The Relational Model
Spring 2018

School of Computer Science
University of Waterloo

Databases CS348
Outline

1 Introduction by Example
2 The Relational Model
3 Integrity Constraints
4 Safety and Finiteness
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\}

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

why?

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. \text{PLUS}(x,y,z) \land \text{PLUS}(z,y,x)\}

Answer: \{(0,0), (1,0), \ldots\}

answer depends on the content (instance) of PLUS!

Find the neutral element (of addition)!

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

Answer: \{(0)\}

Addition Table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
How do we ask Questions (and understand Answers)?

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

**Question:** \( \{(x, y) \mid x + y = 5\} \)
How do we ask Questions (and understand Answers)?

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

Question: \( \{(x, y) | x + y = 5\} \)

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

... but but but why? (explain this to a 6 year old!)
How do we ask Questions (and understand Answers)?

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

<table>
<thead>
<tr>
<th>PLUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
</tr>
<tr>
<td>0 1 1</td>
</tr>
<tr>
<td>1 0 1</td>
</tr>
<tr>
<td>0 2 2</td>
</tr>
<tr>
<td>2 0 2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0 5 5</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>1 4 5</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>2 3 5</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**Question:** \( \{(x, y) \mid \text{PLUS}(x, y, 5)\} \)

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

why? because \((0, 5, 5), \text{etc.}, \) appear in PLUS!
How do we ask Questions (and understand Answers)?

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

Question: \( \{(x, y) \mid \text{PLUS}(x, y, 5)\} \)
Answer: \( \{(0, 5), (1, 4), (2, 3), (3, 2), (4, 1), (5, 0)\} \)

why? because \((0, 5, 5), \text{etc.}, \text{appear in PLUS!}\)

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

Question: \( \{(x, y) \mid \exists z. \text{PLUS}(x, y, z) \land \text{PLUS}(z, y, x)\} \)
Answer: \( \{(0, 0), (1, 0), \ldots\} \)

Addition Table

<table>
<thead>
<tr>
<th>PLUS</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

...
How do we ask Questions (and understand Answers)?

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

Question: \{ (x, y) \mid \text{PLUS}(x, y, 5) \}
Answer: \{ (0, 5), (1, 4), (2, 3), (3, 2), (4, 1), (5, 0) \}

why? 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. \text{PLUS}(x, y, z) \land \text{PLUS}(z, y, x) \}
Answer: \{ (0, 0), (1, 0), \ldots, (5, 5) \}

\ldots answer depends on the content (instance) of PLUS!

Addition Table

<table>
<thead>
<tr>
<th>PLUS</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>
How do we ask Questions (and understand Answers)?

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

Question: \{ (x, y) \mid \text{PLUS}(x, y, 5) \}
Answer: \{ (0, 5), (1, 4), (2, 3), (3, 2), (4, 1), (5, 0) \}

why? 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. \text{PLUS}(x, y, z) \land \text{PLUS}(z, y, x) \}
Answer: \{ (0, 0), (1, 0), \ldots, (5, 5) \}

\ldots answer depends on the content (instance) of PLUS!

Find the neutral element (of addition)!

Question: \{ (x) \mid \text{PLUS}(x, x, x) \}
Answer: \{ (0) \}

Addition Table

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How do we ask Questions about Employees?

Find *all employees who work for “Bob”*!

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

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

*why? because \(\{(\text{Sue}, \text{CS}), (\text{Bob}, \text{CO})\}\), etc., appear in EMP!*

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

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

**A:** \(\{(\text{Sue}, \text{Bob}), (\text{Fred}, \text{John}), (\text{Jim}, \text{Eve})\}\)

← is that all?

Find employees who are their own bosses!

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

**A:** \(\{(\text{Sue}), (\text{Bob})\}\)

---

**Employee Table**

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Boss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>CS</td>
<td>Bob</td>
</tr>
<tr>
<td>Bob</td>
<td>CO</td>
<td>Bob</td>
</tr>
<tr>
<td>Fred</td>
<td>PM</td>
<td>Mark</td>
</tr>
<tr>
<td>John</td>
<td>PM</td>
<td>Mark</td>
</tr>
<tr>
<td>Jim</td>
<td>CS</td>
<td>Fred</td>
</tr>
<tr>
<td>Eve</td>
<td>CS</td>
<td>Fred</td>
</tr>
<tr>
<td>Sue</td>
<td>PM</td>
<td>Sue</td>
</tr>
</tbody>
</table>
How do we ask Questions about Employees?

