course:

Database Systems (NDBI025)

SS2011/12

lecture 9:

Relational design – algorithms

doc. RNDr. Tomáš Skopal, Ph.D.

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

Today's lecture outline

- schema analysis
  - basic algorithms (attribute closure, FD membership and redundancy)
  - determining the keys
  - testing normal forms
- normalization of universal schema
  - decomposition (to BCNF)
  - synthesis (to 3NF)

Attribute closure

- closure X^+ of attribute set X according to FD set F
  - principle: we interatively derive all attributes „F-determined" by attributes in X
  - complexity O(m*n), where **n** is the number of attributes and **m** is number of FDs

```
algorithm AttributeClosure(set of dependencies F, set of attributes X) : returns set X⁺
    ClosureX := X; DONE := false; m = |F|;
    while not DONE do
        DONE := true;
        for i := 1 to m do
            if (LS[i] ⊆ ClosureX and RS[i] ⊄ ClosureX) then
                ClosureX := ClosureX ∪ RS[i];
                DONE := false;
            endif
        endfor
    endwhile
    return ClosureX;
```

Note: expression LS[i] (RS[i], respectively) represents left (right, resp.) side of i-th FD in F
The trivial FD is used (algorithm initialization) and then transitivity (test of left side in the closure).
The composition and decomposition usage is hidden in the inclusion test.

Example – attribute closure

$F = \{a \to b, bc \to d, bd \to a\}$

$\{b,c\}^+ = ?$

1. ClosureX := $\{b,c\}$ (initialization)
2. ClosureX := ClosureX $\cup \{d\} = \{b,c,d\}$ $(bc \to d)$
3. ClosureX := ClosureX $\cup \{a\} = \{a,b,c,d\}$ $(bd \to a)$

$\{b,c\}^+ = \{a,b,c,d\}$

Membership test

- we often need to check if a FD $X \rightarrow Y$ belongs to F^+, i.e., to solve the problem $\{X \rightarrow Y\} \in F^+$
- materializing F^+ is not practical, we can employ the attribute closure

algorithm ***IsDependencyInClosure***(set of dependencies F, FD $X \rightarrow Y$)
return $Y \subseteq$ *AttributeClosure*(F, X);

Redundancy testing

The membership test can be easily used when testing redundancy of

- FD $X \rightarrow Y$ in F.
- attribute in X (according to F and $X \rightarrow Y$).

algorithm ***IsDependencyRedundant***(set of dependencies F, dependency $X \rightarrow Y \in F$)
    return *IsDependencyInClosure*($F - \{X \rightarrow Y\}, X \rightarrow Y$);

algorithm ***IsAttributeRedundant***(set of deps. F, dep. $X \rightarrow Y \in F$, attribute $a \in X$)
    return *IsDependencyInClosure*(F, $X - \{a\} \rightarrow Y$);

In the ongoing slides we find useful the algorithm for reduction of the left side of a FD:

algorithm ***GetReducedAttributes***(set of deps. F, dep. $X \rightarrow Y \in F$)
    $X' := X$;
    for each $a \in X$ **do**
        if *IsAttributeRedundant*(F, $X' \rightarrow Y$, a) **then** $X' := X' - \{a\}$;
    endfor
return X';

Minimal cover

- for all FDs we test redundancies and remove them

algorithm ***GetMinimumCover***(set of dependencies F): returns minimal cover G
decompose each dependency in F into elementary ones
for each $X \rightarrow Y$ **in** F **do**
F := (F – {$X \rightarrow Y$}) $\cup$ {*GetReducedAttributes*(F, $X \rightarrow Y$) $\rightarrow$ Y};
endfor
for each $X \rightarrow Y$ **in** F **do**
if *IsDependencyRedundant*(F, $X \rightarrow Y$) **then** F := F – {$X \rightarrow Y$};
endfor
return F;

