course:

Database Systems (NDBI025)

SS2011/12

lecture 6:

Query formalisms for relational model – relational algebra

doc. RNDr. Tomáš Skopal, Ph.D.

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

Today's lecture outline

- relational algebra
  - relational operations
  - equivalent expressions
  - relational completeness

Querying in relational model

- the main purpose of a database is to provide access to data (by means of querying)
  - query language needed
  - query language should be strong enough to select any (meaningful) subset of the database
  - symbols defined in schemas stand for the basic constructs in the query language

Database query

- query = delimitation of particular set of data instances
  - a single query may be expressed by multiple expressions of the query language – **equivalent expressions**

- query extent (power of the query language)
  - in **classic models**, only subset of the database is expected as a query result (i.e., values actually present in the database tables)
  - in **extended models**, also derived data can be returned (i.e., computations, statistics, aggregations derived from the data)

Query language formalisms

- as “table data” model is based on the relational model, there can be used well-known formalisms
  - **relational algebra** (today lecture)
    (operations on relations used as query constructs)
  - **relational calculus** (next lecture)
    (database extension of the first-order logic used as a query language)

Relational algebra (RA)

- RA is a set of operations (unary or binary) on relations with schemes; their results are also relations (and schemes)
  - for completeness, to a relation (table contents) **R*** we always consider also a scheme **R(A)** consisting of name and (typed) attributes, i.e., a tuple **<R*, R(A)>**
- a scheme will be named by any unique user-defined identifier
  - for relation resulting from an operation we mostly do not need to define a name for the relation and the scheme – it either enters another operation or is the final result
  - if we need to "store" (or label) the result, e.g., for decomposition of complex query, we use

ResultName := <*expression consisting of relational operations*>

Relational algebra (RA)

- if clear from the context, we use just R_1 ***operation*** R_2 instead of
  $<R_1, R_1(A_1)>$ ***operation*** $<R_2, R_2(A_2)>$

- for binary operation we use infix notation, for unary operations we use postfix notation

- the operation result can be used recursively as an operand of another operation, i.e., a tree of operations can be defined for more complex query

  $(<R1^*, R1(A)>$ ***op1*** $<R1^*, R1(A)>)$ ***op2***

RA – attribute renaming

- attribute renaming – unary operation

$$R^*\langle a_i \rightarrow b_i,\ a_j \rightarrow b_j,\ \ldots\ \rangle =$$
$$\langle R^*,\ R_x((A - \{a_i, a_j, \ldots\}) \cup \{b_i, b_j, \ldots\})\rangle$$

  - only attributes in the scheme are renamed, no data manipulation (i.e., the result is the same relation and the same scheme, just of different attribute names)

RA – set operations

- set operations (binary, infix notation)
  - union – $\langle R_1, R_1(A)\rangle \cup \langle R_2, R_2(A)\rangle = \langle R_1 \cup R_2, R_x(A)\rangle$
  - intersection – $\langle R_1, R_1(A)\rangle \cap \langle R_2, R_2(A)\rangle = \langle R_1 \cap R_2, R_x(A)\rangle$
  - subtraction – $\langle R_1, R_1(A)\rangle - \langle R_2, R_2(A)\rangle = \langle R_1 - R_2, R_x(A)\rangle$
  - cartesian product – $\langle R_1, R_1(A)\rangle \times \langle R_2, R_2(B)\rangle$
    $= \langle R_1 \times R_2, R_x(\{R_1\} \times A \cup \{R_2\} \times B)\rangle$
- union, intersection and subtraction require **compatible schemes** of the operands – it is also the scheme of the result

RA – cartesian product

- a cartesian product gives a new scheme consisting of attributes coming from both source schemes
  - if the attribute names are ambiguous, we use a prefix notation, e.g., **$R_1.a$, $R_2.a$**
- if both the operands are the same, we need first to rename the attributes of one operand, i.e.,

$$\langle R_1, R_1(\{a,b,c\})\rangle \times R_1\langle a \rightarrow d,\ b \rightarrow e,\ c \rightarrow f\rangle$$

