The Relational Model

David Toman

School of Computer Science
University of Waterloo

Introduction to Databases CS348

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

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 Structures/Databases

Components:

Universe

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

Relation

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

Database

- schema: finite set of relation schemes
- instance: a relation $\mathbf{R_i}$ for each R_i

Notation

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

Examples of Relational Structures

- the integer numbers with addition and multiplication:

$$(\mathbf{Z}, =, \texttt{plus}, \texttt{times})$$

- a Bibliography Database:

$$(\mathbf{D}, =, \texttt{author}, \texttt{wrote}, \texttt{publication}, \ldots)$$

  for $\mathbf{D}$ the set of strings and ints
- an (abstraction of) OS Kernel:

$$(\mathbf{D}, =, \texttt{process}, \texttt{file}, \texttt{waits-for}, \ldots)$$

  for $\mathbf{D}$ the set of strings, ints, and addresses
- ...

Examples of Relational Structures

- the integer numbers with addition and multiplication:

$$(\mathbf{Z}, =, \mathtt{plus}, \mathtt{times})$$

- a Bibliography Database:

$$(\mathbf{D}, =, \mathtt{author}, \mathtt{wrote}, \mathtt{publication}, \ldots)$$

  for $\mathbf{D}$ the set of strings and ints

- an (abstraction of) OS Kernel:

$$(\mathbf{D}, =, \mathtt{process}, \mathtt{file}, \mathtt{waits\text{-}for}, \ldots)$$

  for $\mathbf{D}$ the set of strings, ints, and addresses

- ...

Examples of Relational Structures

- the integer numbers with addition and multiplication:

$$(\mathbf{Z}, =, \texttt{plus}, \texttt{times})$$

- a Bibliography Database:

$$(\mathbf{D}, =, \texttt{author}, \texttt{wrote}, \texttt{publication}, \ldots)$$

  for $\mathbf{D}$ the set of strings and ints
- an (abstraction of) OS Kernel:

$$(\mathbf{D}, =, \texttt{process}, \texttt{file}, \texttt{waits-for}, \ldots)$$

  for $\mathbf{D}$ the set of strings, ints, and addresses
- ...

Example: Bibliography

Relations (in the `cs348` database) 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)

$$
\begin{array}{rcll}
\texttt{author} & = & \{ & (1, \text{John}), (2, \text{Sue}) \quad \} \\
\texttt{wrote} & = & \{ & (1, 1), (1, 4), (2, 3) \quad \} \\
\texttt{publication} & = & \{ & (1, \text{Mathematical Logic}), \\
 & & & (3, \text{Trans. Databases}), \\
 & & & (2, \text{Principles of DB Syst.}), \\
 & & & (4, \text{Query Languages}) \quad \} \\
\texttt{book} & = & \{ & (1, \text{AMS}, 1990) \quad \} \\
\texttt{journal} & = & \{ & (3, 35, 1, 1990) \quad \} \\
\texttt{proceedings} & = & \{ & (2, 1995) \quad \} \\
\texttt{article} & = & \{ & (4, 2, 30, 41) \quad \}
\end{array}
$$

Example (tabular form)

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

$\Rightarrow$ that's why relations are often called "tables".

Example: OS Kernel

- schema:

```
process(pid, ppid, uid, stime, time, cmd)
file(fd, pid, name, dentry)
waits-for(pid, wpid)
```

- instance—depends on what's going on in the kernel:

```
UID          PID  PPID  STIME     TIME CMD
root           1     0  Sep16 00:00:00 init [2]
root           2     1  Sep16 00:00:00 [ksoftirqd/0]
root           3     1  Sep16 00:00:00 [watchdog/0]
root           4     1  Sep16 00:00:00 [events/0]
root           5     1  Sep16 00:00:00 [khelper]
root           6     1  Sep16 00:00:00 [kthread]
root           9     6  Sep16 00:00:00 [kblockd/0]
...
david      28382  5790  10:10 00:00:00 bash
david      29903 28382  10:10 00:00:00 ps -ef
```

Example: OS Kernel

- schema:

```
process(pid, ppid, uid, stime, time, cmd)
file(fd, pid, name, dentry)
waits-for(pid, wpid)
```

- instance—depends on what's going on in the kernel:

```
UID          PID  PPID  STIME      TIME CMD
root           1     0  Sep16 00:00:00 init [2]
root           2     1  Sep16 00:00:00 [ksoftirqd/0]
root           3     1  Sep16 00:00:00 [watchdog/0]
root           4     1  Sep16 00:00:00 [events/0]
root           5     1  Sep16 00:00:00 [khelper]
root           6     1  Sep16 00:00:00 [kthread]
root           9     6  Sep16 00:00:00 [kblockd/0]
...
david      28382  5790  10:10 00:00:00 bash
david      29903 28382  10:10 00:00:00 ps -ef
```

Atomic Queries

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

Atomic Queries

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

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

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

First-order Variables and Valuations

How do we *interpret* variables?