Determining (first) key

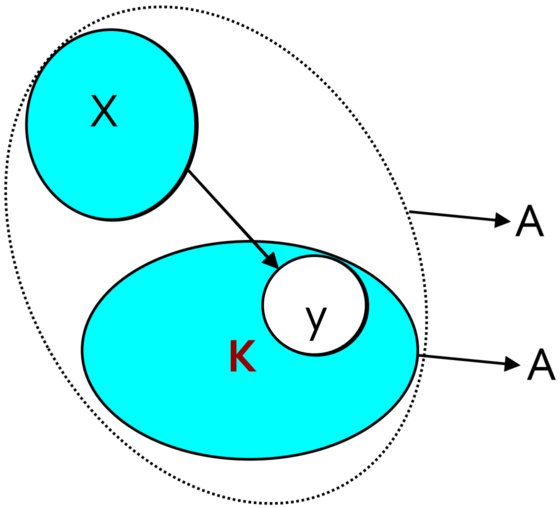
- the algorithm for attribute redundancy testing could be used directly for determining a key
- redundant attributes are iteratively removed from left side of $A \rightarrow A$

algorithm ***GetFirstKey***(set of deps. F, set of attributes A) : **returns a key** K;
    return *GetReducedAttributes*(F, $A \rightarrow A$);

Note: Because multiple keys can exists, the algorithm finds only one of them.
    Which? It depends on the traversing of the attribute set within the algorithm GetReducedAttributes.

Determining all keys, the principle

Let's have a schema S(A, F).
Simplify F to minimal cover.



1. Find any key **K** (see previous slide).
2. Take a FD $X \rightarrow y$ in F such that $y \in K$ or terminate if not exists (there is no other key).
3. Because $X \rightarrow y$ and $K \rightarrow$ **A**, it transitively holds also $X\{K - y\} \rightarrow$ **A**, i.e., $X\{K - y\}$ is super-key.
4. Reduce FD $X\{K - y\} \rightarrow$ **A** so we obtain key **K'** on the left side.
   This key is surely different from **K** (we removed y).
5. If **K'** is not among the determined keys so far, we add it,
   declare **K**=**K'** and repeat from step 2. Otherwise we finish.

Determining all keys, the algorithm

- Lucchesi-Osborn algorithm
  - to an already determined key we search for equivalent sets of attributes, i.e., other keys
- NP-complete problem (theoretically exponential number of keys/FDs)

algorithm ***GetAllKeys***(set of deps. F, set of attributes A) : **returns set of all keys** Keys;
    let all dependencies in F be non-trivial, i.e. replace every $X \rightarrow Y$ by $X \rightarrow (Y - X)$
    K := *GetFirstKey*(F, A);
    Keys := {K};
    Done := **false**;
    while Done = **false do**
        Done := **true**;
        for each $X \rightarrow Y \in F$ **do**
            if $(Y \cap K \neq \emptyset$ **and** $\neg\exists K' \in Keys : K' \subseteq (K \cup X) - Y)$ **then**
                K := *GetReducedAttributes*(F, $((K \cup X) - Y) \rightarrow A$);
                Keys := Keys $\cup$ {K};
                Done := **false**;
        endfor
    endwhile
return Keys;

Example – determining all keys

Contracts(A, F)

A = {c = ContractId, s = SupplierId, j = ProjectId, d = DeptId,
p = PartId, q = Quantity, v = Value}

F = {**c → *all*, sd → p, p → d, jp → c, j → s**}

1. Determine first key – Keys = {**c**}
2. Iteration 1: take **jp → c** that has a part of the last key on right side (in this case the whole key – **c**) and **jp** is not a super-set of already determined key
3. **jp → *all*** is reduced (no redundant attribute), i.e.,
4. Keys = {**c, jp**}
5. Iteration 2: take **sd → p** that has a part of the last key on right side (**jp**), {**jsd**} is not super-set of **c** nor **jp**, i.e., it is a key candidate
6. in **jsd → *all*** we get redundan attribute **s**, i.e.,
7. Keys = {**c, jp, jd**}
8. Iteration 3: take **p → d**, however, **jp** was already found so we do not add it
9. finishing as the iteration 3 resulted in no key addition

Testing normal forms

- NP-complete problem
  - we must know all keys – then it is sufficient to test a FD in F, so we do not need to materialize F^+
  - or, just one key needed, but also needing extension of F to F^+
- fortunately, in practice the keys determination is fast
  - thanks to limited size of F and „separability“ of FDs

Design of database schemas

Two means of modeling relational database:

- we get a set of relational schemas
  (as either direct relational design or conversion from conceptual model)
  - normalization performed separately on each table
  - the database could get unnecessarily highly "granularized" (too many tables)
- considering the whole database as a bag of (global) attributes results in a single *universal database schema* – i.e., one big table – including single set of FDs
  - normalization performed on the universal schema
  - less tables (better „granulating")
  - „classes/entities" are generated (recognized) as the consequence of FD set
  - modeling at the attribute level is less intuitive than the conceptual modeling (historical reasons)
- both approaches could be combined – i.e., at first, create a conceptual database model, then convert it to relational schemas and finally merge some | (all in the extreme case)

Relational schema normalization

- just one way – decomposition to multiple schemas
  - or merging some„abnormal" schemas and then decomposition
- different criteria
  - **data integrity preservation**
    - **lossless join**
    - **dependency preserving**
  - requirement on normal form (3NF or BCNF)
- manually or algorithmically

Why to preserve integrity?

