On the relationship between logic and databases

The case of relational database queries

Maurizio Lenzerini

Sapienza Università di Roma

A.Y. 2024/25

Example of interpretation

Example (of first-order language and interpretation)

Symbols Constants: 1, 2, 3, 4, 5
Predicate symbols: *has-mother*/2, *friends*/2

Domain $\Delta = \{1, 2, 3, 4, 5\}$

Interpretation $\mathcal{I}(1) = 1$, $\mathcal{I}(2) = 2$, $\mathcal{I}(3) = 3$, $\mathcal{I}(4) = 4$, $\mathcal{I}(5) = 5$

$$\mathcal{I}(\textit{has-mother}) = \left\{ \begin{array}{ll} \langle 1, 2 \rangle, & \langle 2, 3 \rangle \\ \langle 3, 4 \rangle, & \langle 4, 5 \rangle \end{array} \right\}$$

$$\mathcal{I}(\textit{friends}) = \left\{ \begin{array}{lll} \langle 1, 2 \rangle, & \langle 2, 1 \rangle, & \langle 3, 4 \rangle, \\ \langle 4, 3 \rangle, & \langle 4, 2 \rangle, & \langle 2, 4 \rangle, \\ \langle 4, 1 \rangle, & \langle 1, 4 \rangle, & \langle 4, 4 \rangle \end{array} \right\}$$

It immediate to see the above first-order intepretation as a relational database, where the domain determines the possible values in the database, each predicate corresponds to a relation, and the extension of each predicate determines the tuples in the corresponding relation.

Database as a first-order interpretation

Let D be a relational database, and let Δ, called the active domain of D, be the set of values stored in D. Since D is finite, Δ is also finite. We often use $adom(D)$ to denote Δ.

Let us define a first-order language L_D with equality (and other "interpreted" predicates) as follows:

- the set of constant symbols is simply Δ (with each value encoded in some way),
- the set of function symbols is empty,
- the set of predicate symbols includes
  - one symbol P/n for each relation P with n columns (where argument i corresponds to the i-th attribute),
  - the equality predicate, plus other predicates: $\neq$, $<$, $\leq$, ...

Database as a first-order interpretation

It is easy to see that the database D is a finite interpretation for L_D such that:

- the domain of such interpretation is Δ, and therefore is finite
- every constant is mapped to itself, and therefore different constants are interpreted as different domain elements (unique name assumption)
- the interpretation function is given by the extension of the various relations in D, where each of them is finite

First-order language and query language

Since the database D plays the role of interpretation for L_D, the formulas of the language L_D can now be evaluated with respect to such an interpretation.

This is the basic idea of "logic as a query language": an open formula in L_D with free variables $x_1, \ldots, x_k$ will correspond to a query that, when evaluated with respect to D, will return the k-tuples of values (each one represented by the corresponding constant) in D that, when assigned to the variables $x_1, \ldots, x_k$, make the formula true.
In other words, the formula defines a relation over the database D, which is the result of the query.

The idea was proposed by E. Codd, the inventor of the relational model. In contrast to the relational algebra, also proposed by Codd, the query language so defined was called relational calculus (with the semantics based on the active domain).

Relational calculus expressions

We will now see, given D, how we will write queries as relational calculus expressions in L_D

A relational calculus expression in L_D is an expression of the form

$$\{(x_1, \ldots, x_k) : \phi(x_1, \ldots, x_k)\}$$

where $\phi(x_1, \ldots, x_k)$ is a first-order formula of L_D with $x_1, \ldots, x_k$ as its free variables.

When applied to a relational database D, this relational calculus expression returns the k-ary relation that consists of all k-tuples $(a_1, \ldots, a_k)$ of constants in D that make the formula true on D.

Relational calculus expressions

Consider the interpretation/database shown at page 2.

Example

The relational calculus expression

$$\{(x, y) : \exists z(\textit{has-mother}(x, z) \wedge \textit{has-mother}(z, y))\}$$

returns the set of all pairs (a, b) such that b is the mother of the mother of a.

Example

The relational calculus expression

$$\{(x) : \exists y(\textit{has-mother}(x, y) \wedge \forall z(\textit{friends}(x, z) \rightarrow \textit{friends}(y, z)))\}$$

returns the set of all objects a all of whose friends are friends of her/his mother.

Relationship between relational algebra and calculus

If Q is a query and D a database, let us denote by $Q(D)$ the result of evaluating Q wrt D.

Theorem (Codd's theorem)

The relational algebra and the relational calculus are "equivalent", i.e.,

- *For every relational algebra expression E there is a relational calculus query F such that for every database D, $E(D) = F(D)$.*
- *For every relational calculus query F there is a relational algebra expression E such that for every database D, $E(D) = F(D)$.*

Relationship between relational algebra and calculus

Theorem (Codd's theorem)

The relational algebra and the relational calculus are "equivalent", i.e.,

- *For every relational algebra expression E there is a relational calculus query F such that for every database D, $E(D) = F(D)$.*
- *For every relational calculus query F there is a relational algebra expression E such that for every database D, $E(D) = F(D)$.*

Exercise: Prove the first item of Codd's theorem. (Hint: by induction on the relational algebra expression).

