0, 5, 5)$, etc., appear in PLUS!

Find pairs of numbers that add to the same number as they subtract to (i.e., $x + y = x - y$)!

Question: $\{(x, y) \mid \exists z.\mathrm{PLUS}(x, y, z) \wedge \mathrm{PLUS}(z, y, x)\}$

Answer: $\{(0, 0), (1, 0), \ldots, (5, 5)?\}$

... answer depends on the content (instance) of PLUS!

Find the *neutral element* (of addition)!

Question: $\{(x) \mid \mathrm{PLUS}(x, x, x)\}$

Answer: $\{(0)\}$

Addition Table

PLUS		
0	0	0
0	1	1
1	0	1
0	2	2
2	0	2
	...	
0	5	5
	...	
1	4	5
	...	
2	3	5
	...	
	...	

How do we ask Questions about Employees?

Find *all employees who work for "Bob"*!

Question: $\{(x, y) \mid \text{EMP}(x, y, Bob)\}$

Answer: $\{(Sue, \text{CS}), (Bob, \text{CO})\}$

why? because (Sue, CS, Bob), etc., appear in EMP!

Find pairs of emp-s working for the same boss!

Q: $\{(x_1, x_2) \mid \exists y_1, y_2, z.\text{EMP}(x_1, y_1, z) \wedge \text{EMP}(x_2, y_2, z)\}$

A: $\{(Sue, Bob), (Fred, John), (Jim, Eve)\}$ $\leftarrow$ is that all?

Find employees who are their own bosses!

Q: $\{(x) \mid \exists y.\text{EMP}(x, y, x)\}$

A: $\{(Sue), (Bob)\}$

Employee Table

EMP

name	dept	boss
Sue	CS	*Bob*
Bob	CO	*Bob*
Fred	PM	*Mark*
John	PM	*Mark*
Jim	CS	*Fred*
Eve	CS	*Fred*
Sue	PM	*Sue*

The Relational Model

Idea

All information is organized in (a finite number of) relations.

Features:

- simple and clean data model
- powerful and declarative query/update languages
- semantic integrity constraints
- data independence

Relational Databases

Components:

Universe	- a set of *values* **D** with equality ($=$)
Relation	- predicate name R, and arity k of R (the number of columns) - instance: a relation $\mathbf{R} \subseteq \mathbf{D}^k$.
Database	- signature: finite set ρ of predicate names - instance: a relation $\mathbf{R_i}$ for each R_i

Notation

Signature: $\rho = (R_1, \ldots, R_n)$ *Instance:* $\mathbf{DB} = (\mathbf{D}, =, \mathbf{R_1}, \ldots, \mathbf{R_n})$

Examples of Relational Databases

- The integers, with addition and multiplication:

  $\rho = (\texttt{PLUS}, \texttt{TIMES})$ $\qquad$ $\mathbf{DB} = (\mathbf{Z}, =, \mathbf{PLUS}, \mathbf{TIMES})$

- A Bibliography Database (see following slides)

- ...

A Bibliography Relational Database Signature

Predicates (also called table headers):

```
AUTHOR(aid, name)
WROTE(author, publication)
PUBLICATION(pubid, title)
BOOK(pubid, publisher, year)
JOURNAL(pubid, volume, no, year)
PROCEEDINGS(pubid, year)
ARTICLE(pubid, crossref, startpage, endpage)
```

$\Rightarrow$ identifiers, called *attributes*, label columns (needed for SQL)

A Bibliography Relational Database Instance

Relations (also called tables):

$$
\begin{array}{rcll}
\texttt{AUTHOR} & = & \{ & (1, \text{John}), (2, \text{Sue}) \quad \} \\
\texttt{WROTE} & = & \{ & (1, 1), (1, 4), (2, 3) \quad \} \\
\texttt{PUBLICATION} & = & \{ & (1, \text{Mathematical Logic}), \\
 & & & (3, \text{Trans. Databases}), \\
 & & & (2, \text{Principles of DB Syst.}), \\
 & & & (4, \text{Query Languages}) \quad \} \\
\texttt{BOOK} & = & \{ & (1, \text{AMS}, 1990) \quad \} \\
\texttt{JOURNAL} & = & \{ & (3, 35, 1, 1990) \quad \} \\
\texttt{PROCEEDINGS} & = & \{ & (2, 1995) \quad \} \\
\texttt{ARTICLE} & = & \{ & (4, 2, 30, 41) \quad \}
\end{array}
$$