If the decomposition is not limited, we can decompose the table to several single-column ones that surely are all in BCNF.

Company	HQ	Altitude
Sun	Santa Clara	25 m
Oracle	Redwood	20 m
Microsoft	Redmond	10 m
IBM	New York	15 m

Company,
HQ $\rightarrow$ Altitude

Company
Sun
Oracle
Microsoft
IBM

Company

HQ
Santa Clara
Redwood
Redmond
New York

HQ

Altitude
25 m
20 m
10 m
15 m

Altitude

Clearly, there is something wrong with such a decomposition…

…it is **lossy** and it does not **preserve dependencies**

Lossless join

- a property of decomposition that ensures correct joining (reconstruction) of the universal relation from the decomposed ones

- <u>Definition 1</u>:
  Let $R(\{X \cup Y \cup Z\}, F)$ be universal schema, where $Y \rightarrow Z \in F$.
  Then decomposition $R_1(\{Y \cup Z\}, F_1)$, $R_2(\{Y \cup X\}, F_2)$ is lossless.

- Alternative <u>Definition 2</u>:
  Decomposition of $R(A, F)$ into $R_1(A_1, F_1)$, $R_2(A_2, F_2)$ is lossless, if $A_1 \cap A_2 \rightarrow A_1$ or $A_2 \cap A_1 \rightarrow A_2$

- **Alternative <u>Definition 3</u>:**
  Decomposition of $R(A, F)$ into $R_1(A_1, F_1)$, ..., $R_n(A_n, F_n)$ is lossless, if $R' = *_{i=1..n} R'[A_i]$.

  <u>Note:</u>
  R' is an instance of schema R (i.e., actual relation/table – the data).
  Operation * is natural join and $R'[A_i]$ is projection of R' on an attribute subset $A_i \subseteq A$.

Example – lossy decomposition

Company	Uses DBMS	Data managed
Sun	Oracle	50 TB
Sun	DB2	10 GB
Microsoft	MSSQL	30 TB
Microsoft	Oracle	30 TB

Company, Uses DBMS

Company	Uses DBMS
Sun	Oracle
Sun	DB2
Microsoft	MSSQL
Microsoft	Oracle

Company, Uses DBMS

Company	Data managed
Sun	50 TB
Sun	10 GB
Microsoft	30 TB

Company, Data managed

„reconstruction"
(natural join)

Company	Uses DBMS	Data managed
Sun	Oracle	50 TB
Sun	Oracle	10 GB
Sun	DB2	10 GB
Sun	DB2	50 TB
Microsoft	MSSQL	30 TB
Microsoft	Oracle	30 TB

Company, Uses DBMS, Data managed

Example – lossless decomposition

Company	HQ	Altitude
Sun	Santa Clara	25 m
Oracle	Redwood	20 m
Microsoft	Redmond	10 m
IBM	New York	15 m

<ins>Company</ins>,
HQ → Altitude

Company	HQ
Sun	Santa Clara
Oracle	Redwood
Microsoft	Redmond
IBM	New York

<ins>Company</ins>

HQ	Altitude
Santa Clara	25 m
Redwood	20 m
Redmond	10 m
New York	15 m

<ins>HQ</ins>

„reconstruction“
(natural join)

Dependency preserving

- a decomposition property that ensures no FD will be lost
- <u>Definition:</u>
  Let $R_1(A_1, F_1)$, $R_2(A_2, F_2)$ is decomposition of $R(A, F)$, then such decomposition preserves dependencies if $F^+ = (\cup_{i=1..n} F_i)^+$.
- Dependency preserving could be violated in two ways
  - during decomposition of F we do not derive all valid FDs – we lose FD that should be preserved in a particular schema
  - even if we derive all valid FDs (i.e., we perform projection of F^+), we may lose a FD that is valid **across the schemas**