Example – set operations

- FILM(FILM_NAME, ACTOR_NAME)
- AMERICAN_FILM = {('Titanic', 'DiCaprio'), ('Titanic', 'Winslet'), ('Top Gun', 'Cruise')}
- NEW_FILM = {('Titanic', 'DiCaprio'), ('Titanic', 'Winslet'), ('Samotáři', 'Macháček')}

  CZECH_FILM = {('Vesničko má, středisková', 'Labuda'), ('Samotáři', 'Macháček')}

  ALL_FILM := **AMERICAN_FILM ∪ CZECH_FILM** =
  {('Titanic', 'DiCaprio'), ('Titanic', 'Winslet'), ('Top Gun', 'Cruise'),
  ('Pelíšky', 'Donutil'), ('Samotáři', 'Macháček')}
- OLD_AMERICAN_AND_CZECH_FILM :=
  (AMERICAN_FILM ∪ CZECH_FILM) – NEW_FILM =
  {('Top Gun', 'Cruise'), ('Vesničko má, středisková', 'Labuda')}

  NEW_CZECH_FILM := **NEW_FILM ∩ CZECH_FILM** = {('Samotáři', 'Macháček')}

RA – projection

- projection (unary operation)

  <R*[C], R(A)> = <{u[C] $\in$ R*}, R(C)>, where C $\subseteq$ A

  - u[C] relation element with values only in C attributes
  - possible duplicities are removed

RA – selection

- selection (unary)

  $\langle R^*(\varphi), R(A)\rangle$ $= \langle\{u \mid u \in R^* \text{ and } \varphi(u)\}, R(A)\rangle$

  - selection of those elements from R* that match a condition $\varphi(u)$
  - condition is a boolean expression (i.e., using **and, or, not**) on atomic formulas $\mathbf{t_1} \,\boldsymbol{\Theta}\, \mathbf{t_2}$ or $\mathbf{t_1} \,\boldsymbol{\Theta}\, \mathbf{a}$, where $\Theta \in \{<, >, =, \geq, \leq, \neq\}$ and $\mathbf{t_i}$ are names of attributes

RA – natural join

- natural join (binary)

  $<R^*, R(A)> * <S^*, S(B)> =$

  $<\{u \mid u[A] \in R^* \text{ and } u[B] \in S^*\}, R_x(A \cup B)>$

  - joining elements of relations A, B using **identity on all shared attributes**
  - if $A \cap B = \varnothing$, natural join is cartesian product
    (no shared attributes, i.e., everything in A is joined with everything in B)
  - could be expressed using cartesian product, selection and projection

Example – selection, projection, natural join

FILM(FILM_NAME, ACTOR_NAME)

ACTOR(ACTOR_NAME, BIRTH_YEAR)

FILM = {('Titanic', 'DiCaprio'), ('Titanic', 'Winslet'), ('Top Gun', 'Cruise')}

ACTOR = {('DiCaprio',1974), ('Winslet',1975), ('Cruise', 1962), ('Jolie', 1975)}

ACTOR_YEAR := **ACTOR[BIRTH_YEAR]** =

{(1974), (1975), (1962)}

YOUNG_ACTOR := **ACTOR(BIRTH_YEAR > 1970) [ACTOR_NAME]** =

{('DiCaprio'), ('Winslet'), ('Jolie')}

FILM_ACTOR := **FILM * ACTOR** =

{('Titanic', 'DiCaprio', 1974), ('Titanic', 'Winslet', 1975), ('Top Gun', 'Cruise', 1962)}

RA – inner Θ-join

- inner Θ-join (binary)

$$\langle R^*, R(A)\rangle[t_1 \Theta t_2]\langle S^*, S(B)\rangle = \langle\{u \mid u[A] \in R^*, u[B] \in S^*, u.t_1 \Theta u.t_2\}, A \cup B\rangle$$

  - generalization of natural join
  - joins over predicate (condition) Θ applied on individual attributes (of schemes entering the operation)