A Common Visualization for Relational Databases

AUTHOR

aid	name
1	John
2	Sue

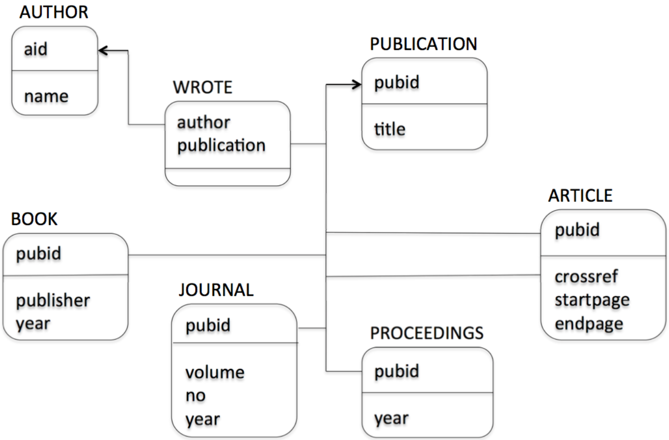
WROTE

author	publication
1	1
1	4
2	3

PUBLICATION

pubid	title
1	Mathematical Logic
3	Trans. Databases
2	Principles of DB Syst.
4	Query Languages

A Common Visualization for Relational Database Schemata†



†Relational database signatures plus *integrity constraints*.

Simple (Atomic) "Truth"

Idea

Relationships between objects (tuples) that are present in an instance are true, relationships absent are false.

In the sample *Bibliography* database instance

- "John" is an author with id "1": $(1, \text{John}) \in \textbf{AUTHOR};$
- "Mathematical Logic" is a publication: $(1, \text{Mathematical Logic}) \in \textbf{PUBLICATION};$
  Moreover, it is a book published by "AMS" in "1990": $(1, \text{AMS}, 1990) \in \textbf{BOOK};$
- "John" wrote "Mathematical Logic": $(1, 1) \in \textbf{WROTE};$
- "John" has **NOT** written "Trans. Databases": $(1, 3) \notin \textbf{WROTE};$
- etc.

Query Conditions

Idea1: use *variables* to generalize conditions

$\mathtt{AUTHOR}(x, y)$ will be true of any valuation $\{x \mapsto a, y \mapsto b, \ldots\}$ exactly when the pair $(a, b) \in$ **AUTHOR**

Idea2: build more complex conditions from simpler ones using ...

Logical connectives:

Conjunction (and):	$\mathtt{AUTHOR}(x, y) \wedge \mathtt{WROTE}(x, z)$
Disjunction (or):	$\mathtt{AUTHOR}(x, y) \vee \mathtt{PUBLICATION}(x, y)$
Negation (not):	$\neg \mathtt{AUTHOR}(x, y)$

Quantifiers:

Existential (there is...): $\exists x.\mathtt{author}(x, y)$

Conditions in the Relational Calculus

Idea

Conditions can be formulated using the language of first-order logic.

Definition (Syntax of Conditions)

Given a database schema $\rho = (R_1, \ldots, R_k)$ and a set of variable names $\{x_1, x_2, \ldots\}$, conditions are *formulas* defined by

$$\varphi ::= \underbrace{\underbrace{\underbrace{R_i(x_{i_1}, \ldots, x_{i_k}) \mid x_i = x_j \mid \varphi \land \varphi \mid \exists x_i.\varphi}_{\text{conjunctive formulas}} \mid \varphi \lor \varphi}_{\text{positive formulas}} \mid \neg\varphi}_{\text{first-order formulas}}$$

First-order Variables and Valuations

How do we *interpret* variables?