Example – dependency preserving

dependencies not preserved, we lost
HQ $\rightarrow$ Altitude

Company	HQ	Altitude
Sun	Santa Clara	25 m
Oracle	Redwood	20 m
Microsoft	Redmond	10 m
IBM	New York	15 m

Company,
HQ $\rightarrow$ Altitude

Company	Altitude
Sun	25 m
Oracle	20 m
Microsoft	10 m
IBM	15 m

Company

Company	HQ
Sun	Santa Clara
Oracle	Redwood
Microsoft	Redmond
IBM	New York

HQ

dependencies
preserved

Company	HQ
Sun	Santa Clara
Oracle	Redwood
Microsoft	Redmond
IBM	New York

Firma

HQ	Altitude
Santa Clara	25 m
Redwood	20 m
Redmond	10 m
New York	15 m

Sídlo

The "Decomposition" algorithm

- algorithm for decomposition into BCNF, preserving lossless join
- does not preserve dependencies
  - not an algorithm property – sometimes we simply cannot decompose into BCNF with all FDs preserved

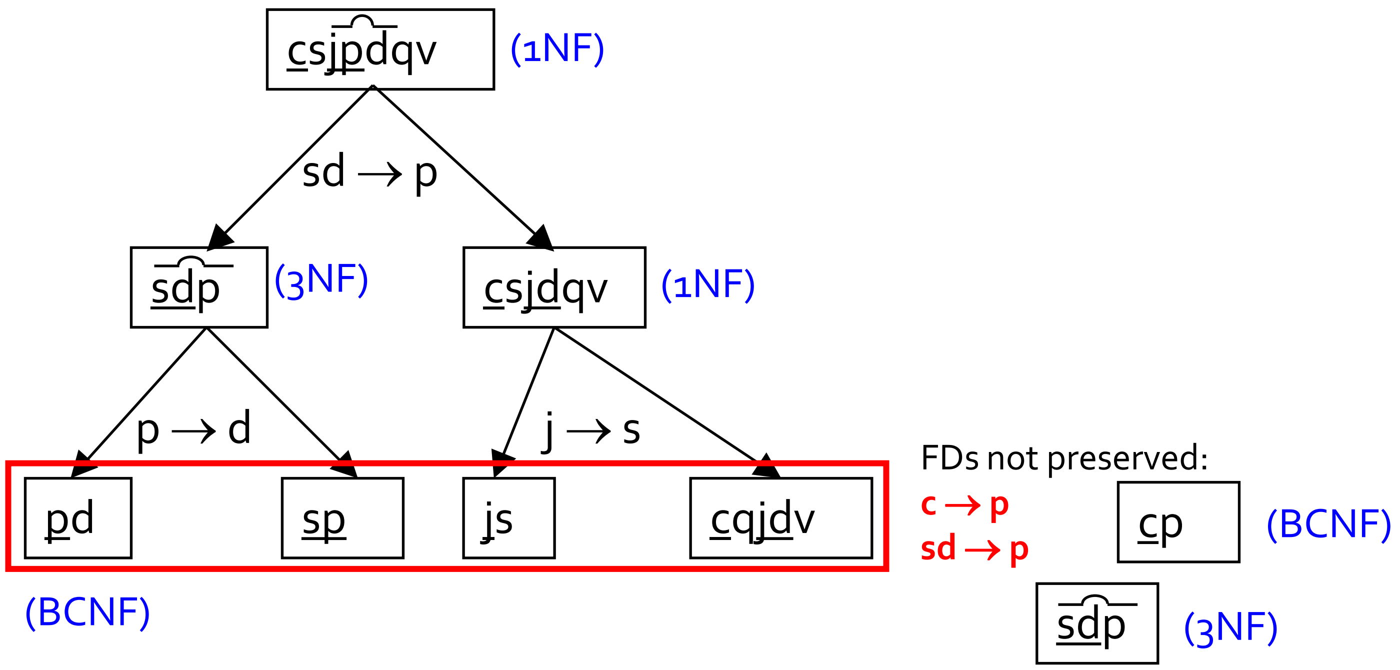
algorithm ***Decomposition***(set of elem. deps. F, set of attributes A) : **returns set** $\{R_i(A_i, F_i)\}$
  Result := $\{R(A, F)\}$;
  Done := **false**;
  Create F^+;
  while not Done **do**
    if $\exists R_i(F_i, A_i) \in$ Result not being in BCNF **then** // *if there is a schema in the result violating BCNF*
      Let $X \rightarrow Y \in F_i$ such that $X \rightarrow A_i \notin F^+$. // *X is not (super)key and so $X \rightarrow Y$ violates BCNF*
      Result := (Result – $\{R_i(A_i, F_i)\}$) $\cup$ // *we remove the schema being decomposed*
        $\{R_i(A_i - Y$, cover$(F, A_i - Y))\}$ $\cup$ // *we add the schema being decomposed without attributes Y*
        $\{R_j(X \cup Y$, cover$(F, X \cup Y))\}$ // *we add the schema with attributes XY*
    else
      Done := **true**;
    endwhile
  return Result;

