How do we ask Questions (and understand Answers)?

In the beginning ...

Set comprehension syntax for questions:
\[ \{(x_1, \ldots, x_k) \mid \langle \text{condition} \rangle \} \]

Answers:

All \textit{k-tuples} of \textit{values} that satisfy \langle \text{condition} \rangle.
How do we ask Questions (and understand Answers)?

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

**Question:** \{ \( (x, y) \mid x + y = 5 \) \}
**Answer:** \{ (0, 5), (1, 4), (2, 3), (3, 2), (4, 1), (5, 0) \}

... but 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:** \{ \( (x, y) \mid \exists z. \text{PLUS}(x, y, z) \land \text{PLUS}(z, y, x) \) \}
**Answer:** \{ (0, 0), (1, 0), ..., (5, 5) \}

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

Find the neutral element (of addition)!

**Question:** \{ \( (x) \mid \text{PLUS}(x, x, x) \) \}
**Answer:** \{ (0) \}
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: \( \{(Sue), (Bob)\} \)
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 $R \subseteq D^k$.

- **Database**
  - signature: finite set $\rho$ of predicate names
  - instance: a relation $R_i$ for each $R_i$

**Notation**

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

*Instance*: $DB = (D, =, R_1, \ldots, R_n)$
Examples of Relational Databases

- The integers, with addition and multiplication:
  \[ \rho = (\text{PLUS}, \text{TIMES}) \]
  \[ \text{DB} = (\mathbb{Z}, =, \text{PLUS}, \text{TIMES}) \]

- 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)

⇒ identifiers, called *attributes*, label columns (needed for SQL)
# A Bibliography Relational Database Instance

Relations (also called tables):

<table>
<thead>
<tr>
<th>Relation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUTHOR</strong></td>
<td>{(1, John), (2, Sue)}</td>
</tr>
<tr>
<td><strong>WROTE</strong></td>
<td>{(1, 1), (1, 4), (2, 3)}</td>
</tr>
<tr>
<td><strong>PUBLICATION</strong></td>
<td>{(1, Mathematical Logic), (3, Trans. Databases), (2, Principles of DB Syst.), (4, Query Languages)}</td>
</tr>
<tr>
<td><strong>BOOK</strong></td>
<td>{(1, AMS, 1990)}</td>
</tr>
<tr>
<td><strong>JOURNAL</strong></td>
<td>{(3, 35, 1, 1990)}</td>
</tr>
<tr>
<td><strong>PROCEEDINGS</strong></td>
<td>{(2, 1995)}</td>
</tr>
<tr>
<td><strong>ARTICLE</strong></td>
<td>{(4, 2, 30, 41)}</td>
</tr>
</tbody>
</table>
A Common Visualization for Relational Databases

<table>
<thead>
<tr>
<th>AUTHOR</th>
<th>WROTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>aid</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>Sue</td>
</tr>
<tr>
<td></td>
<td></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>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
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”:
  \[(1, \text{John}) \in \text{AUTHOR};\]
- “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”:
  \[(1, 3) \not\in \text{WROTE};\]
- etc.
Query Conditions

Idea 1: use *variables* to generalize conditions

\[ \text{AUTHOR}(x, y) \text{ will be true of any valuation } \{ x \mapsto a, y \mapsto b, \ldots \} \]

exactly when the pair \((a, b) \in \text{AUTHOR}\)

Idea 2: build more complex conditions 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...):** \( \exists x. \text{author}(x, y) \)
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

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

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

How do we *interpret* variables?

Definition (Valuation)

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

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

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

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

Idea

*Answers to queries $\iff$ valuations to free variables that make the formula true with respect to a database.*
Complete Semantics for Conditions

Definition

The *truth* of a formula $\varphi$ is defined with respect to

1. a database instance $DB = (D, =, R_1, R_2, \ldots)$, and
2. a valuation $\theta : \{x_1, x_2, \ldots\} \rightarrow D$

as follows:

- $DB, \theta \models R(x_{i_1}, \ldots, x_{i_k})$ if $R \in \rho, (\theta(x_{i_1}), \ldots, \theta(x_{i_k})) \in R$
- $DB, \theta \models x_i = x_j$ if $\theta(x_i) = \theta(x_j)$
- $DB, \theta \models \varphi \land \psi$ if $DB, \theta \models \varphi$ and $DB, \theta \models \psi$
- $DB, \theta \models \neg \varphi$ if not $DB, \theta \models \varphi$
- $DB, \theta \models \exists x_i. \varphi$ if $DB, \theta[x_i \mapsto v] \models \varphi$ for some $v \in D$
Equivalences and Syntactic Sugar

Boolean Equivalences

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

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

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

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

\^\text{\( FV \) denotes the free variables of \( \varphi \).}
The free variables of a formula $\varphi$, written $FV(\varphi)$, are defined as follows:

$$FV(R(x_{i_1}, \ldots, x_{i_k})) \equiv \{x_{i_1}, \ldots, x_{i_k}\}$$
$$FV(x_i = x_j) \equiv \{x_i, x_j\}$$
$$FV(\varphi \land \psi) \equiv FV(\varphi) \cup FV(\psi)$$
$$FV(\neg \varphi) \equiv FV(\varphi)$$
$$FV(\exists x_i. \varphi) \equiv FV(\varphi) - \{x_i\}$$

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) | \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

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) \land \text{PLUS}(y, x, y)\} \quad (*)

**Idea**

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

because \text{PLUS} is *commutative*

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

because \text{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
  \[ \forall x, y, z. \text{PLUS}(x, y, z) \rightarrow \text{PLUS}(y, x, z) \]
  \[ (\neg \exists x, y, z. \text{PLUS}(x, y, z) \land \neg \text{PLUS}(y, x, z)) \]

- 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 conditions 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

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 Constraints

- 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.)

- **AUTHOR**
  - aid
  - name

- **BOOK**
  - pubid
  - publisher
  - year

- **JOURNAL**
  - pubid
  - volume
  - no
  - year

- **PROCEEDINGS**
  - pubid

- **PUBLICATION**
  - pubid
  - title

- **ARTICLE**
  - pubid
  - crossref
  - startpage
  - endpage

  - WROTE
    - author
    - publication
**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  
\[ \forall x_1, \ldots, x_k. R(x_1, \ldots, x_k) \leftrightarrow \varphi \]

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

Definition (Relational Database Schema)
A relational database schema is a signature $\rho$ and a (finite) set of integrity constraints $\Sigma$ over $\rho$.

Definition
A relational 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 $\theta$. 
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\}

⇒ 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).

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?
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
  ⇒ relational calculus is not *Turing complete*

- **Data Complexity** in the size of the database, for a *fixed* query.
  ⇒ in PTIME
  ⇒ in LOGSPACE
  ⇒ $AC_0$ (constant time on polynomially many CPUs in parallel)

- **Combined complexity**
  ⇒ in PSPACE
  ⇒ can express NP-hard problems (encode SAT)
Query Evaluation vs. Theorem Proving

Query Evaluation

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

Query Satisfiability

Given a query \( \{(x_1, \ldots, x_k) | \varphi \} \) 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 \( \Sigma \) 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
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)