RA – left Θ-semi-join

- left inner Θ-semi-join (binary)

$$\mathbf{<R^*, R(A)><t_1\Theta t_2]<S^*, S(B)>} = (R[t_1\Theta t_2]S)[A]$$

  - join restricted to the “left side”
    (only attributes of A in the result scheme)
  - right semi-join similar (projection on B)

RA – relation division

- relation division (binary)

$$\langle R^*, R(A)\rangle \div \langle S^*, S(B \subset A)\rangle = \langle\{t \mid \forall s \in S^*\ (t \oplus s) \in R^*\}, A - B\rangle$$

- $\oplus$ is concatenation operation
  (relation elements $\langle a_1, a_2, \ldots\rangle$ and $\langle b_1, b_2, \ldots\rangle$ become $\langle a_1, a_2, \ldots, b_1, b_2, \ldots\rangle$)
- returns those elements from **R*** that, when projected on **A–B**, are **duplicates** and, when projected on **B, is equal** to **S***
- alternative definition: $R^* \div S^* = R^*[A-B] - ((R^*[A-B] \times S^*) - R^*)[A-B]$
- used in situations where objects with **all properties** are needed
  - kind of universal quantifier in RA

Example – relation division

FILM(FILM_NAME, ACTOR_NAME) ACTOR(ACTOR_NAME, BIRTH_YEAR)

*What are the films where **all** the actors appeared?*

ACTOR_ALL_FILM := **FILM ÷ ACTOR[ACTOR_NAME])** = {('Titanic')}

FILM_NAME	ACTOR_NAME
Titanic	DiCaprio
Titanic	Winslet
The Beach	DiCaprio
Enigma	Winslet
The Kiss	Zane
Titanic	Zane

ACTOR_NAME	BIRTH_YEAR
DiCaprio	1974
Zane	1966
Winslet	1975

Inner vs. outer join

- so far, we considered **inner joins**
- in practice, it is useful to introduce **null metavalues** (NULL) of attributes
- **outer join** appends series of NULL values to those elements, that were not joined (i.e., they do not appear in inner join)
  - left outer join

    $\mathbf{R} *_L \mathbf{S} = (R * S) \cup (\underline{R} \times (NULL, NULL, \ldots))$
  - right outer join

    $\mathbf{R} *_R \mathbf{S} = (R * S) \cup ((NULL, NULL, \ldots) \times \underline{S})$

    where $\underline{R}$, resp. $\underline{S}$ consist of n-tuples not joined with S, resp. R
  - full outer join

    $\mathbf{R} *_F \mathbf{S} = (R *_L S) \cup (R *_R S)$
  - the above joins are defined as natural joins, outer Θ-joins are defined similarly
- the reason for outer join is a complete information on elements of a relation being joined (some are joined regularly, some only with NULLs)

Example – all types of joins

table **Flight**

Flight	Company	Destination	Passengers
OK251	CSA	New York	276
LH438	Lufthansa	Stuttgart	68
OK012	CSA	Milano	37
AC906	Air Canada	Torronto	116
KL1245	KLM	Amsterdam	130

table **Plane**

Plane	Capacity
Boeing 717	106
Airbus A380	555
Airbus A350	253

Query: In which planes all the passengers can travel (in the respective flights) such that the number of unoccupied seats in plane is lower than 200?

Inner Θ-join – we want the flights and planes that match the given condition

Flight [Flight.Passengers ≤ Plane.Capacity AND Flight.Passengers + 200 > Plane.Capacity] Plane

Left/right/full outer Θ-join – besides the above flight-plane pairs we also want those flights and planes that **do not match** the condition **at all**

