THE RELATIONAL MODEL

University of Waterloo

List of Slides

The Relational Model

Entities:

(names of) real-world objects
⇒ “customer”, “bank”, “account”, . . .
⇒ “author”, “publication”, . . .
⇒ “1” (the number 1), . . .

Relationships:

relationships between entities
⇒ “has an account”, “located at”,. . .
⇒ “wrote”, “author”, “book”, . . .

Attributes:

properties of relationships (and, in turn, entities)
⇒ “address”, “account #”, “balance”,. . .
⇒ “name”, “title”, . . .

A collection of *data* describing a real-world situation.

Entities are (generally) “real world” things we want to store information about. In the database they’re encoded using an identifier.

Relationships tie entities with their other entities.

Attributes form the “observable” properties of relationships; note that two tuples with identical values of attributes cannot be distinguished and are considered to be the same; that’s why everyone has a SSN (SIN in Canada, CPR# in Denmark).

In the above example, “author” is an entity. However, to associate author names and id’s with the entities, we can use an “author” relationship.

Relations and Databases

Idea: All information is organized in relations.

A RELATION has

- a name
- a schema

  $\Rightarrow$ a list of **attribute names** describing individual elements of **tuples** (and their types)
- an instance

  $\Rightarrow$ a finite set of tuples of (identifiers of) **entities**

The attribute name denotes what role an entity plays in a relation(ship).

Example

publications					
pubid	title	author	type	year	cref
ChSa98	Logics for Databases and Information Systems	Chomicki, Jan	book	1998	
ChTo98	Temporal Logic in Information Systems	Chomicki, Jan	chapter	1998	ChSa98
ChTo98	Temporal Logic in Information Systems	Toman, David	chapter	1998	ChSa98
JLP0398	Journal of Logic Programming		journal	1998	
ChTo98a	Datalog with Integer Periodicity Constraints	Toman, David	article		JLP0398
DOOD97	International Conference on Deductive and Object-Oriented Databases		proceedings	1998	
Tom97	Point-Based Temporal Extension of Temporal SQL	Toman, David	paper		DOOD97

Another Example

author	
aid	Name
1	Toman, David
2	Chomicki, Jan
...	...

wrote	
AUTHOR	PUBLICAT
1	ChTo98
1	ChTo98a
1	Tom97
2	ChTo98
2	ChTo98a
...	...

publication	
pubid	title
ChTo98	Temporal Logic in Information Systems
JLP-3-98	Journal of Logic Programming
ChTo98a	Datalog with Integer Periodicity Constraints
DOOD97	International Conference on Deductive and Object...
Tom97	Point-Based Temporal Extension of Temporal SQL
...	...

What does it mean?

Membership (presence) of
a tuple $(a_1, \ldots, a_k)$ in a relation R means that
the relationship R **holds among** $(a_1, \ldots, a_k)$
in the given database instance D.

We write $(a_1, \ldots, a_k) \in R^D$ or $D \models R(a_1, \ldots, a_k)$.

For example:

$$\texttt{DB} \models \texttt{wrote}(1, \text{ChTo98})$$

means that, in the database **DB**, a publication identified by **ChTo98** has been **written** by author with author id **1**.

Database *modification* (update) changes the
relationships that are true/false in the database
(for which a tuple is present/absent).