Definition (Valuation)

A valuation is a function

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

that maps *variable names* to values in the universe.

Idea

Answers to queries ⇔ valuations to free variables that make the formula true with respect to a database.

First-order Variables and Valuations

How do we *interpret* variables?

Definition (Valuation)

A valuation is a function

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

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** $D = (\mathbf{D}, =, \mathbf{R}, \mathbf{S}, \ldots)$, and
2. a **valuation** $\theta : \{x_1, x_2, \ldots\} \to \mathbf{D}$

as follows:

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

Definition

An **answer** to a query φ over a database D is a **relation**:

$$\varphi(D) = \{\theta_{|\mathrm{FV}(\varphi)} : D, \theta \models \varphi\}$$

Complete Semantics

Definition

The *truth* of formulas is defined with respect to

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

as follows:

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

Definition

An **answer** to a query φ over a database D is a **relation**:

$$\varphi(D) = \{\theta_{|\mathrm{FV}(\varphi)} : D,\theta \models \varphi\}$$

Examples of Queries

over integers:

- list all pairs of numbers that add to 10
- 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

w.r.t. the OS kernel:

- list pairs of PIDs of processes that share an open file
- list PIDs of processes do not have any open file
- list pairs of PIDs of processes that wait for each other

Examples of Queries

over integers:

- list all pairs of numbers that add to 10
- 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

w.r.t. the OS kernel:

- list pairs of PIDs of processes that share an open file
- list PIDs of processes do not have any open file
- list pairs of PIDs of processes that wait for each other

Examples of Queries

over integers:

- list all pairs of numbers that add to 10
- 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

w.r.t. the OS kernel:

- list pairs of PIDs of processes that share an open file
- list PIDs of processes do not have any open file
- list pairs of PIDs of processes that wait for each other

Equivalences and Syntactic Sugar

Boolean Equivalences

- $\neg(\neg\varphi) \equiv \varphi$
- $\varphi \lor \psi \equiv \neg(\neg\varphi \land \neg\psi)$
- $\varphi \to \psi \equiv \neg\varphi \lor \psi$
- $\varphi \leftrightarrow \psi \equiv (\varphi \to \psi) \land (\psi \to \varphi)$
- $\ldots$

First-order Equivalences

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

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 (Integers)

What must be always true for the *integers*?

- `plus` is commutative
- `plus` is a (relation representing a) binary function
- `plus` and `times` obey the distributive law

Idea

Integrity constraints

$\Rightarrow$ *closed queries true for every valid database instance.*

Example Revisited (Integers)

What must be always true for the *integers*?

- `plus` is commutative
- `plus` is a (relation representing a) binary function
- `plus` and `times` obey the distributive law

Idea

Integrity constraints
$\Rightarrow$ *closed queries true for every valid database instance.*

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”.

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

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

where R is a fresh relation name and $x_1, \ldots, x_k$ are free variables of φ.

$\Rightarrow$ why not simply an arbitrary formula containing R?

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

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

where R is a fresh relation name and $x_1, \ldots, x_k$ are free variables of φ.

$\Rightarrow$ why not simply an arbitrary formula containing R?

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

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

where R is a fresh relation name and $x_1, \ldots, x_k$ are free variables of φ.

$\Rightarrow$ why not simply an arbitrary formula containing R?

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

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

where R is a fresh relation name and $x_1, \ldots, x_k$ are free variables of φ.

$\Rightarrow$ why not simply an arbitrary formula containing R?

Story so far...

1. databases ⇔ relational structures
2. queries ⇔ 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

Story so far. . .

1. databases ⇔ relational structures
2. queries ⇔ 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

Story so far...