Find all employees who work for “Bob”!

Question: \{ (x, y) \mid \text{EMP}(x, y, \text{Bob}) \} \\
Answer: \{ (\text{Sue}, \text{CS}), (\text{Bob}, \text{CO}) \}
Find all employees who work for “Bob”!

Question: \{ (x, y) \mid \text{EMP}(x, y, Bob) \}  
Answer: \{ (Sue, CS), (Bob, CO) \}  

why? because (Sue, CS, Bob), etc., appear in EMP!
How do we ask Questions about Employees?

Find all employees who work for “Bob”!

Question: \{ (x, y) \mid EMP(x, y, Bob) \}
Answer: \{ (Sue, CS), (Bob, 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. EMP(x_1, y_1, z) \land EMP(x_2, y_2, z) \}
A: \{ (Sue, Bob), (Fred, John), (Jim, Eve) \}

Employee Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Boss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>CS</td>
<td>Bob</td>
</tr>
<tr>
<td>Bob</td>
<td>CO</td>
<td>Bob</td>
</tr>
<tr>
<td>Fred</td>
<td>PM</td>
<td>Mark</td>
</tr>
<tr>
<td>John</td>
<td>PM</td>
<td>Mark</td>
</tr>
<tr>
<td>Jim</td>
<td>CS</td>
<td>Fred</td>
</tr>
<tr>
<td>Eve</td>
<td>CS</td>
<td>Fred</td>
</tr>
<tr>
<td>Sue</td>
<td>PM</td>
<td>Sue</td>
</tr>
</tbody>
</table>
How do we ask Questions about Employees?

Find **all employees who work for “Bob”!**

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

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

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

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

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

**A:** \( \{(Sue, Bob), (Fred, John), (Jim, Eve)\} \quad \leftarrow \text{is that all?} \)
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, CS), (Bob, 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) \land \text{EMP}(x_2, y_2, z) \}
A: \{ (Sue, Bob), (Fred, John), (Jim, Eve) \} ← is that all?

Find employees who are their own bosses!

Q: \{ (x) \mid \exists y. \text{EMP}(x, y, x) \}
A: \{ (Sue), (Bob) \}
Outline

1. Introduction by Example
2. The Relational Model
3. Integrity Constraints
4. Safety and Finiteness
The Relational Model

Idea

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

Components:

- **Universe**
  - a set of values $\mathbf{D}$ with equality ($\equiv$)

- **Relation**
  - schema: name $R$, arity $k$ (the number of attributes)
  - instance: a relation $R \subseteq \mathbf{D}^k$.

- **Database**
  - schema: finite set of relation schemes
  - instance: a relation $R_i$ for each $R_i$

**Notation**

*Signature*: $\rho = (R_1, \ldots, R_n)$

*Instance*: $\text{DB} = (\mathbf{D}, \equiv, R_1, \ldots, R_n)$
Examples of Relational Structures a.k.a. Databases

- the integer numbers with addition and multiplication:
  \[ \rho = (\text{plus, times}) \]
  \[ \text{DB} = (\mathbb{Z}, =, \text{plus, times}) \]
- a Bibliography Database
- ...
Example: Bibliography

Relations (signatures) used in examples:

- `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)`

⇒ names of attributes will be important later (for SQL)
Example (sample instance)

\[
\begin{align*}
\text{author} & = \{ (1, \text{John}), (2, \text{Sue}) \} \\
\text{wrote} & = \{ (1, 1), (1, 4), (2, 3) \} \\
\text{publication} & = \{ (1, \text{Mathematical Logic}), \\
& \quad (3, \text{Trans. Databases}), \\
& \quad (2, \text{Principles of DB Syst.}), \\
& \quad (4, \text{Query Languages}) \} \\
\text{book} & = \{ (1, \text{AMS}, 1990) \} \\
\text{journal} & = \{ (3, 35, 1, 1990) \} \\
\text{proceedings} & = \{ (2, 1995) \} \\
\text{article} & = \{ (4, 2, 30, 41) \}
\end{align*}
\]
Example (tabular form)

<table>
<thead>
<tr>
<th>author</th>
<th>wrote</th>
</tr>
</thead>
<tbody>
<tr>
<td>aid</td>
<td>author</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>Sue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>pubid</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

⇒ that’s why relations are often called “tables”.