Example Database

Tables (in the **`DB`** 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)
```

What can we do with it?

Idea: Combine simple questions about what is true in the database into more complex questions using logical connectives.

- Yes/No queries:

  $\Rightarrow$ in **DB**, does the publication **ChTo98** have a title **“Temporal Logic in Information Systems”**?

  $\Rightarrow$ . . . we could explore the database by
  asking (lots) of Yes/No queries.

  $\Rightarrow$ not efficient
  $\Rightarrow$ you can only verify statements

  $\Rightarrow$ . . . not quite satisfactory:
  we’d like to retrieve **all** elements for which some statement holds “in one shot”.

Variables and Answers

Idea: we use **logical variable(s)** to range over values in the database. The values of these variable(s) form new relations.

- variables and answers:
  - $\Rightarrow$ Yes/No queries are **parametrized** by variables.
  - $\Rightarrow$ An **answer** is the set of all values that,
    when substituted for the variables,
    make the query true in D.
  - $\Rightarrow$ retrieve all x such that $Q(x)$ is true in D:

$$\{x : D \models Q(x)\}$$

  - $\Rightarrow$ queries may contain many parameters (variables).

. . . its just like Logic

Don’t panic!

- Statements about numbers:

  $\Rightarrow 1 < 3$ is true (in $\mathbf{R}$)
  $\Rightarrow 2 + 2 = 0$ is false in $\mathbf{Z}$ (but true in $\mathbf{Z_4}$)
  $\rightarrow \exists x . x^2 = -1$ (false in $\mathbf{R}$, true in $\mathbf{C}$)

- Formation of sets:

  $\Rightarrow \{x : x > 5\}$ the set of all numbers bigger than 5
  $\Rightarrow \{x : x^2 - 3x + 2 = 0\}$ the roots of the equation
  $\Rightarrow \{(x, y) : x = sin(t) \wedge y = cos(t), t \in \mathbf{R}\}$ a circle

Variables and Quantifiers

Often we don’t care what value(s) a variable contains , as long as there is at least one such value.

Moreover, we often do not want such values to appear in the answer.

- Questions of the form “there is a value for x such that $Q(x)$ is true in the database”:

  $$\{y : \textbf{DB} \models \exists x.Q(x,y)\}$$

  note that Q may contain other variables (e.g., y).

Selecting Things

If we need to retrieve only answers that satisfy some additional predicate we use **built-in** predicates.

- equality: $x = y$, $x = 5$, $x =$ 'string', etc.
- other predicates (depending on the data type, e.g., $x > 5$ for numerical values)

The built-in predicates also define tables, but are not *stored* in the database:

$\Rightarrow$ the DBMS computes the value of the predicate.
$\Rightarrow$ the arguments have to be **bound** in a base relation.

Combining Relations

We can ask about what is true in several relations:

- conjunction ($\wedge$) of formulas (usually atomic)
- use of shared variables:

  $\Rightarrow R(x, y) \wedge S(y, z)$

  $\Rightarrow$ forces equality on parts of tuples.
- use of quantification to hide uninteresting variables:

  $\Rightarrow \exists y.R(x, y) \wedge S(y, z)$
- by far the most common pattern in queries

Negation and CWA

Idea: Only relationships **explicitly stored** in the database are **true**.

- everything **not** in the database is **false**:

  $\Rightarrow$ to find out that

  $$\texttt{\textbf{DB}} \models \neg\texttt{\textbf{wrote}}(5, \mathrm{ChTo98})$$

  we simply see that

  $$\texttt{\textbf{DB}} \not\models \texttt{\textbf{wrote}}(5, \mathrm{ChTo98})$$

  i.e., the tuple (5,ChTo98) is not in **`wrote`**.

- Closed World Assumption.

Safety

- **Answers** to a queries have to be **finite**.
- Not completely clear for negation, compare the following two queries:
  - $\Rightarrow$ List the names of all authors (good).
  - $\Rightarrow$ List the names of all non-authors (bad).
- Only **Range-Restricted** queries allowed

  all SQL queries are range-restricted.

Relational Calculus

The language we formulate statements about truth in the database is based on **first-order (predicate) logic** (with few restrictions that guarantee safety):

$$
\begin{array}{rlll}
Q ::= & R(x_1, \ldots, x_k) & & \\
\mid & Q_1 \wedge Q_2 & & \\
\mid & \exists x_i . Q & & \\
& & & \\
\mid & Q \wedge (x_i = x_j) & \text{for } \{x_i, x_j\} \subseteq FV(Q) \\
\mid & Q_1 \wedge \neg Q_2 & \text{for } FV(Q_2) \subseteq FV(Q_1) \\
\mid & Q_1 \vee Q_2 & \text{for } FV(Q_1) = FV(Q_2)
\end{array}
$$

where $FV(Q)$ is the set of **free variables** of Q.

Properties of RC

- Clean semantics

  $\Rightarrow$ Results do **NOT** depend on implementation
  $\Rightarrow$ Efficient implementation (Relational Algebra)

- All queries terminate

  $\Rightarrow$ not Turing complete (in PTIME, LOGSPACE)
  $\Rightarrow$ estimation of execution time possible

- Expressiveness results

  $\Rightarrow$ it is known what questions can be formulated in RC
  sufficient for most busines applications,
  but no transitive closure, cardinality, counting, . . .

Summary

- A **Relational database** stores information about **relationships** between entities in **finite relations**.
  - ⇒ Entities are denoted by constants *which the DBMS does NOT understand*.
  - ⇒ A relationship can be **true**: a tuple is present, or **false**: a tuple is absent (CWA).
- A **query language** allows us to combine simple facts into complex statements:
  - ⇒ The statements may be parametric:
    The result is a set of values that, when substituted for parameters, make the statement true.
  - ⇒ Results of queries are relations themselves.
- We use (safe) **relational calculus** to formulate queries. SQL is just a (strange) syntax for RC.