Definition (Valuation)

A valuation is a function θ that maps *variable names* to values in the universe:

$$\theta : \{x_1, x_2, \ldots\} \to \mathbf{D}.$$

To denote a modification to θ in which variable x is instead mapped to value v, one writes:

$$\theta[x \mapsto v].$$

Idea

Answers to queries $\Leftrightarrow$ *valuations to free variables that make the formula true with respect to a database.*

Complete Semantics for Conditions

Definition

The *truth* of a formula φ is defined with respect to

1. a **database instance DB** $= (\mathbf{D}, =, \mathbf{R}_1, \mathbf{R}_2, \ldots)$, and
2. a **valuation** $\theta : \{x_1, x_2, \ldots\} \to \mathbf{D}$

as follows:

$$
\begin{array}{ll}
\mathbf{DB}, \theta \models R(x_{i_1}, \ldots, x_{i_k}) & \text{if } R \in \rho, (\theta(x_{i_1}), \ldots, \theta(x_{i_k})) \in \mathbf{R} \\
\mathbf{DB}, \theta \models x_i = x_j & \text{if } \theta(x_i) = \theta(x_j) \\
\mathbf{DB}, \theta \models \varphi \wedge \psi & \text{if } \mathbf{DB}, \theta \models \varphi \text{ and } \mathbf{DB}, \theta \models \psi \\
\mathbf{DB}, \theta \models \neg\varphi & \text{if not } \mathbf{DB}, \theta \models \varphi \\
\mathbf{DB}, \theta \models \exists x_i.\varphi & \text{if } \mathbf{DB}, \theta[x_i \mapsto v] \models \varphi \text{ for some } v \in \mathbf{D}
\end{array}
$$

Equivalences and Syntactic Sugar

Boolean Equivalences

- $\neg(\neg\varphi_1) \equiv \varphi_1$
- $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$
- $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
- $\varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$
- $\ldots$

First-order Equivalences

- $\forall x.\varphi \equiv \neg\exists x.\neg\varphi$

Relational Calculus

Definition (Queries)

A *query* in the relational calculus is a set comprehension of the form

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

Definition (Query Answers)

An *answer* to a *query* $\{(x_1, \ldots, x_k) \mid \varphi\}$ over **DB** is the **relation**

$$\{(\theta(x_1), \ldots, \theta(x_k)) \mid \mathbf{DB}, \theta \models \varphi\},$$

where $\{x_1, \ldots, x_k\} = FV(\varphi)^\dagger$.

†*FV* denotes the *free variables* of φ.

On Formulas

Definition (Free Variables)

The *free variables* of a formula φ, written $FV(\varphi)$, are defined as follows:

$$
\begin{array}{lcl}
FV(R(x_{i_1},\ldots,x_{i_k})) & \equiv & \{x_{i_1},\ldots,x_{i_k}\} \\
FV(x_i = x_j) & \equiv & \{x_i, x_j\} \\
FV(\varphi \wedge \psi) & \equiv & FV(\varphi) \cup FV(\psi) \\
FV(\neg\varphi) & \equiv & FV(\varphi) \\
FV(\exists x_i.\varphi) & \equiv & FV(\varphi) - \{x_i\}
\end{array}
$$

A formula that has no free variables expresses is called a *sentence*.

Example

Find pairs of emp-s working for the same boss!

Q: $\{(x_1, x_2) \mid \exists y_1, y_2, z.\text{EMP}(x_1, y_1, z) \wedge \text{EMP}(x_2, y_2, z)\}$
A: $\{(\mathit{Sue}, \mathit{Fred}), \ldots\}$

because:

1. $\textbf{EMP}, \{x_1 \mapsto \mathit{Sue}, y_1 \mapsto \text{CS}, z \mapsto \mathit{Bob}, \ldots\} \models \text{EMP}(x_1, y_1, z)$
2. $\textbf{EMP}, \{x_2 \mapsto \mathit{Fred}, y_2 \mapsto \text{CO}, z \mapsto \mathit{Bob}, \ldots\} \models \text{EMP}(x_2, y_2, z)$
3. $\textbf{EMP}, \{x_1 \mapsto \mathit{Sue}, y_1 \mapsto \text{CS}, x_2 \mapsto \mathit{Fred}, y_2 \mapsto \text{CO}, z \mapsto \mathit{Bob}, \ldots\}$
   $\models \text{EMP}(x_1, y_1, z) \wedge \text{EMP}(x_2, y_2, z)$
4. $\textbf{EMP}, \{x_1 \mapsto \mathit{Sue}, x_2 \mapsto \mathit{Fred}, \ldots\}$
   $\models \exists y_1, y_2, z.\text{EMP}(x_1, y_1, z) \wedge \text{EMP}(x_2, y_2, z)$