This partial decomposition on two tables is lossless, we get two schemas that both contain X, while the second one contains also Y and it holds $X \rightarrow Y$.
X is now in the second table a super-key and $X \rightarrow Y$ is no more violating BCNF (in the first table there is not Y anymore).

Note: Function cover(X, F) returns all FDs valid on attributes from X, i.e., a subset of F^+ that contains only attributes from X. Therefore it is necessary to compute F^+.

Example – decomposition

Contracts(A, F)
A = {c = ContractId, s = SupplierId, j = ProjectId, d = DeptId, p = PartId, q = Quantity, v = Value}
F = {c $\rightarrow$ all, sd $\rightarrow$ p, p $\rightarrow$ d, jp $\rightarrow$ c, j $\rightarrow$ s}



The "Synthesis" algorithm

- algorithm for decomposition into 3NF, preserving dependencies
  - basic version not preserving lossless joins

algorithm ***Synthesis***(set of elem. deps. F, set of attributes A) : **returns set** $\{R_i(F_i, A_i)\}$
    create minimal cover from F into G
    compose FDs having equal left side into a single FD
    every composed FD forms a scheme $R_i\ (A_i, F_i)$ of decomposition
return $\cup_{i=1..n}\{R_i\ (A_i, F_i)\}$

- lossless joins can be preserved by adding another schema into the decomposition that contains *universal key* (i.e., a key from the original universal schema)
- a schema in decomposition that is a subset of another one can be deleted
- we can try to merge schemas that have functionally equivalent keys, but such an operation can violate 3NF! (or BCNF if achieved)

Example – synthesis

Contracts(A, F)

A = {c = ContractId, s = SupplierId, j = ProjectId, d = DeptId, p = PartId,
q = Quantity, v = Value}

F = {c $\rightarrow$ sjdpqv, sd $\rightarrow$ p, p $\rightarrow$ d, jp $\rightarrow$ c, j $\rightarrow$ s}

Minimal cover:

- There are no redundant attributes in FDs. There were removed redundant FDs c $\rightarrow$ s and c $\rightarrow$ p.
- G = {c $\rightarrow$ j, c $\rightarrow$ d, c $\rightarrow$ q, c $\rightarrow$ v, sd $\rightarrow$ p, p $\rightarrow$ d, jp $\rightarrow$ c, j $\rightarrow$ s}

Composition:

- G' = {c $\rightarrow$ jdqv, sd $\rightarrow$ p, p $\rightarrow$ d, jp $\rightarrow$ c, j $\rightarrow$ s}

Result:

- R_1({cqjdv}, {c $\rightarrow$ jdqv}), R_2({sdp}, {sd $\rightarrow$ p}), ~~R_3({pd},{p $\rightarrow$ d})~~, R_4({jpc}, {jp $\rightarrow$ c}), R_5({js}, {j $\rightarrow$ s}))

(subset in R_2)

Equivalent keys: {c, jp, jd}

R_1({cqjpdv}, {c $\rightarrow$ jdqv, jp $\rightarrow$ c}), R_2({sdp}, {sd $\rightarrow$ p, p $\rightarrow$ d}), R_5({js}, {j $\rightarrow$ s}))

merging R_1 and R_4
(however, now p $\rightarrow$ d violates BCNF)

Bernstein's extension

- if merging the schemas using equivalent keys K_1, K_2 violated 3NF, we perform the decomposition again
  1. $F_{new} = F \cup \{K_1 \rightarrow K_2, K_2 \rightarrow K_1\}$
  2. we determine redundant FDs in F_{new}, but remove them from F
  3. the final tables are made from reduced F and $\{K_1 \cup K_2\}$

Demo

- program Database algorithms
  - download from my web page
- example 1
- example 2