Flight	Company	Destination	Passengers	Plane	Capacity
OK251	CSA	New York	276	NULL	NULL
KL1245	KLM	Amsterdam	130	Airbus A350	253
AC906	Air Canada	Torronto	116	Airbus A350	253
LH438	Lufthansa	Stuttgart	68	Airbus A350	253
LH438	Lufthansa	Stuttgart	68	Boeing 717	106
OK012	CSA	Milano	37	Boeing 717	106
NULL	NULL	NULL	NULL	Airbus A380	555

full outer join

inner join

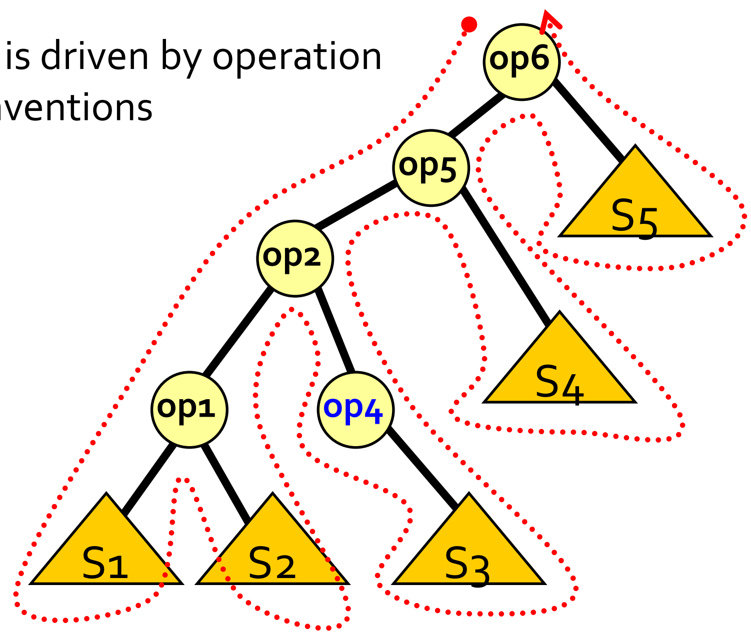
left outer join

right outer join

left/right semi-join (w/o first and last row + after duplicates removal)

RA query evaluation

- logical order of operation evaluation
  - nested operand evaluation needed – depth-first traversal of a syntactic tree
  - e.g., (…((S1 **op1** S2) **op2** (**op4** S3)) **op5** S4 op6 S5)
  - syntactic tree construction (query parsing) is driven by operation priorities, parentheses, or associativity conventions



- operation precedence (priority)
  1. projection R[] (highest)
  2. selection R()
  3. cart. product ×
  4. join, division *, ÷
  5. subtraction –
  6. union, intersection ∪, ∩ (lowest)

Example – query evaluation

Which destination can fly Boeings? (such that all passengers in the flight fit the plane)

(Flight[Passengers, Destination] [Passengers <= Capacity] (Plane(Plane = 'Boeing*')[Capacity]))[Destination]

projection
[Destination]

Destination
Stuttgart
Milano

Θ-join
[Passengers <= Capacity]

Destination	Passengers	Capacity
Stuttgart	68	106
Milano	37	106

Destination	Passengers
New York	276
Stuttgart	68
Milano	37
Torronto	116
Amsterdam	130

projection
[Capacity]

Capacity
106

projection
[Passengers, Destination]

selection
(Plane = 'Boeing*')

Plane	Capacity
Boeing 717	106

Flight	Company	Destination	Passengers
OK251	CSA	New York	276
LH438	Lufthansa	Stuttgart	68
OK012	CSA	Milano	37
AC906	Air Canada	Torronto	116
KL1245	KLM	Amsterdam	130

