course:
Database Systems (NDBI025)
SS2011/12

lecture 7:

Query formalisms for relational model – relational calculus

doc. RNDr. Tomáš Skopal, Ph.D.

Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague

Today’s lecture outline

- relational calculus
  - domain relational calculus
  - tuple relational calculus
  - safe formulas

Relational calculus

- application of first-order calculus (predicate logic) for database querying
- extension by „database“ predicate testing a membership of an element in a relation, defined at two levels of granularity
  - domain calculus (DRC) – variables are attributes
  - tuple calculus (TRC) – variables are tuples (whole elements of relation)
- query result in DRC/TRC is relation
  (and the appropriate schema)

Relational calculus

- the "language"
  - terms – variables and constants
  - predicate symbols
    - standard binary predicates $\{<, >, =, \geq, \leq, \neq\}$
    - „database predicates" (extending the first-order calculus)
  - formulas
    - atomic – $R(t_1, t_2, \ldots)$, where R is predicate symbol and t_i is term
    - complex – expressions that combine atomic or other complex formulas using logic predicates $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
  - quantifiers $\exists$ (existential), $\forall$ (universal)

Domain relational calculus (DRC)

- variables stand for attributes, resp. their values
- database predicate R(x, y, …)
  - R is stands for the name of table the predicate is applied on
  - predicate that for interpreted *x*, *y*, … returns *true* if there is an element in R (table row) with the same values
    - i.e., row membership test
  - predicate scheme (input parameters) is the same as the scheme of relation R, i.e., each parameter *x*, *y*,… is substituted by a (interpreted) variable or constant

Domain relational calculus (DRC)

- database predicate
  - variable and constants in the predicate have an attribute name assigned, determining the value testing, e.g.:
    CINEMA(*NAME_CINEMA* : x , *FILM* : y)
    Then relation schema can be defined as a (unordered) set {*NAME_CINEMA*, *FILM*}.
  - if the list of variables and constants does not contain the attribute names, it is assumed they belong to the attributes given by the relation schema R, e.g.:
    CINEMA(x, y), where <*NAME_CINEMA, FILM*> is the schema if CINEMA – in the following we consider this short notation

Domain relational calculus (DRC)

- the result of query given in DRC is a set of all interpretations of variables and constants in the form of ordered n-tuple, for which the formula of the query is true
  - **{(t_1, t_2, ...) | query formula that includes t_1, t_2, ... }**
    - t_i is either a constant or a **free variable**, i.e., a variable that is not quantified inside the formula
    - the schema of the result relation is defined directly by the names of the free variables
  - e.g., query **{(x, y) | CINEMA(x, y)}** returns relation consisting of all CINEMA elements
  - query **{(x) | CINEMA(x, 'Titanic')}** returns names of cinemas that screen the Titanic film

Domain relational calculus (DRC)

- quantifiers allow to declare (bound) variables that are interpreted within the database predicates
  - formula $\exists$**x R(t_1, t_2, …, x, …)** is evaluated as true if there **exists** domain interpretation of **x** such that n-tuple (t_1, t_2, …, **x**, …) is a member of **R**
  - formula $\forall$**x R(t_1, t_2, …, x, …)** is evaluated as true if **all** domain interpretations of **x** leading to n-tuples (t_1, t_2, …, x, …) are members of **R**
  - e.g., query **{(film) | $\exists$name_cinema CINEMA(name_cinema, film) }** returns names of all films screened **at least** in one cinema

Domain relational calculus (DRC)

- it is important to determine which domain is used of interpretation of variables (both bound and free variables)
  1. domain can be not specified (i.e., interpretation is not limited) – the domain is the whole **universum**

  2. domain is an attribute type – **domain** interpretation

  3. domain is a set of values of a given attribute that exist in the relation on which the interpretation is applied – **actual domain** interpretation

Domain relational calculus (DRC)

- e.g., query **{(film) | ∀name_cinema CINEMA(name_cinema, film) }** could be evaluated differently based on the way variable **name_cinema** (type/domain string) is interpreted
  - if the **universe** is used, the query result is an empty set because the relation CINEMA is surely not infinite also is type-restricted
    - e.g., values in **NAME_CINEMA** will not include 'horse', 125, 'quertyuiop')
  - if the **domain** (attribute type) is used, the query answer will be also empty – still infinite relation CINEMA assumed, containind all strings, e.g., 'horse', 'qwertyuiop', ...
  - if the **actual domain** is used, the query result consists of names of films screened in all cinemas (that are contained in the relation **CINEMA**)

Domain relational calculus (DRC)

- if we implicitly consider interpretation based on the actual domain, we call such limited DRC as **DRC with limited interpretation**

- because schemas often consist of many attributes, we can use simplifying notation of quantification
  - an expression $R(t_1, \ldots, t_i, t_{i+2}, \ldots)$,
    i.e., t_{i+1} is missing,
    is understood as $\exists t_{i+1}\, R(t_1, \ldots, t_i, \mathbf{t_{i+1}}, t_{i+2}, \ldots)$
    - the position of variables then must be clear from the context,
      or strict attribute assignment must be declared
  - e.g., query **{(x) | CINEMA(x)}** is the same as **{(x) | ∃y CINEMA(x, y)}**

