On the relationship between logic and databases

The case of relational database queries

Maurizio Lenzerini

Sapienza Università di Roma

A.Y. 2021/22

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,
- 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: $\neq$, $<$, $\leq$, $\ldots$

It is easy to see that 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 constants 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

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

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

Example

The relational calculus expression

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

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, $E(D) = F(D)$.*

Proof of Codd's theorem – from algebra to calculus: 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?
  $\{(x) \mid \exists y \exists z \; Schedule(x, y) \wedge Movie(y, Tarantino, z)\}$
- 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?
  $\{(x) \mid \exists y \exists z \; Schedule(x, y) \wedge Movie(y, Tarantino, z)\}$
- 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?
- 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?
  $\{(x) \mid \forall y\ ((\exists w\ Movie(y, Tarantino, w)) \rightarrow Schedule(x, y))\}$

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.

The complexity of the query evaluation problem

Vardi's taxomonmy (M.Y Vardi, "The Complexity of Relational Query Languages", 1982):

Definition

Let L be a database query language.

- The combined complexity of L is the complexity of the following decision problem: given as input an L-sentence ϕ and a database D, is ϕ true on D? (in symbols, $D \models \phi$?)
- The data complexity of L is the family of the following decision problems P_ϕ, where ϕ is an L-sentence: given as input a database D, $D \models \phi$?
- The query complexity of L is the family of the following decision problems P_D, where D is a database: given as input an L-sentence ϕ, $D \models \phi$?

Complexity of query evaluation in the relational calculus

Definition

- The combined complexity of the relational calculus is PSPACE-complete
- The query complexity of the relational calculus is PSPACE-complete
- The data complexity of the relational calculus is in LOGSPACE (and therefore in PTIME)

Question: Are there interesting sublanguages of relational calculus for which the Query Evaluation Problem are “easier” than the full relational calculus?

Conjunctive queries

Definition

A conjunctive query is a query expressible by a relational calculus formula in prenex normal form (i.e., all quantifiers appear at the beginning of the formula) built from atomic formulas $R(y_1, \ldots, y_n)$, and $\wedge$ and $\exists$ only:

$$\{(x_1, \ldots, x_k) \mid \exists z_1 \ldots \exists z_m \; \chi(x_1, \ldots, x_k, z_1, \ldots, z_m)\}$$

where $\chi(x_1, \ldots, x_k, z_1, \ldots, z_m)\}$ is a conjunction of atomic formulas, called the *body* of the query.

These queries corresponds to relational algebra expressions of the form $PROJ_X(SEL_\gamma(R_1 \times \cdots \times R_n))$, where γ is a conjunction of equality atoms.
They also correspond to the basic form of SQL, namely "SELECT FROM WHERE".

Some notation

- Recall that for a database D, the active domain of D, called $adom(D)$, denotes the values appearing in the relations of D.
- We have observed that a database can be seen as a first-order intepretation. Since the projection of an interpetation function on predicate symbols can be seen as a set of facts (i.e., ground atomic formulas), we can regard a database simply as a set of facts:

$$\{P(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \text{ is a tuple in the relation } P \text{ of } D\}$$

- We often write a conjunction query $\{(x_1, \ldots, x_n) \mid \exists z_1 \ldots \exists z_k \; \gamma_1 \wedge \gamma_2 \wedge \ldots \wedge \gamma_m\}$ of a conjunction query in the simplified form:

$$\{(x_1, \ldots, x_n) \mid \gamma_1, \gamma_2, \ldots, \gamma_m\}$$

The notion of homorphism

Definition (Homomorphism)

Let D and F be two databases over the same relational schema S. A homomorphism $h : D \to F$ is a function from $adom(D)$ to $adom(F)$ such that for every relational symbol P of S and every $(a_1, \ldots, a_m)$, we have that if $(a_1, \ldots, a_m) \in P^D$, then $(h(a_1), \ldots, h(a_m)) \in P^F$.

In what follows, we often write $h((a_1, \ldots, a_m))$ to denote $(h(a_1), \ldots, h(a_m))$.

Homomorphism: example

- $D_1 = \{P_1(a, b), P_2(b, c, d), P_2(c, a, b)\}$
- $D_2 = \{P_1(n, m), P_1(n, n), P_2(m, m, r), P_2(m, n, m)\}$
- h_1 such that $h_1(a) = n, h(b) = m, h(c) = m, h(d) = r$ is a homomorphism from D_1 to D_2
- no homomorphism from D_2 to D_1 exists.

The homorphism problem

Definition (Homomorphism problem)

Given two databases D and F, is there a homomorphism $h : D \to F$?