Plane	Capacity
Boeing 717	106
Airbus A380	555
Airbus A350	253

Equivalent expressions

- a single query may be defined by multiple expressions
  - by replacing “redundant” operations by the basic ones (e.g., division, natural join)
  - by use of commutativity, distributivity and associativity of (some) operations
- selection
  - selection cascade $(\ldots((R(\varphi_1))(\varphi_2))\ldots)(\varphi_n) \equiv R(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n)$
  - commutativity of selection $(R(\varphi_1))(\varphi_2) \equiv (R(\varphi_2))(\varphi_1)$
- projection
  - projection cascade $(\ldots(R[A_1])[A_2])\ldots)[A_n] \equiv R[A_n]$, where $A_n \subseteq A_{n-1} \subseteq \ldots \subseteq A_2 \subseteq A_1$
- join and cartesian product
  - commutativity $R \times S \equiv S \times R$, $R\ [\Theta]\ S \equiv S\ [\Theta]\ R$, etc.
  - associativity $R \times (S \times T) \equiv (R \times S) \times T$, $R\ [\Theta]\ (S\ [\Theta]\ T) \equiv (R\ [\Theta]\ S)\ [\Theta]\ T$, etc.
  - combination, e.g., $R\ [\Theta]\ (S\ [\Theta]\ T) \equiv (R\ [\Theta]\ T)\ [\Theta]\ S$

Equivalent expressions

- complex equivalences for selection, projection and join
  - selection and projection swap
    $(R[A_i])(\varphi) \equiv (R(\varphi))[A_i]$, if $\forall a \in \varphi \Rightarrow a \in A_i$
  - combination of selection and cartesian product (join definition):
    $R\,[\Theta]\,S \equiv (R \times S)(\Theta)$
  - distributive swap of selection and cartesian product (or join)
    $(R \times S)(\varphi) \equiv R(\varphi) \times S$, if $\forall a \in \varphi \Rightarrow a \in A_R \wedge a \notin A_S$
  - distributive swap of projection and cartesian product (or join)
    $(R \times S)[A_1] \equiv R[A_2] \times S[A_3]$,
    if $A_2 \subseteq A_1 \wedge A_2 \subseteq R_A$ and $A_3 \subseteq A_1 \wedge A_3 \subseteq S_A$
    similarly for join, $(R\,[\Theta]\,S)[A_1] \equiv R[A_2]\,[\Theta]\,S[A_3]$,
    where moreover $\forall a \in \Theta \Rightarrow a \in A_1$
- other equivalences can be obtained when including set operations

Example – natural join

$$\langle R, A_R\rangle * \langle S, A_S\rangle \equiv (R \times S)(\forall a \in A_R \cap A_S \Rightarrow R.a = S.a)[(\{R\} \times A_R) \cup (\{S\} \times (A_S - A_R))]$$

projection
$[(\{R\} \times A_R) \cup (\{S\} \times (A_S - A_R))]$

R.Flight	R.Company	R.Destination	R.Capacity	S.Plane
OK251	CSA	New York	276	Boeing 717
KL1245	KLM	Amsterdam	130	Airbus A350

selection
$[(\forall a \in A_R \cap A_S \Rightarrow R.a = S.a)]$

R.Flight	R.Company	R.Destionation	R.Capacity	S.Company	S.Plane	S.Capacity
OK251	CSA	New York	276	CSA	Boeing 717	276
KL1245	KLM	Amsterdam	130	KLM	Airbus A350	130

cartesian product

R.Flight	R.Company	R.Destination	R.Capacity	S.Company	S.Plane	S.Capacity
OK251	CSA	New York	276	CSA	Boeing 717	276
OK251	CSA	New York	276	CSA	Airbus A380	555
OK251	CSA	New York	276	KLM	Airbus A350	130
AC906	Air Canada	Torronto	116	CSA	Boeing 717	276
AC906	Air Canada	Torronto	116	CSA	Airbus A380	555
AC906	Air Canada	Torronto	116	KLM	Airbus A350	130
KL1245	KLM	Amsterdam	130	CSA	Boeing 717	276
KL1245	KLM	Amsterdam	130	CSA	Airbus A380	555
KL1245	KLM	Amsterdam	130	KLM	Airbus A350	130

Flight	Company	Destination	Capacity
OK251	CSA	New York	276
AC906	Air Canada	Torronto	116
KL1245	KLM	Amsterdam	130

Company	Plane	Capacity
CSA	Boeing 717	276
CSA	Airbus A380	555
KLM	Airbus A350	130

Example – relation division

Which companies fly to every destination?