Exercise

We have a relational database whose schema is Movie(title, director, actor) and Schedule(theater, mtitle), where both attributes, title and mtitle, refer to the title of a movie.

Consider the following queries:

- Which theaters show some movies directed by Tarantino?
- Which theaters do not show any movies directed by Tarantino?
- Which theaters show only movies directed by Tarantino?
- Which theaters show all movies directed by Tarantino?

Express each of the queries above in the three query languages of

- Relational Calculus
- Relational Algebra
- SQL

Exercise

We have a relational database whose schema is Movie(title, director, actor) and Schedule(theater, mtitle), where both attributes, title and mtitle, refer to the title of a movie.

Consider the following queries:

- Which theaters show some movies directed by Tarantino?
  $\{(x) \mid \exists y \exists z\ \mathit{Schedule}(x, y) \wedge \mathit{Movie}(y, \mathit{Tarantino}, z)\}$
- Which theaters do not show any movies directed by Tarantino?
- Which theaters show only movies directed by Tarantino?
- Which theaters show all movies directed by Tarantino?

Exercise

We have a relational database whose schema is Movie(title, director, actor) and Schedule(theater, mtitle), where both attributes, title and mtitle, refer to the title of a movie.

Consider the following queries:

- Which theaters show some movies directed by Tarantino?
  $\{(x) \mid \exists y \exists z\ \mathit{Schedule}(x, y) \wedge \mathit{Movie}(y, \mathit{Tarantino}, z)\}$
- Which theaters do not show any movies directed by Tarantino?
  $\{(x) \mid \neg\exists y \exists w\ \mathit{Schedule}(x, y) \wedge \mathit{Movie}(y, \mathit{Tarantino}, w)\}$
- Which theaters show only movies directed by Tarantino?
- Which theaters show all movies directed by Tarantino?

Exercise

We have a relational database whose schema is Movie(title, director, actor) and Schedule(theater, mtitle), where both attributes, title and mtitle, refer to the title of a movie.

Consider the following queries:

- Which theaters show some movies directed by Tarantino?
  $\{(x) \mid \exists y \exists z\ Schedule(x, y) \wedge Movie(y, Tarantino, z)\}$
- Which theaters do not show any movies directed by Tarantino?
  $\{(x) \mid \neg\exists y \exists w\ Schedule(x, y) \wedge Movie(y, Tarantino, w)\}$
- Which theaters show only movies directed by Tarantino?
  $\{(x) \mid \forall y\ (Schedule(x, y) \rightarrow \exists w\ Movie(y, Tarantino, w))\}$
- Which theaters show all movies directed by Tarantino?

Exercise

We have a relational database whose schema is Movie(title, director, actor) and Schedule(theater, mtitle), where both attributes, title and mtitle, refer to the title of a movie.

Consider the following queries:

- Which theaters show some movies directed by Tarantino?
  $\{(x) \mid \exists y \exists z\ Schedule(x, y) \wedge Movie(y, Tarantino, z)\}$
- Which theaters do not show any movies directed by Tarantino?
  $\{(x) \mid \neg\exists y \exists w\ Schedule(x, y) \wedge Movie(y, Tarantino, w)\}$
- Which theaters show only movies directed by Tarantino?
  $\{(x) \mid \forall y\ (Schedule(x, y) \rightarrow \exists w\ Movie(y, Tarantino, w))\}$
- Which theaters show all movies directed by Tarantino?
  $\{(x) \mid \forall y\ ((\exists w\ Movie(y, Tarantino, w)) \rightarrow Schedule(x, y))\}$

First-order logic and integrity constraints

We recall that an integrity constraints is a condition that the database has to satisfy in order to be coherent with the domain it represents.

It is interesting to observe that logic (in particular, first-order logic) is an ideal language for expressing integrity constraints over a database schema. For examples:

- Key constraint: to specify that the attribute A is a key of the relation $R(A, B, C)$, the following formula can be used: $\forall x \forall y_1 \forall y_2 \forall z_1 \forall z_2 \ (R(x, y_1, z_1) \wedge R(x, y_2, z_2) \rightarrow (y_1 = y_2 \wedge y_2 = z_2))$
- Foreign key constraint: if A is a key of $R(A, B, C)$, then to specify that the attribute E in $Q(D, E)$ is a foreign key of R, the following formula can be used: $\forall x \forall y \ (Q(x, y) \rightarrow \exists w \exists z \ R(y, w, z))$

More generally, logic allows expressing many sophisticated conditions on the database.

First-order logic and integrity constraints

We remind the reader that a schema S is constituted by the alphabet of the database (fixing the schema of each of the relations) and a set of integrity constraints.

If we express integrity constraints in first-order logic, the notion of database is formalized as follows.

Definition

If $S = \langle \Sigma, \Gamma \rangle$ is a relational schema with alphabet Σ and set Γ of integrity constraints, a database D assigning an extension to all the relations in Σ is said to be legal for S if $D \models \Gamma$, i.e., the interpretation D is a model of all the formulas expressing the integrity constraints in Γ.

A database that is legal for S is also called an S-database. In summary, an S-database is a model of the schema S, where S is seen as a logical theory.