course:

Database Systems (A7B36DBS)

lecture 6:

Query formalisms for relational model – relational algebra

Doc. RNDr. Irena Holubova, Ph.D.

Acknowledgement:
The slides were kindly lent by Doc. RNDr. Tomas Skopal, Ph.D.,
Department of Software Engineering, Charles University in Prague

Today’s lecture outline

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

Map of the Lecture

Database design

Formulation of the task

Conceptual modelling

ER, UML

Transformation to **relational** model

or XML, object, object-relational…

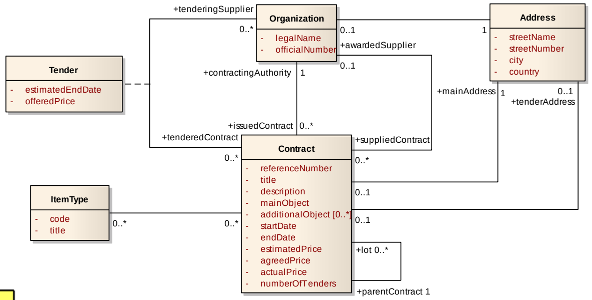
functional dependencies, normal forms, decomposition, synthesis

SQL: Data Definition

relational algebra, relational calculus

SQL: Data Manipulation

Example: Information system for contracts, organizations and tenders



Name	Producer	Price	Availability
Mouse	Dell	5,80	100000
Windows	Microsoft	12,50	NULL
Printer	Toshiba	325,00	15000
Phone	Nokia	135,00	32000
Laptop	Dell	500,00	9000
Bing	Microsoft	0,00	NULL

key

foreign key
(key in table Producer)

Name	Address	Debt
Dell	USA	YES
Microsoft	USA	NO
Samsung	Korea	NO
Nokia	Norway	YES
E.ON	Germany	NO
Toshiba	Japan	NO

key

```
CREATE TABLE Contract (
  id INTEGER,
  startDate DATETIME,
  endDate DATETIME,
  ...);
```

physical storage, implementation of indices

```
SELECT id, name
FROM Contract, Organization
WHERE Organization.id = ... AND
      startDate > ...;
```

Database query

- query = delimiting 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 **classical 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 the “table data” model is based on the relational model, there can be used well-known formalisms
  - **relational algebra** (this 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 (having schemas); their results are also relations (having schemas)
  - for completeness, with a relation (table content) **R*** we always consider also a schema **R(A)** consisting of name and (typed) attributes, i.e., a tuple **<R*, R(A)>**
- a schema will be named by any unique user-defined identifier
  - for the relation resulting from an operation we mostly do not need to define a name for the relation and the schema
    - it either enters another operation or it 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 it is 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 operations 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 a more complex query

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

infix

postfix

RA – attribute renaming

- attribute renaming (unary operation)

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

  - only attributes in the schema are renamed, no data manipulation (i.e., the result is the same relation and the same schema, 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 schemas** of the operands!!!
  - it is also the schema of the result
  - i.e., we cannot, e.g., unify two different schemas

RA – Cartesian product

- a Cartesian product produces a new schema consisting of attributes from both source schemas
  - if the attribute names are ambiguous, we use a prefix notation, e.g., $\mathbf{R_1.a}$, $\mathbf{R_2.a}$

- if both the operands are the same, we first need 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 = {('Pelíšky', 'Donutil'), ('Samotáři', 'Macháček')}

ALL_FILMS := **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'), ('Pelíšky', 'Donutil')}

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] = values only in attributes from C
  - possible duplicities are removed

RA – selection

- selection (unary)

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

  - selection of those elements from R* that match a condition φ
  - condition φ is a Boolean expression
    - using **and, or, not** on atomic formulas $\mathbf{t_1\ \Theta\ t_2}$ or $\mathbf{t_1\ \Theta\ a}$
      - $\Theta \in \{<, >, =, \geq, \leq, \neq\}$
      - $\mathbf{t_i}$ 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 <u>all shared</u> attributes**
  - if $A \cap B = \emptyset$, natural join corresponds to Cartesian product
    - no shared attributes
    - 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)

