

Ontology and Database Systems: Foundations of Database Systems

Part 1: Databases and Queries

Werner Nutt

Faculty of Computer Science
European Master in Computational Logic

A.Y. 2015/2016

 Freie Universität Bozen
Libera Università di Bolzano
Università Lìedia de Bulsan

Relational Databases: Principles

A database has two parts: **schema** and **instance**

The schema describes *how data is organized*:

- relations with their names, arity, and names and types of attributes
- integrity constraints like key and foreign key constraints, functional dependencies, inclusion dependencies, domain constraints

The instance contains the *actual data*:

- for every relation, there is a relation instance
- the relation instance is a set (multiset?) of tuples of the right arity and type

Often, we ignore types and integrity constraints
Sometimes, we ignore also the attribute names

Example Schema: Students and Courses

Relation schemas

```
Student(sid: INTEGER, sname: STRING, city: STRING, age: INTEGER)
```

```
Course(cid: INTEGER, cname: STRING, faculty: STRING)
```

```
Enrolled(sid: INTEGER, cid: INTEGER, aY: STRING, mark: STRING)
```

Integrity constraints

- Primary keys

```
Student(sid)
```

```
Course(cid)
```

```
Enrolled(sid, cid, aY)
```

- Foreign keys:

```
Enrolled(sid) references Student(sid)
```

```
Enrolled(cid) references Course(cid)
```

Schemas: Formalization

A **relation schema** consists of

- a relation name
- an ordered list of attributes, possibly with types

Abstract notation $R(A_1, \dots, A_n)$, or $R(A_1: \tau_1, \dots, A_n: \tau_n)$

The *arity* of R , written $ary(R)$, is the number of arguments of R

A **database schema** \mathcal{S} consists of

- a *signature* Σ , which is a set of relation schemas
- a set Γ of *integrity constraints* over Σ ,
which may be expressed as formulas in first-order logic (FOL)

Simplified notation: $\mathcal{S} = \{R_1, \dots, R_m\}$, or $\mathcal{S} = \{R_1/n_1, \dots, R_m/n_m\}$,
(i.e., we only mention the names, or the names with their arity)

Exercise: Express the primary and foreign key constraints in the Students and Courses schema by FOL formulas

Domain: Formalization

We assume there is an infinite set of constants **dom**, called the **domain**

When we ignore types, we do not make any assumptions about the constants in **dom**

Otherwise, $\mathbf{dom} = \bigcup_{i=1}^k \tau_i$, where τ_1, \dots, τ_k are the types

Definition

A type τ with an order “ $<$ ” is an *ordered type*. The order “ $<$ ” is

- *dense* if for every $a, b \in \tau$ with $a < b$, there is a $c \in \tau$ such that $a < c < b$
- *discrete* if for every $a, b \in \tau$ with $a < b$, there are at most finitely many c such that $a < c < b$

Example

Consider integers, reals, strings, and booleans.

Which type has a dense and which a discrete ordering?

Relation Instances

Relation R with arity n :

- an instance of R is a finite set of n -tuples over **dom**

Relation R with schema $R(A_1: \tau_1, \dots, A_n: \tau_n)$:

- as before, plus the components of the n -tuples in an instance have to be of the right type

Schema Instances

An **instance of the signature** Σ is a function \mathbf{I} that

- maps every $R \in \Sigma$ to an instance of R , denoted $\mathbf{I}(R)$

Every instance \mathbf{I} of Σ can be seen as a **first-order interpretation/structure** (also denoted \mathbf{I}):

- domain of \mathbf{I} is $\Delta^{\mathbf{I}} = \mathbf{dom}$
- $c^{\mathbf{I}} = c$, for every $c \in \mathbf{dom}$
(proper names, i.e., every constant is interpreted as itself)
- $R^{\mathbf{I}} = \mathbf{I}(R)$

A function \mathbf{I} is an **instance of the schema** $\mathcal{S} = (\Sigma, \Gamma)$ if

- \mathbf{I} is an instance of Σ
- \mathbf{I} satisfies every integrity constraint $\gamma \in \Gamma$ in the sense of first-order logic (FOL)

Logic Programming Perspective

Often an alternate definition of instances is helpful

Definition

- A *fact* over a relation R with arity n is an expression $R(a_1, \dots, a_n)$, where $a_1, \dots, a_n \in \mathbf{dom}$
- A *relation instance* is a finite set of facts over R
- A *signature instance* \mathbf{I} of Σ is a finite set of facts over the relations in Σ

Example

$$\mathbf{I}_{\text{univ}} = \{ \text{Student}(123, \text{Egger}, \text{Bozen}, 25), \text{Student}(777, \text{Hussein}, \text{Dresden}, 23), \\ \text{Course}(104, \text{Programming}, \text{CS}), \text{Course}(106, \text{Databases}, \text{CS}), \\ \text{Course}(217, \text{Optics}, \text{PHYS}) \\ \text{Enrolled}(123, 104, 11/12, \text{fail}), \text{Enrolled}(123, 104, 12/13, \text{fail}), \\ \text{Enrolled}(123, 104, 13/14, \text{pass}), \text{Enrolled}(123, 106, 12/13, \text{pass}), \\ \text{Enrolled}(777, 217, 12/13, \text{pass}) \}$$

Relational Database Queries

A **query** over a schema \mathcal{S} is

- a **function** that maps every instance of \mathcal{S} to a set of tuples such that
 - all tuples have the same length (= arity of the query)
 - tuple values at the same position have the same type
- a **piece of syntax** that defines such a function