1. databases $\Leftrightarrow$ relational structures
2. queries $\Leftrightarrow$ 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

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

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

Definition (Domain-independent Query)

A query is *domain-independent* if its answer depends only in the database instance (but not on the domain $\mathbf{D}$).

Theorem

Answers to domain-independent queries contain only values that exist in the database instance (or as a constant in the query).

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

Unsafe Queries

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

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

Definition (Domain-independent Query)

A query is *domain-independent* if its answer depends only in the database instance (but not on the domain $\mathbf{D}$).

Theorem

Answers to domain-independent queries contain only values that exist in the database instance (or as a constant in the query).

Domain-independent + finite database $\Rightarrow$ “safe”

Unsafe Queries

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

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

Definition (Domain-independent Query)

A query is *domain-independent* if its answer depends only in the database instance (but not on the domain $\mathbf{D}$).

Theorem

Answers to domain-independent queries contain only values that exist in the database instance (or as a constant in the query).

Domain-independent + finite database $\Rightarrow$ “safe”

Unsafe Queries

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

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

Definition (Domain-independent Query)

A query is *domain-independent* if its answer depends only in the database instance (but not on the domain $\mathbf{D}$).

Theorem

Answers to domain-independent queries contain only values that exist in the database instance (or as a constant in the query).

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

Unsafe Queries

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

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

Definition (Domain-independent Query)

A query is *domain-independent* if its answer depends only in the database instance (but not on the domain $\mathbf{D}$).

Theorem

Answers to domain-independent queries contain only values that exist in the database instance (or as a constant in the query).

Domain-independent + finite database $\Rightarrow$ “safe”

Safety and Query Satisfiability

Theorem

Satisfiability[a] of first-order formulas is undecidable;

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

---

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

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 $x = y \wedge \varphi$ is not domain-independent. □

Safety and Query Satisfiability

Theorem

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

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

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

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 $x = y \wedge \varphi$ is not domain-independent. □

Range-restricted Queries

Definition (Range restricted queries)

$$
\begin{array}{lll}
Q & ::= & R(x_{i_1}, \ldots, x_{i_k}) \\
 & & \left.\begin{array}{l} Q \wedge Q \\ Q \wedge x_i = x_j \\ \exists x_i . Q \end{array}\right\} \quad x_i, x_j \in FV(Q) \\
 & & \left.\begin{array}{l} Q_1 \vee Q_2 \\ Q_1 \wedge \neg Q_2 \end{array}\right\} \quad FV(Q_1) = FV(Q_2)
\end{array}
$$

Theorem

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

Range-restricted Queries

Definition (Range restricted queries)

$$
Q ::= \begin{array}{ll}
R(x_{i_1}, \ldots, x_{i_k}) & \\
\left.\begin{array}{l} Q \land Q \\ Q \land x_i = x_j \\ \exists x_i . Q \end{array}\right\} & x_i, x_j \in FV(Q) \\
\left.\begin{array}{l} Q_1 \lor Q_2 \\ Q_1 \land \neg Q_2 \end{array}\right\} & FV(Q_1) = FV(Q_2)
\end{array}
$$

Theorem

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

Safe v.s. Range-restricted

Do we lose expressiveness by using 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* (constants present in the database instance),
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 φ and a finite database instance D find substitutions θ such that $D, \theta \models \varphi$.

Query Satisfiability

Given a query φ find whether there is a (finite) database instance D and a substitution θ such that $D, \theta \models \varphi$.

- much harder (undecidable) problem
- can be solved for fragments

(conjunctive/positive) queries

Query Equivalence and Schema

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

Definition (Query Subsumption)

φ *subsumes* ψ (with respect to a schema Σ) if

$$\varphi(D) \subseteq \psi(D)$$

for every database D such that $D \models \Sigma$.

- *necessary* for query simplification
- equivalent to proving

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

- undecidable in general; decidable for fragments of RC

Query Equivalence and Schema

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

Definition (Query Subsumption)

φ *subsumes* ψ (with respect to a schema Σ) if

$$\varphi(D) \subseteq \psi(D)$$

for every database D such that $D \models \Sigma$.

- *necessary* for query simplification
- equivalent to proving

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

- undecidable in general; decidable for fragments of RC

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/Transitive closure

- *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 sources)

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/Transitive closure

- *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 sources)

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/Transitive closure

- *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 sources)

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/Transitive closure

- *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 sources)