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

ACTOR(ACTOR_NAME, BIRTH_YEAR)

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

theta

- inner Θ-join (binary)

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

  $<\{u \mid u[A] \in R^*, u[B] \in S^*, u.t_1 \Theta u.t_2\}, A \cup B>$

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

RA – left Θ-semi-join

- left inner Θ-semi-join (binary)

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

  - join restricted to the “left side”
    - only attributes of A in the resulting schema
  - right semi-join similar
    - only attributes of B in the resulting schema

Inner vs. outer join

- in practice, it is useful to introduce **null meta-values** (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

RA – relation division

- relation division (binary)

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

  $<\{t \mid \forall s \in S^* \ (t \oplus s) \in R^*\}, A - B\}$

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

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')}

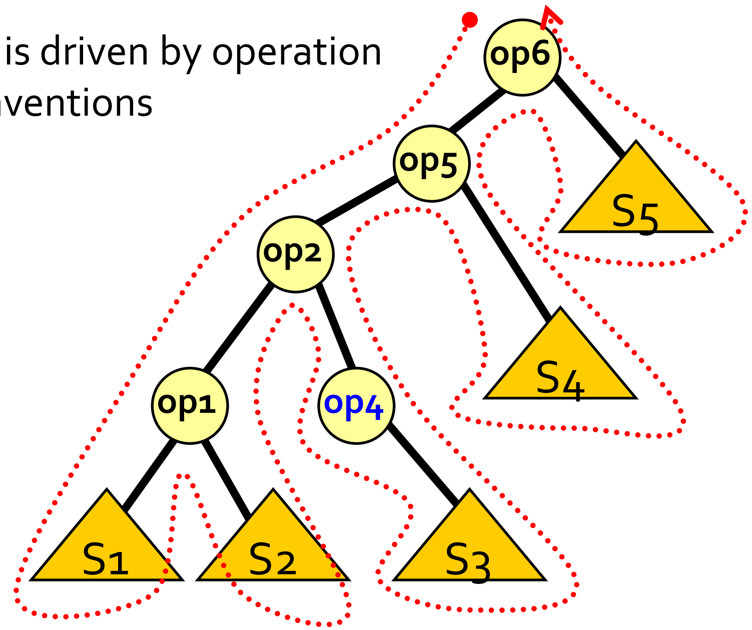
FILM_NAME	ACTOR_NAME
Titanic	DiCaprio
Titanic	Winslet
The Beach	DiCaprio
Enigma	Winslet
The Kiss	Zane
Titanic	Zane

ACTOR_NAME	BIRTH_YEAR
DiCaprio	1974
Zane	1966
Winslet	1975

RA query evaluation

- logical order of operation evaluation
  - depth-first traversal of a syntactic tree
  - e.g., (((**S1 op1 S2**) **op2** (**S3 op4**)) **op5 S4** op6 **S5**)

  - syntactic tree construction (query parsing) is driven by operation priorities, parentheses, or associativity conventions