Emp Table

EMP

name	dept	boss
Sue	CS	*Bob*
Fred	CO	*Bob*
Bob	PM	*Mark*
John	PM	*Mark*
Jim	CS	*Fred*
Eve	CS	*Fred*
Sue	PM	*Sue*

Sample Queries

Over numbers (with addition and multiplication):

- list all composite numbers
- list all prime numbers

Over the bibliography database:

- list all publications
- list titles of all publications
- list titles of all books
- list all publications without authors
- list (pairs of) coauthor names
- list titles of publications written by a single author

How do we ask Questions (and understand Answers)?

Find the *neutral element* (of addition)!

Question: $\{(x) \mid \text{PLUS}(x, x, x)\}$

Answer: $\{(0)\}$

but shouldn't the query really be

$$\{(x) \mid \forall y.\text{PLUS}(x, y, y) \wedge \text{PLUS}(y, x, y)\} \qquad (*)$$

Idea

$(*)$ is the same as $\{(x) \mid \forall y.\text{PLUS}(x, y, y)\}$

because PLUS is *commutative*

is the same as $\{(x) \mid \text{PLUS}(x, x, x)\}$

because PLUS is *monotone*

$\Rightarrow$ Laws of Arithmetic for Natural Numbers

Addition Table

PLUS		
0	0	0
0	1	1
1	0	1
0	2	0
2	0	2
	...	
0	5	5
	...	
1	4	5
	...	
2	3	5
	...	
	...	

Laws a.k.a. Integrity Constraints

Idea

What must be always true for the natural numbers (i.e., for PLUS)?

- addition is commutative

$$\forall x, y, z.\text{PLUS}(x, y, z) \rightarrow \text{PLUS}(y, x, z)$$
$$(\neg\exists x, y, z.\text{PLUS}(x, y, z) \wedge \neg\text{PLUS}(y, x, z))$$

- addition is a (relational representation of a) binary function

$$\forall x, y, z_1, z_2.\text{PLUS}(x, y, z_1) \wedge \text{PLUS}(x, y, z_2) \rightarrow z_1 = z_2$$
$$(\neg\exists x, y, z_1, z_2.\text{PLUS}(x, y, z_1) \wedge \text{PLUS}(x, y, z_2) \wedge \neg(z_1 = z_2))$$

- addition is a total function

$$\forall x, y.\exists z.\text{PLUS}(x, y, z)$$

- addition is monotone in both arguments (harder), etc., etc.

Laws a.k.a. Integrity Constraints for Employees

Idea

Integrity constraints
$\Rightarrow$ *yes/no conditions that must be true in every valid database instance.*

- Every Boss is an Employee

$$\forall x, y, z.\mathrm{EMP}(x, y, z) \rightarrow \exists u, w.\mathrm{EMP}(z, u, w)$$

- Every Boss manages a unique Department

$$\forall x_1, x_2, y_1, y_2, z.\mathrm{EMP}(x_1, y_1, z) \wedge \mathrm{EMP}(x_2, y_2, z) \rightarrow y_1 = y_2$$

- No Boss cannot have another Employee serving as their Boss

$$\forall x, y, z.\mathrm{EMP}(x, y, z) \rightarrow \mathrm{EMP}(z, y, z)$$

Integrity Constraints

A relational *signature* captures only the structure of relations.

Idea

Valid database instances satisfy additional integrity constraints.

- Values of a particular attribute belong to a prescribed *data type*.
- Values of attributes are unique among tuples in a relation (*keys*).
- Values appearing in one relation must also appear in another relation (*referential integrity*).
- Values cannot appear simultaneously in certain relations (*disjointness*).
- Values in certain relation must appear in at least one of another set of relations (*coverage*).
- ...