Flight[Company, Destination] ÷ Flight[Destination]

Flight	Company	Destination	Passengers
OK251	CSA	New York	276
LH438	Lufthansa	Stuttgart	68
OK012	CSA	Milano	37
LH123	Lufthansa	Torronto	132
AC906	Air Canada	Torronto	116
LH123	Lufthansa	Milano	69
AC906	Air Canada	Stuttgart	56
LH19	Lufthansa	New York	62
KL24	KLM	Milano	115
LH52	Lufthansa	Amsterdam	164
KL1245	KLM	Amsterdam	130

projection [Company, Destination]

Company	Destination
CSA	New York
Lufthansa	Stuttgart
CSA	Milano
Lufthansa	Torronto
Air Canada	Torronto
Lufthansa	Milano
Air Canada	Stuttgart
Lufthansa	New York
KLM	Milano
Lufthansa	Amsterdam
KLM	Amsterdam

projection [Destination]

Destination
New York
Stuttgart
Milano
Torronto
Amsterdam

division

Company
Lufthansa

Example – “division without division”

Which companies fly to every destination?

$(R^* \div S^* = R^*[A\text{–}B] - ((R^*[A\text{–}B] \times S^*) - R^*)[A\text{–}B])$

Flight[Company, Destination] ÷ Flight[Destination]

Flight[Company] – ((Flight[Company] × Flight[Destination]) – Flight[Company,Destination])[Company]

subtraction

Company
Lufthansa

projection [Company]

Company
CSA
Air Canada
KLM

subtraction

Company	Destination
CSA	Stuttgart
CSA	Torronto
CSA	Amsterdam
Air Canada	New York
Air Canada	Milano
Air Canada	Amsterdam
KLM	New York
KLM	Stuttgart
KLM	Torronto

cart. product

projection [Company, Destination]

Company	Destination
CSA	New York
Lufthansa	Stuttgart
CSA	Milano
Lufthansa	Torronto
Air Canada	Torronto
Lufthansa	Milano
Air Canada	Stuttgart
Lufthansa	New York
KLM	Milano
Lufthansa	Amsterdam
KLM	Amsterdam

projection [Company]

Company
CSA
Lufthansa
Air Canada
KLM

projection [Company]

Company	Destination
CSA	New York
CSA	Stuttgart
CSA	Milano
CSA	Torronto
CSA	Amsterdam
Lufthansa	New York
Lufthansa	Stuttgart
Lufthansa	Milano
Lufthansa	Torronto
Lufthansa	Amsterdam
Air Canada	New York
Air Canada	Stuttgart
Air Canada	Milano
Air Canada	Torronto
Air Canada	Amsterdam
KLM	New York
KLM	Stuttgart
KLM	Milano
KLM	Torronto
KLM	Amsterdam

Flight	Company	Destination	Passengers
OK251	CSA	New York	276
LH438	Lufthansa	Stuttgart	68
OK012	CSA	Milano	37
LH123	Lufthansa	Torronto	132
AC906	Air Canada	Torronto	116
LH123	Lufthansa	Milano	69
AC906	Air Canada	Stuttgart	56
LH19	Lufthansa	New York	62
KL24	KLM	Milano	115
LH52	Lufthansa	Amsterdam	164
KL1245	KLM	Amsterdam	130

projection [Destination]

Destination
New York
Stuttgart
Milano
Torronto
Amsterdam

Relational completeness

- not all the mentioned operations are necessary for expression of every query
  - minimal set consists of the following operations

    B = {union, cartesian product, subtraction, selection, projection, attribute renaming}
- relational algebra query language is the of expressions that result from composition of operations in B over scheme given by database scheme
- if two expressions denote the same query they are equivalent
- query language that is able to express all queries of RA is **relational complete**

RA – properties

- RA = declarative query language
  - i.e., non-procedural, however, the structure of the expression suggests the sequence of operations
- the result is **always finite** relation
  - „safely" defined operations
- operation properties
  - associativity, commutativity
    - cart. product, join