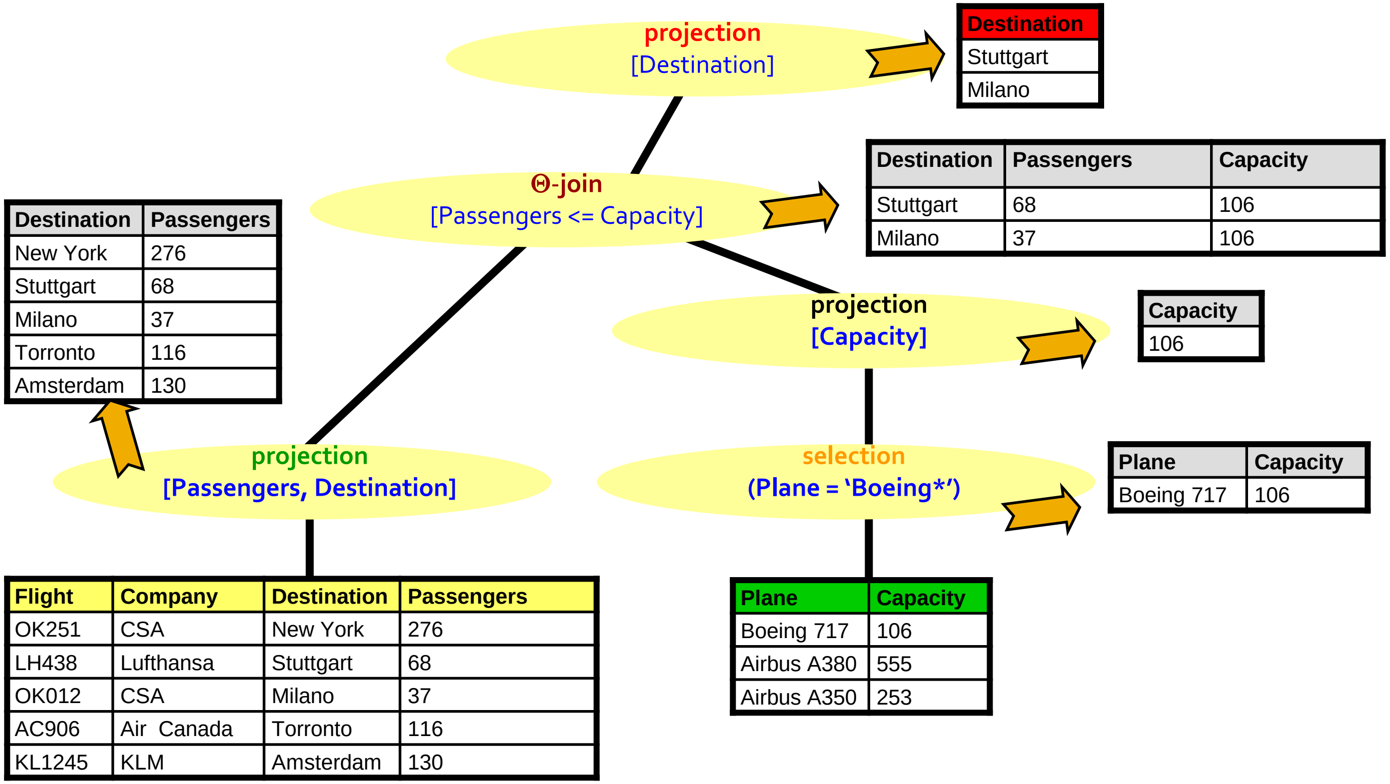


- operation precedence (priority)
  1. projection R[] (highest)
  2. selection R()
  3. Cart. product $\times$
  4. join, division $*$, $\div$
  5. subtraction $-$
  6. union, intersection $\cup$, $\cap$ (lowest)

Example – query evaluation

To 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]



Destination
Stuttgart
Milano

Destination	Passengers	Capacity
Stuttgart	68	106
Milano	37	106

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

Capacity
106

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 with the basic ones (e.g., division, natural join)
  - by use of commutativity, distributivity and associativity of (some) operations
- selection
  - selection cascade $(\ldots((R(\varphi_1))(\varphi_2))\ldots)(\varphi_n) \equiv R(\varphi_1 \land \varphi_2 \land \ldots \land \varphi_n)$
  - 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.

Relational completeness

- not all the mentioned operations are necessary for expression of every query
  - the minimal set consists of the following operations
    B = {union, Cartesian product, subtraction, selection, projection, attribute renaming}
- relational algebra query language is a set of expressions that result from composition of operations in B over a database schema
- if two expressions denote the same query they are **equivalent**
- query language that is able to express all queries of RA is **relational complete**

- **Questions:**
  - **How can we prove that a particular language is relational complete?**
  - **Is SQL 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