The homomorphism problem is a fundamental algorithmic problem:

- Satisfiability can be viewed as a special case of it.
- k-Colorability can be viewed as a special case of it.
- Many Artificial Intelligence problems, such as planning, can be viewed as a special case of it.
- Every constraint satisfaction problem can be viewed as a special case of the Homomorphism Problem (Feder and Vardi – 1993).

Duality between databases and conjunctive queries

A Boolean conunctive query can be seen as a database.

Definition (Canonical database of a conjunctive query)

Given a conjunctive query Q, the canonical database of Q is the database D^Q with the variables of Q as active domain and the conjuncts of Q as database facts.

For example, the canonical database of $\{() \mid E(x,y) \wedge E(y,z) \wedge E(z,w)\}$ is constituted by the facts $\{E(x,y), E(y,z), E(z,w)\}$.

Duality between databases and conjunctive queries

A database can be seen as a Boolean conjunctive query.

Definition (Canonical conjunctive query of a database)

Given a database D, the canonical conjunctive query Q^D is the conjunctive query with (a renaming of) the elements in $adom(D)$ as variables and the facts of D as conjuncts.

For example, the canonical conjunctive query of the database
$\{E(a,b) \wedge E(b,c) \wedge E(c,a)\}$ is
$\{() \mid \exists x \exists y \exists z \; E(x,y), E(y,z), E(z,x)\}$.

Complexity of conjunction query evaluation

Theorem

Checking whether a Boolean conjunctive query

$$\{() \mid \exists z_1 \ldots \exists z_m\ \chi(z_1, \ldots, z_m)\}$$

is true with respect to a database is NP-complete in combined complexity.

An NP algorithm for the problem of checking whether a Boolean conjunctive query Q is true wrt D is as follows:

1. guess a function f from $adom(D^Q)$ to $adom(D)$,
2. check in polynomial time if f is a homomorphism from D^Q to D.

Indeed, a function from $adom(D^Q)$ to $adom(D)$, corresponds to an assignment to all quantified variables of Q with values in the database D, and then checking whether such a function is a homomorphism corresponds to checking whether the assignment makes the query Q true wrt D.

Complexity of conjunction query evaluation

Theorem

Checking whether a Boolean conjunctive query

$$\{() \mid \exists z_1 \ldots \exists z_m \ \chi(z_1, \ldots, z_m)\}$$

is true with respect to a database is NP-complete in combined complexity.

The proof of NP-hardness is by reduction to 3-colorability:

- the complete undirected graph with 3 nodes (such nodes are called (b)lue, (g)reen and (r)ed respectively), can be represented as the database $K_3 = \{E(r,b), E(b,r), E(b,g), E(g,b), E(r,g), E(g,r)\}$
- if G is a graph, then the query Q_G associated to G is the Boolean conjunctive query that has as existential quantified variables all the nodes of G, and as body the conjunction of atoms $E(x,y)$ for each edge of G from x to y
- Q_G is true with respect to K_3 if and only if G is 3-colorable.

Query containment and the containment problem

Definition

Given two queries Q_1 and Q_2, we say that Q_1 is contained in Q_2, written as $Q_1 \subseteq Q_2$, if for every database D we have that $Q(D) \subseteq Q_2(D)$.

Definition

The containment problem is the following decision problem: given two queries Q_1 and Q_2, decide whether $Q_1 \subseteq Q_2$.

Note that the containment problem is a logical implication problem in *finite models*.

Indeed, if Q_1 and Q_2 have arity n, then $Q_1 \subseteq Q_2$ if and only if

$$\models_{fin} \forall x_1 \forall x_2 \ldots \forall x_n \; Q_1(x_1, \ldots, x_n) \rightarrow Q_2(x_1, \ldots, x_n)$$

where $\models_{fin} \alpha$ means that α is true in every finite interpretation.

Complexity of conjunctive query containment

Theorem

The containment problem for conjunctive queries, i.e., checking whether $Q_1 \subseteq Q_2$, where Q_1 and Q_2 are conjunctive queries, is an NP-complete problem.

We leave the proof as an exercise. Hint: prove that $Q_1 \subseteq Q_2$ if and only if there is a homomorphism from Q_1^D to Q_2^D.

Unions of conjunctive queries

Definition

A union of conjunctive queries is a query of the form:

$$\{(x_1, \ldots, x_n) \mid \gamma_1 \vee \ldots \vee \gamma_m\}$$

where each γ_i is the body of a conjunction query with free variables $x_1, \ldots, x_n$.

Exercise

Prove the following:

- The complexity of evaluating unions of conjunctive queries is the same as the complexity of evaluating conjunctive queries.
- The complexity of query containment for unions of conjunctive queries is the same as the complexity of query containment for conjunctive queries.