(University of Waterloo)
Simple (Atomic) “Truth”

Idea

*Relationships between objects (tuples) that are present in an instance are true, relationships absent are false.*
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”:
  \[(1, \text{John}) \in \text{author};\]

- “Mathematical Logic” is a publication:
  \[(1, \text{Mathematical Logic}) \in \text{publication};\]
  Moreover it is a book published by “AMS” in “1990”:
  \[(1, \text{AMS}, 1990) \in \text{book};\]

- “John” wrote “Mathematical Logic”:
  \[(1, 1) \in \text{wrote};\]

- “John” has **NOT** written “Trans. Databases”:
  \[(1, 3) \notin \text{wrote};\]

- etc.
Idea 1: use variables to collect answers

\( \text{author}(x, y) \) asks for all valuations \( [x \mapsto a, y \mapsto b, \ldots] \)

such that the pair \((a, b) \in \text{author}\)
Queries

**Idea 1:** use *variables* to collect answers

\[
\text{author}(x, y) \text{ asks for all valuations } [x \mapsto a, y \mapsto b, \ldots] \text{ such that the pair } (a, b) \in \text{author}
\]

**Idea 2:** build more complex queries from simpler ones using...

**Logical connectives:**
- Conjunction (and): \( \text{author}(x, y) \land \text{wrote}(x, z) \)
- Disjunction (or): \( \text{author}(x, y) \lor \text{publication}(x, y) \)
- Negation (not): \( \neg \text{author}(x, y) \)

**Quantifiers:**
- Existential (there is...): \( \exists x. \text{author}(x, y) \)
Relational Calculus: Syntax

Idea

Complex statements about truth can be formulated using the language of first-order logic.

Definition (Syntax)

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

$$\varphi ::= R_i(x_{i_1}, \ldots, x_{i_k}) \mid x_i = x_j \mid \varphi \land \varphi \mid \exists x_i.\varphi \mid \varphi \lor \varphi \mid \neg \varphi$$

- conjunctive formulas
- positive formulas
- first-order formulas
First-order Variables and Valuations

How do we interpret variables?

**Definition (Valuation)**

A *valuation* is a function

\[ \theta : \{ x_1, x_2, \ldots \} \rightarrow D \]

that maps *variable names* to values in the universe.
How do we interpret variables?

Definition (Valuation)

A valuation is a function

\[ \theta : \{ x_1, x_2, \ldots \} \rightarrow D \]

that maps variable names to values in the universe.

Idea

Answers to queries \( \Leftrightarrow \) valuations to free variables that make the formula true with respect to a database.
**Complete Semantics**

**Definition**

The *truth* of formulas is defined with respect to

1. a **database instance**=DB=(D, =, R, S, ...), and
2. a **valuation** θ : \{x_1, x_2, ...\} → D

as follows:

\[
\begin{align*}
\text{DB, } \theta &\models R(x_{i_1}, ... , x_{i_k}) \quad \text{if } R \in \rho, (\theta(x_{i_1}), ... , \theta(x_{i_k})) \in R \\
\text{DB, } \theta &\models x_i = x_j \quad \text{if } \theta(x_i) = \theta(x_j) \\
\text{DB, } \theta &\models \varphi \land \psi \quad \text{if } \text{DB, } \theta \models \varphi \text{ and } \text{DB, } \theta \models \psi \\
\text{DB, } \theta &\models \neg \varphi \quad \text{if not } \text{DB, } \theta \models \varphi \\
\text{DB, } \theta &\models \exists x_i. \varphi \quad \text{if } \text{DB, } \theta[x_i \mapsto v] \models \varphi \text{ for some } v \in D
\end{align*}
\]
Complete Semantics

Definition

The *truth* of formulas is defined with respect to

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

as follows:

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

Definition

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

\[
\{(\theta(x_1), \ldots, \theta(x_k)) \mid DB, \theta \models \varphi\}
\]

where \( \{x_1, \ldots, x_k\} = FV(\varphi) \).
Example

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

Q: \{ (x_1, x_2) | \exists y_1, y_2, z. \text{EMP}(x_1, y_1, z) \land \text{EMP}(x_2, y_2, z) \}\n
A: \{ (Sue, Fred), \ldots \}\n
because:

1. \text{EMP}, [x_1 \mapsto Sue, y_1 \mapsto CS, z \mapsto Bob, \ldots] \models \text{EMP}(x_1, y_1, z)
2. \text{EMP}, [x_2 \mapsto Fred, y_2 \mapsto CO, z \mapsto Bob, \ldots] \models \text{EMP}(x_2, y_2, z)
3. \text{EMP}, [x_1 \mapsto Sue, y_1 \mapsto CS, x_2 \mapsto Fred, y_2 \mapsto CO, z \mapsto Bob, \ldots] \models \text{EMP}(x_1, y_1, z) \land \text{EMP}(x_2, y_2, z)
4. \text{EMP}, [x_1 \mapsto Sue, x_2 \mapsto Fred, \ldots] \models \exists y_1, y_2, z. \text{EMP}(x_1, y_1, z) \land \text{EMP}(x_2, y_2, z)
Sample Queries

over numbers (with addition and multiplication):

- list all composite numbers
- list all prime numbers
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
Equivalences and Syntactic Sugar

Boolean Equivalences

- \( \neg (\neg \varphi_1) \equiv \varphi_1 \)
- \( \varphi_1 \lor \varphi_2 \equiv \neg (\neg \varphi_1 \land \neg \varphi_2) \)
- \( \varphi_1 \rightarrow \varphi_2 \equiv \neg \varphi_1 \lor \varphi_2 \)
- \( \varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \rightarrow \varphi_2) \land (\varphi_2 \rightarrow \varphi_1) \)
- ...  

First-order Equivalences

- \( \forall x. \varphi \equiv \neg \exists x. \neg \varphi \)
Outline

1 Introduction by Example
2 The Relational Model
3 Integrity Constraints
4 Safety and Finiteness
How do we ask Questions (and understand Answers)?

Find the *neutral element* (of addition)!

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

**Answer:** $\{(0)\}$

### Addition Table

<table>
<thead>
<tr>
<th>PLUS</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

\[\{(x) \mid \forall y. \text{PLUS}(x, y, y) \land \text{PLUS}(y, x, y)\}\]  \(\ast\)

---

Addition Table

<table>
<thead>
<tr>
<th>PLUS</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

(University of Waterloo)  The Relational Model  21 / 38
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) \land \text{PLUS}(y, x, y) \}  

(*)

Idea

(*) is the same as \{ (x) \mid \forall y. \text{PLUS}(x, y, y) \}
Find the *neutral element* (of addition)!

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

Answer: \{ (0) \}

but shouldn’t the query really be
\[
\{ (x) \mid \forall y. \text{PLUS}(x, y, y) \land \text{PLUS}(y, x, y) \}
\] (*)

Idea

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

because \text{PLUS} is *commutative*
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

\( \{(x) \mid \forall y. \text{PLUS}(x, y, y) \land \text{PLUS}(y, x, y)\} \)  

\((*)\)

**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*

\( \Rightarrow \text{Laws of Arithmetic for Natural Numbers} \)
Laws a.k.a. Integrity Constraints

Idea

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

- addition is commutative

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

- addition is a total function

- addition is monotone in both arguments (harder), etc., etc.
Laws a.k.a. Integrity Constraints

Idea

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

- Addition is commutative
  \[ \forall x, y, z. \text{PLUS}(x, y, z) \rightarrow \text{PLUS}(y, x, z) \]

- Addition is a (relational representation of a) binary function
  \[ \forall x, y, z_1, z_2. \text{PLUS}(x, y, z_1) \land \text{PLUS}(x, y, z_2) \rightarrow z_1 = z_2 \]

- 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

Idea

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

- addition is commutative
  \[ \forall x, y, z. \text{PLUS}(x, y, z) \rightarrow \text{PLUS}(y, x, z) \]
  \[ (\neg \exists x, y, z. \text{PLUS}(x, y, z) \land \neg \text{PLUS}(y, x, z)) \]