Example Revisited (Bibliography)

Typing Constraints / Domain Contraints

- Author id's are integers.
- Author names are strings.

Uniqueness of Values / Identification (keys)

- Author id's are unique and determine author names.
- Publication id's are unique as well.
- Articles can be identified by their publication id.
- Articles can also be identified by the publication id of the collection they have appeared in and their starting page number.

Referential Integrity / Foreign Keys

- Books, journals, proceedings and articles are publications.
- The components of a WROTE tuple must be an author and a publication.

Example Revisited (cont.)

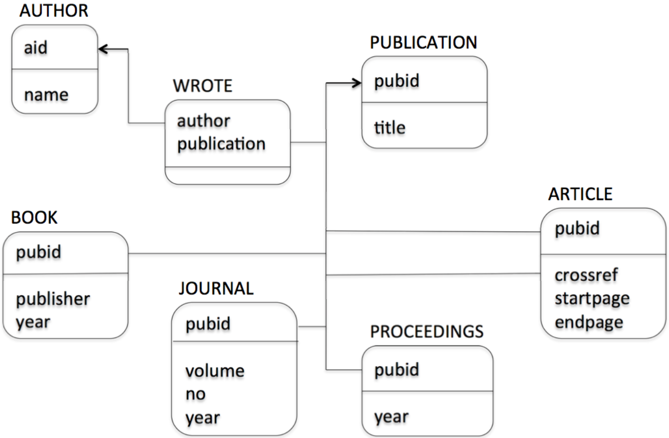
Disjointness

- Books are different from journals.
- Books are also different from proceedings.

Coverage

- Every publication is a book or a journal or a proceedings or an article.
- Every article appears in a journal or in a proceedings.

Example Revisited (cont.)



Views and Integrity Constraints

Idea

Answers to queries can be used to define derived relations (views)
$\Rightarrow$ *extension of a DB schema*

- subtraction, complement, ...
- *collection*-style publication, editor, ...

In general, a view is an integrity constraint of the form

$$\forall x_1, \ldots, x_k . R(x_1, \ldots, x_k) \leftrightarrow \varphi$$

for R a new relation name and $x_1, \ldots, x_k$ free variables of φ.

Database Instances and Integrity Constraints

Definition (Relational Database Schema)

A *relational database schema* is a signature ρ and a (finite) set of integrity constraints Σ over ρ.

Definition

A relational database instance **DB** (over a schema ρ) *conforms to a schema* Σ (written **DB** $\models \Sigma$) if and only if **DB**$, \theta \models \varphi$ for any integrity constraint $\varphi \in \Sigma$ and any valuation θ.

Story so far...

1. databases $\Leftrightarrow$ relational structures
2. queries $\Leftrightarrow$ set comprehensions with formulas in First-Order logic
3. integrity constraints $\Leftrightarrow$ closed formulas in FO logic

... so is there anything new here?

$\Rightarrow$YES: database instances must be finite

Unsafe Queries

- $\{(y) \mid \neg\exists x.\mathtt{author}(x, y)\}$
- $\{(x, y, z) \mid \mathtt{book}(x, y, z) \vee \mathtt{proceedings}(x, y)\}$
- $\{(x, y) \mid x = y\}$

$\Rightarrow$ we want only queries with finite answers (over finite databases).

Definition (Domain-independent Query)

A query $\{(x_1, \ldots, x_k) \mid \varphi\}$ is *domain-independent* if

$$\mathbf{DB}_1, \theta \models \varphi \iff \mathbf{DB}_2, \theta \models \varphi$$

for any pair of database instances $\mathbf{DB}_1 = (\mathbf{D_1}, =, \mathbf{R_1}, \ldots, \mathbf{R_k})$ and $\mathbf{DB}_2 = (\mathbf{D_2}, =, \mathbf{R_1}, \ldots, \mathbf{R_k})$ and all θ.

Theorem

Answers to domain-independent queries contain only values that exist in $\mathbf{R_1}, \ldots, \mathbf{R_k}$ (the active domain).