Examples – DRC

FILM(NAME_FILM, NAME_ACTOR)   ACTOR(NAME_ACTOR, YEAR_BIRTH)

*In what films **all** the actors appeared?*

$\{(f) \mid FILM(f) \wedge \forall a\ (ACTOR(a) \Rightarrow FILM(f, a))\}$

*Which actor is the **youngest**?*

$\{(a,y) \mid ACTOR(a,y) \wedge \forall a2\ \forall y2\ (ACTOR(a2,y2) \wedge a \neq a2) \Rightarrow y2 < y\}$

or

$\{(a,y) \mid ACTOR(a,y) \wedge \forall a2\ (ACTOR(a2) \Rightarrow \neg\exists y2(ACTOR(a2,y2) \wedge a \neq a2 \wedge y2 > y))\}$

*Which pairs of actors appeared **at least** in one film?*

$\{(a1, a2) \mid ACTOR(a1) \wedge ACTOR(a2) \wedge a1 \neq a2 \wedge$

$\exists f, fa1\ FILM(f, fa1) \wedge (\exists fa2\ FILM(f, fa2) \wedge a1 = fa1 \wedge a2 = fa2)\}$

Evaluation of DRC query

*Which actor is the **youngest**?*

$\{(a,y) \mid ACTOR(a,y) \land \forall a2(ACTOR(a2) \Rightarrow \neg\exists y2\ (ACTOR(a2,y2) \land a \neq a2 \land y2 > y))\}$

universal quantifier = chain of conjunctions

existential quantifier = chain of disjunctions

```
$result = ∅
for each (a,y) do
     if (ACTOR(a,y) and
          (for each a2 do
               if (not ACTOR(a2) or not (for each y2 do
                         if (ACTOR(a2,y2) ∧ a ≠ a2 ∧ y2 > y) = true then return true
                end for
                return false)) = false then return false

          end for
          return true) ) = true then Add (a,y) into $result
end for
```

Tuple relational calculus (TRC)

- almost the same as DRC, the difference is variables/constants are whole elements of relations (i.e., row of tables), i.e., predicate R(**t**) is interpreted as true if (a row) **t** belongs to R
  - the result schema is defined by concatenation of schemas of the free variables (n-tuples)
- to access the attributes within a tuple **t**, a "dot notation" is used
  - e.g., query **{t | CINEMA(t) $\wedge$ t.FILM = 'Titanic'}** returns cinemas which screen the film Titanic
- the result schema could be projected only on a subset of attributes
  - e.g., query **{t[NAME_CINEMA] | CINEMA(t)}**

Examples – TRC

FILM(NAME_FILM, NAME_ACTOR) ACTOR(NAME_ACTOR, YEAR_BIRTH)

Get the pairs of actors of the same age acting in the same film.

$$\{a1, a2 \mid \text{ACTOR}(a1) \land \text{ACTOR}(a2) \land a1.\text{YEAR\_BIRTH} = a2.\text{YEAR\_BIRTH} \\ \land \exists f1, f2\ \text{FILM}(f1) \land \text{FILM}(f2) \land f1.\text{NAME\_FILM} = f2.\text{NAME\_FILM} \\ \land f1.\text{NAME\_ACTOR} = a1.\text{NAME\_ACTOR} \\ \land f2.\text{NAME\_ACTOR} = a2.\text{NAME\_ACTOR}\}$$

*Which films were casted by **all** the actors?*

$$\{film[\text{NAME\_FILM}] \mid \forall actor(\text{ACTOR}(actor) \Rightarrow \\ \exists f(\text{FILM}(f) \land f.\text{NAME\_ACTOR} = actor.\text{NAME\_ACTOR} \land \\ f.\text{NAME\_FILM} = film.\text{NAME\_FILM}))\}$$

Safe formulas in DRC

- unbound interpretation of variables (domain-dependent formulas, resp.) could lead to infinite query results
  - negation: **$\{x \mid \neg R(x)\}$**
    - e.g. $\{j \mid \neg \text{Employee(Name: } j)\}$
  - disjunction: **$\{x,y \mid R(.., x, ...) \vee S(.., y, ...)\}$**
    - e.g. $\{i, j \mid \text{Employee(Name: } i) \vee \text{Student(Name: } j)\}$
  - universal quantifiers lead to an empty set
    $\{x \mid \forall y\ R(x, ..., y)\}$, and generally **$\{x \mid \forall y\ \varphi(x, ..., y)\}$**, where φ does not include disjunctions (implications, resp.)
- even if the query result is finite,
  how to manage **infinite** quantifications in **finite** time?
- the solution is to limit the set of DRC formulas – set of **safe formulas**