- addition is a (relational representation of a) binary function
  \[ \forall x, y, z_1, z_2. \text{PLUS}(x, y, z_1) \land \text{PLUS}(x, y, z_2) \rightarrow z_1 = z_2 \]
  \[ (\neg \exists x, y, z_1, z_2. \text{PLUS}(x, y, z_1) \land \text{PLUS}(x, y, z_2) \land \neg (z_1 = z_2)) \]

- 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* ⇒ *yes/no queries that must be true in every valid database instance.*

- Every Boss is an Employee

- Every Boss manages a unique Department

- No Boss cannot have another Employee serving as their Boss
Laws a.k.a. Integrity Constraints for Employees

**Idea**

*Integrity constraints* ⇒ *yes/no queries that must be true in every valid database instance.*

- Every Boss is an Employee
  \[ \forall x, y, z. \text{EMP}(x, y, z) \rightarrow \exists u, w. \text{EMP}(z, u, w) \]

- Every Boss manages a unique Department

- No Boss cannot have another Employee serving as their Boss
Laws a.k.a. Integrity Constraints for Employees

**Idea**

*Integrity constraints* ⇒ *yes/no queries that must be true in every valid database instance.*

- Every Boss is an Employee
  \[ \forall x, y, z. \text{EMP}(x, y, z) \rightarrow \exists u, w. \text{EMP}(z, u, w) \]

- Every Boss manages a unique Department
  \[ \forall x_1, x_2, y_1, y_2, z. \text{EMP}(x_1, y_1, z) \land \text{EMP}(x_2, y_2, z) \rightarrow y_1 = y_2 \]

- No Boss cannot have another Employee serving as their Boss
Laws a.k.a. Integrity Constraints for Employees

Idea

*Integrity constraints* ⇒ *yes/no queries that must be true in every valid database instance.*

- Every Boss is an Employee
  \[ \forall x, y, z. \text{EMP}(x, y, z) \rightarrow \exists u, w. \text{EMP}(z, u, w) \]

- Every Boss manages a unique Department
  \[ \forall x_1, x_2, y_1, y_2, z. \text{EMP}(x_1, y_1, z) \land \text{EMP}(x_2, y_2, z) \rightarrow y_1 = y_2 \]

- No Boss cannot have another Employee serving as their Boss
  \[ \forall x, y, z. \text{EMP}(x, y, z) \rightarrow \text{EMP}(z, y, z) \]
Integrity Constraints

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

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

Uniqueness of values/Keys

- Author id’s are unique and determine author names.
- Publication id’s are unique as well.
- Articles are identified by their id and the id of a collection they have appeared in.

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 different from “proceedings”.

Coverage

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

(University of Waterloo)
Idea

Answers to queries can be used to define derived relations (views) ⇒ extension of a DB schema
Views and Integrity Constraints

Idea

Answers to queries can be used to define derived relations (views) ⇒ extension of a DB schema

- subtraction, complement, . . .
- collection-style publication, editor, . . .
Views and Integrity Constraints

Idea

Answers to queries can be used to define derived relations (views) ⇒ extension of a DB schema

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

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

\[ \forall x_1, \ldots, x_k. R(x_1, \ldots, x_k) \iff \varphi \]

for \( R \) a new relation name and \( x_1, \ldots, x_k \) free variables of \( \varphi \).
Definition (Database Schema)

Let $\rho$ be a signature. A database schema is a (finite) set of integrity constraints $\Sigma$ over $\rho$.

Definition

A database instance $\text{DB}$ (over a schema $\rho$) conforms to a schema $\Sigma$ (written $\text{DB} \models \Sigma$) if and only if $\text{DB}, \theta \models \varphi$ for any integrity constraint $\varphi \in \Sigma$ and any valuation $\theta$. 
Outline

1. Introduction by Example
2. The Relational Model
3. Integrity Constraints
4. Safety and Finiteness
Story so far...

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

⇒ YES: database instances must be finite
Story so far...

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

... so is there anything new here?
Story so far...

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

... so is there anything new here?