Domain-independent + finite database $\Rightarrow$ "safe"

Safety and Query Satisfiability

Theorem

Satisfiability[1] *of first-order formulas is undecidable;*

- *co-r.e. in general*
- *r.e for finite databases*

Proof.

Reduction from PCP (see Abiteboul *et. al.* book, p.122-126). □

Theorem

Domain-independence of first-order queries is undecidable.

Proof.

φ is satisfiable iff $\{(x, y) \mid (x = y) \wedge \varphi\}$ is not domain-independent. □

[1] Is there a database for which the answer is non-empty?

Range-restricted Queries

Definition (Range restricted formulas)

A formula φ is *range restricted* when, for φ_i that are also range restricted, φ has the form

$$
\begin{array}{ll}
R(x_{i_1}, \ldots, x_{i_k}), & \\
\varphi_1 \wedge \varphi_2\,, & \\
\varphi_1 \wedge (x_i = x_j) & (\{x_i, x_j\} \cap FV(\varphi_1) \neq \emptyset), \\
\exists x_i.\varphi_1 & (x_i \in FV(\varphi_1)), \\
\varphi_1 \vee \varphi_2 & (FV(\varphi_1) = FV(\varphi_2)), \text{ or} \\
\varphi_1 \wedge \neg\varphi_2 & (FV(\varphi_2) \subseteq FV(\varphi_1)).
\end{array}
$$

Theorem

Range-restricted $\Rightarrow$ *Domain-independent.*

Domain Independent v.s. Range-restricted

Do we lose expressiveness by restricting to Range-restricted queries?

Theorem

Every domain-independent query *can be written equivalently as a* range restricted *query.*

Proof.

1. restrict every variable in φ to *active domain*,
2. express the active domain using a *unary query* over the database instance.

$\square$

Computational Properties

- Evaluation of every query terminates
  $\Rightarrow$ relational calculus is not *Turing complete*
- **Data Complexity** in the size of the database, for a *fixed* query.
  $\Rightarrow$ in PTIME
  $\Rightarrow$ in LOGSPACE
  $\Rightarrow$ AC_0 (constant time on polynomially many CPUs in parallel)
- **Combined complexity**
  $\Rightarrow$ in PSPACE
  $\Rightarrow$ can express NP-hard problems (encode SAT)

Query Evaluation vs. Theorem Proving

Query Evaluation

Given a query $\{(x_1, \ldots, x_k) \mid \varphi\}$ and a finite database instance **DB** find all answers to the query.

Query Satisfiability

Given a query $\{(x_1, \ldots, x_k) \mid \varphi\}$ determine whether there is a (finite) database instance **DB** for which the answer is non-empty.

- much harder (undecidable) problem
- can be solved for fragments of the query language

Query Equivalence and DB Schema

Do we ever need the power of *theorem proving*?

Definition (Query Subsumption)

A query $\{(x_1, \ldots, x_k) \mid \varphi\}$ *subsumes* a query $\{(x_1, \ldots, x_k) \mid \psi\}$ with respect to a database schema Σ if

$$\{(\theta(x_1), \ldots, \theta(x_k)) \mid \mathbf{DB}, \theta \models \psi\} \subseteq \{(\theta(x_1), \ldots, \theta(x_k)) \mid \mathbf{DB}, \theta \models \varphi\}$$

for every database **DB** such that $\mathbf{DB} \models \Sigma$.

- *necessary* for query simplification
- equivalent to proving

$$\left(\bigwedge_{\phi_i \in \Sigma} \phi_i\right) \rightarrow (\forall x_1, \ldots x_k . \psi \rightarrow \varphi)$$

- undecidable in general; decidable for fragments of relational calculus

What queries cannot be expressed in RC?

Note

RC is not Turing-complete
$\Rightarrow$ *there must be computable queries that cannot be written in RC.*

Built-in Operations

- ordering, arithmetic, string operations, etc.

Counting/Aggregation

- cardinality of sets (*parity*)

Reachability/Connectivity/...

- *paths in a graph (binary relation)*

Model extensions: Incompleteness/Inconsistency

- tuples with *unknown* (but existing) values
- incomplete relations and *open world assumption*
- conflicting information (e.g., from different data sources)