Query languages are/should be **declarative**:

- you express what you want to know, not how to compute it
(a query engine analyzes the query and creates an execution plan)

Relational Query Languages

- Theoretical languages
 - Relational Algebra (that's how Codd started it)
 - Relational Calculus (= FOL in essence)
 - Datalog (drops negation, adds recursion)
- Commercial language: SQL
 - = Relational Calculus (at its core)
 - + Relational Algebra
 - + a bit of Datalog (implemented in IBM DB2, Microsoft SQL Server)
 - + aggregates, arithmetic, nulls, . . . , functions, procedures

Relational Calculus Queries

Definition

A **query** in (domain) relational calculus (RelCalc) has the form

$$Q = \{(x_1, \dots, x_n) \mid \phi\}$$

where

- ϕ is a predicate logic formula
- x_1, \dots, x_n are the free variables of ϕ

We say that

- ϕ is the **body** of the query,
- x_1, \dots, x_n are the **output variables**, and
- n is the **arity** of the query.

If the arity is not important, we write \bar{x} instead of x_1, \dots, x_n

We sometimes write Q_ϕ to denote the query defined by ϕ

Reminder on Predicate Logic Formulas

A *term* is a constant or a variable

An *atom* is an expression $R(t_1, \dots, t_n)$ where R is a relation symbol of arity n and t_1, \dots, t_n are terms

A *formula* F is an atom or has the form

- $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, or $(F_1 \rightarrow F_2)$
- $\neg F$
- $(\exists x F)$, $(\forall x F)$

where F , F_1 , F_2 are formulas.

(Operators have the usual precedences.

We drop parentheses that are not needed for the structure of a formula.)

Exercise (once the semantics has been defined):

Show that the logical symbols \wedge , \exists , \neg suffice to express all other symbols

Equality and Built-in Predicates

Sometimes we use also the predicate symbols

“=”, “<”, “≤”, “≠”

Atoms with these symbols are called

- equalities (“=”)
- comparisons (“<”, “≤”)
- disequalities (“≠”)

Clearly, they can only be applied to terms of the same type

Comparisons can only be used for terms of a type that is ordered

Bound and Free Variables

Definition

- An occurrence of a variable x in formula ϕ is *bound* if it is within the scope of a quantifier $\exists x$ or $\forall x$
- An occurrence of a variable in ϕ is *free* iff it is not bound
- A variable of formula ϕ is *free* if it has a free occurrence

Free variables specify the output of a query

Relational Calculus Queries: Semantics

In FOL, the semantics of a formula is defined in terms of *interpretations* and *assignments*. Recall:

- every instance \mathbf{I} defines a first-order interpretation \mathbf{I}
- an assignment is a mapping $\alpha: \mathbf{var} \rightarrow \mathbf{dom}$

There is a classical recursive definition of when an interpretation \mathbf{I} and an assignment α *satisfy* a formula ϕ , written

$$\mathbf{I}, \alpha \models \phi,$$

which we take for granted

Definition

Let $Q = \{(x_1, \dots, x_n) \mid \phi\}$ be a query. We define the **answer** of Q over \mathbf{I} as

$$Q(\mathbf{I}) = \{\alpha(\bar{x}) \mid \mathbf{I}, \alpha \models \phi\}$$

Exercise

Express the following queries over our university schema in Relational Calculus

- Which are the names of students that have passed an exam in CS?
- Which students (given by their id) have never failed an exam in CS?
- Which students (given by their id) have passed the exams for all courses in CS?

Evaluate the expressions over the instance \mathbf{I}_{univ}

Relational Algebra

Expressions E are built up from

- relation symbols R

using the operators

- *union* ($E_1 \cup E_2$), *intersection* ($E_1 \cap E_2$), *set difference* ($E_1 \setminus E_2$), called boolean operators
- *selection* $\sigma_C(E)$
- *projection* $\pi_X(E)$
- *cartesian product* $E_1 \times E_2$
- *join* $E_1 \bowtie_C E_2$
- *attribute renaming* ($\rho_{A \leftarrow B}(E)$)

where C is a condition involving equalities and comparisons between attributes and constants, and X is a set of attributes of E

For an instance \mathbf{I} , an expression E is evaluated as a set of tuples $E(\mathbf{I})$

A **query** is an **expression**

Relational Algebra: Remarks

- An operator not only returns a set of tuples as the result, but also a schema for the result.
- Operators that mention attributes can only be applied to expressions that have that attribute in their schema.
- Boolean operators can only be applied to expressions with the same schema.

Relational Algebra: Examples

What is the meaning of the following queries?

- $\sigma_{\text{city}='Bozen' \wedge \text{age} > 21}(\text{Student})$
- $\pi_{\text{name}, \text{faculty}}(\text{Course})$
- $\pi_{\text{name}}(\text{Course} \bowtie_{\text{Course.cid}=\text{Enrolled.cid}} \text{Enrolled})$
- $\pi_{\text{sid}}(\text{Student}) \setminus \pi_{\text{sid}}(\text{Enrolled})$

Relational Algebra: Exercise

Express the following queries over our university schema in Relational Algebra

- What are the names of the courses for which student Egger has failed an exam?
- Which students have failed an exam for the same course at least twice?
- Which students have never failed an exam in Physics?

Evaluate the expressions over the instance \mathbf{I}_{univ}