Safe formulas in DRC

- to simply avoid infinite quantification ad-hoc, it is good to constrain the quantifiers so that the **interpretation of bound variables is limited to a finite set**
  - using $\exists x\, (R(x) \wedge \varphi(x))$ instead of $\exists x\, (\varphi(x))$
  - using $\forall x\, (R(x) \Rightarrow \varphi(x))$ instead of $\forall x\, (\varphi(x))$
    - by this convention the evaluation is implemented as

      for each x in R // finite enumeration

      instead of

      for each x // infinite enumeration
- free variables in $\varphi(x)$ can be limited as well – by conjunction
  - $R(x) \wedge \varphi(x)$

Safe formulas in DRC

- more generally, a formula is safe if
  1. it does not contain $\forall$ (not a problem, $\forall x\ \varphi(x)$ can be replaced by $\neg\exists x\ (\neg\varphi(x))$)
  2. for each disjunction $\varphi_1 \vee \varphi_2$ it holds that φ_1, φ_2 share the same free variables
     (we consider all implications $\varphi_1 \Rightarrow \varphi_2$ transformed to disjunctions $\neg\varphi_1 \vee \varphi_2$ and the same for equivalences)
  3. all free variables in each maximal conjunction $\varphi_1 \wedge \varphi_2 \wedge \ldots \varphi_n$ **are limited**, i.e., for each free variable **x** at least one of the following conditions holds:
     1. there exists a φ_i with the variable which not a negation or binary (in)equation
        (i.e., φ_i is non-negated complex formula or non-negated „database predicate")
     2. there exists $\varphi_i \equiv x = a$, where a is constant
     3. there exists $\varphi_i \equiv x = v$, where v is limited
  4. the negation is only applicable on conjunctions of step 3

Examples – safe formulas

$\{x,y \mid x = y\}$ not safe (x, y not limited)

$\{x,y \mid x = y \lor R(x,y)\}$ not safe

(the disjunction elements share both free variables, but the first maximal conjunction (x=y) contains equation of not limited variables)

$\{x,y \mid x = y \land R(x,y)\}$ is safe

$\{x,y,z \mid R(x,y) \land \neg(P(x,y) \lor \neg Q(y,z))\}$ not safe (*z* is not limited in the conjunction + the disjunction elements do not share the same variables)

$\{x,y,z \mid R(x,y) \land \neg P(x,y) \land Q(y,z)\}$ equivalent formula to the previous one – now safe

Relational calculus – properties

- "even more declarative" than relational algebra
  (where the structure of nested operations hints the evaluation)
  - just specification of what the result should satisfy
- both DRC and TRC are relational complete
  - moreover, could be extended to be stronger
- besides the different language constructs, the three formalisms can be used for differently "coarse" access to data
  - operations of relational algebra work with entire relations (tables)
  - database predicates of TRC work with relation elements (rows)
  - database predicates of DRC work with attributes (attributes)

Examples – comparison of RA, DRC, TRC

FILM(NAME_FILM, NAME_ACTOR)   ACTOR(NAME_ACTOR, YEAR_BIRTH)

*Which films were casted by **all** the actors?*

RA:

FILM % ACTOR[NAME_ACTOR]

DRC:

$\{(f) \mid FILM(f) \wedge \forall a\ (ACTOR(a) \Rightarrow FILM(f, a))\}$

TRC:

$$\{film[NAME\_FILM] \mid \forall actor(ACTOR(actor) \Rightarrow \exists f(FILM(f) \wedge f.NAME\_ACTOR = actor.NAME\_ACTOR \wedge f.NAME\_FILM = film.NAME\_FILM))\}$$

Examples – comparison of RA, DRC, TRC

EMPLOYEE(firstname, surname, status, children, qualification, practice, health, crimerecord, salary)
– the key is everything except for **salary**

Pairs of employees having similar salary (max. $100 difference)?

DRC:

$\{(e1, e2) \mid \exists p1, s1, pd1, k1, dp1, zs1, tr1, sa1, p2, s2, pd2, k2, dp2, zs2\ tr2, sa2$
$EMPLOYEE(e1, p1, s1, pd1, k1, dp1, zs1, tr1, sa1) \wedge$
$EMPLOYEE(e2, p2, s2, pd2, k2, dp2, zs2\ tr2, sa2) \wedge |sa1 - sa2| \leq 100 \wedge$
$(e1 \neq e2 \vee s1 \neq s2 \vee pd1 \neq pd2 \vee k1 \neq k2 \vee dp1 \neq dp2 \vee zs1 \neq zs2 \vee tr1 \neq tr2)\}$

TRC:

$\{e1[firstname], e2[firstname] \mid EMPLOYEEE(e1) \wedge EMPLOYEEE(e2) \wedge$
$e1 \neq e2 \wedge |e1.salary - e2.salary| \leq 100\}$

Extension of relational calculus

- relational completeness is not enough
  - just data in tables can be retrieved
- we would like to retrieve also derived data,
  - use derived data in queries, respectively
- e.g., queries like
  „Which employees have their salary by 10% higher than an average salary?“
- solution – introducing aggregation functions
  - as shown in SQL SELECT … GROUP BY … HAVING