⇒ **YES**: database instances must be finite
Unsafe Queries

- \{ (y) \mid \neg \exists x. \text{author}(x, y) \} 
- \{ (x, y, z) \mid \text{book}(x, y, z) \lor \text{proceedings}(x, y) \} 
- \{ (x, y) \mid x = y \}
Unsafe Queries

- \{ (y) \mid \neg \exists x. \text{author}(x, y) \}\}
- \{ (x, y, z) \mid \text{book}(x, y, z) \lor \text{proceedings}(x, y) \}\}
- \{ (x, y) \mid x = y \}\}

⇒ we want only queries with finite answers (over finite databases).
Unsafe Queries

- \{ (y) | \neg \exists x. \text{author}(x, y) \} \\
- \{ (x, y, z) | \text{book}(x, y, z) \vee \text{proceedings}(x, y) \} \\
- \{ (x, y) | x = y \}

⇒ we want only queries with finite answers (over finite databases).

Definition (Domain-independent Query)

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

\[ \text{DB}_1, \theta \models \varphi \iff \text{DB}_2, \theta \models \varphi \]

for any pair of database instances \( \text{DB}_1 = (D_1, =, R_1, \ldots, R_k) \) and \( \text{DB}_2 = (D_2, =, R_1, \ldots, R_k) \) and all \( \theta \).
Unsafe Queries

- \{ (y) \mid \neg \exists x. \text{author}(x, y) \} \\
- \{ (x, y, z) \mid \text{book}(x, y, z) \lor \text{proceedings}(x, y) \} \\
- \{ (x, y) \mid x = y \}

\implies \text{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

\[ \text{DB}_1, \theta \models \varphi \iff \text{DB}_2, \theta \models \varphi \]

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

**Theorem**

Answers to domain-independent queries contain only values that exist in \( R_1, \ldots, R_k \) (the active domain).
Unsafe Queries

- \( \{ (y) \mid \neg \exists x.\text{author}(x, y) \} \)
- \( \{ (x, y, z) \mid \text{book}(x, y, z) \lor \text{proceedings}(x, y) \} \)
- \( \{ (x, y) \mid x = y \} \)

\( \Rightarrow \) 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

\[
\text{DB}_1, \theta \models \varphi \iff \text{DB}_2, \theta \models \varphi
\]

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

**Theorem**

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

Domain-independent + finite database ⇒ “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).
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.**

\(\varphi\) is satisfiable iff \(\{(x, y) \mid (x = y) \land \varphi\}\) is not domain-independent.

\(^1\)Is there a database for which the answer is non-empty?
Definition (Range restricted formulas)

A formula $\varphi$ is *range restricted* when, for $\varphi_i$ that are also range restricted, $\varphi$ has the form

$$R(x_{i_1}, \ldots, x_{i_k}),$$

$$\varphi_1 \land \varphi_2,$$

$$\varphi_1 \land (x_i = x_j) \quad (\{x_i, x_j\} \cap FV(\varphi_1) \neq \emptyset),$$

$$\exists x_i. \varphi_1 \quad (x_i \in FV(\varphi_1)),$$

$$\varphi_1 \lor \varphi_2 \quad (FV(\varphi_1) = FV(\varphi_2)),$$

$$\varphi_1 \land \neg \varphi_2 \quad (FV(\varphi_2) \subseteq FV(\varphi_1)).$$
Range-restricted Queries

Definition (Range restricted formulas)

A formula $\varphi$ is *range restricted* when, for $\varphi_i$ that are also range restricted, $\varphi$ has the form

$$R(x_{i_1}, \ldots, x_{i_k}),$$

$$\varphi_1 \land \varphi_2,$$

$$\varphi_1 \land (x_i = x_j) \quad (\{x_i, x_j\} \cap FV(\varphi_1) \neq \emptyset),$$

$$\exists x_i. \varphi_1 \quad (x_i \in FV(\varphi_1)),$$

$$\varphi_1 \lor \varphi_2 \quad (FV(\varphi_1) = FV(\varphi_2)), \text{ or}$$

$$\varphi_1 \land \neg \varphi_2 \quad (FV(\varphi_2) \subseteq FV(\varphi_1)).$$

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 $\varphi$ to *active domain,*
2. express the active domain using a *unary query* over the database instance.
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*?
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 \( \Sigma \) if

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

for every database \( DB \) such that \( DB \models \Sigma \).

- 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

(University of Waterloo)
What queries cannot be expressed in RC?

**Note**

*RC is not Turing-complete*  
⇒ *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*)
What queries cannot be expressed in RC?

Note

*RC is not Turing-complete*

⇒ *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)