lecture 9:
Relational design – algorithms

course:
Database Systems (NDBI025)
SS2011/12

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 $\text{AttributeClosure}(\text{set of dependencies } F, \text{set of attributes } X) : \text{returns set } X^+$

$\text{ClosureX} := X; \text{DONE} := \text{false}; m = |F|;$

$\text{while not DONE do}$

$\text{DONE} := \text{true};$

$\text{for } i := 1 \text{ to } m \text{ do}$

$\text{if } (\text{LS}[i] \subseteq \text{ClosureX} \text{ and } \text{RS}[i] \not\subseteq \text{ClosureX}) \text{ then}$

$\text{ClosureX} := \text{ClosureX} \cup \text{RS}[i];$

$\text{DONE} := \text{false};$

$\text{endif}$

$\text{endfor}$

$\text{endwhile}$

$\text{return } \text{ClosureX};$

Note: expression $\text{LS}[i]$ ($\text{RS}[i]$, 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

\[ F = \{ a \rightarrow b, \ bc \rightarrow d, \ bd \rightarrow a \} \]

\( \{b,c\}^+ = ? \)

1. \( \text{Closure}_X := \{b,c\} \) \quad \text{(initialization)}

2. \( \text{Closure}_X := \text{Closure}_X \cup \{d\} = \{b,c,d\} \) \quad (bc \rightarrow d)

3. \( \text{Closure}_X := \text{Closure}_X \cup \{a\} = \{a,b,c,d\} \) \quad (bd \rightarrow 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 $Y \subseteq \text{AttributeClosure}(F, X)$;
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 \textit{GetMinimumCover}(set of dependencies F): returns minimal cover G

decompose each dependency in F into elementary ones

\begin{verbatim}
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;
\end{verbatim}
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 \( K \);

```plaintext
return GetReducedAttributes(F, A → A);
```

**Note:** Because multiple keys can exist, 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.

Relational design – algorithms (NDBIO25, Lect. 9)
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 \texttt{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 $X \rightarrow (Y - X)$

$K := \text{GetFirstKey}(F, A)$;
Keys := \{K\};
Done := false;

\textbf{while} Done = false \textbf{do}
  Done := true;
  \textbf{for each} $X \rightarrow Y \in F$ \textbf{do}
    \textbf{if} $(Y \cap K \neq \emptyset \text{ and } \not\exists ! K' \in \text{Keys} : K' \subseteq (K \cup X) - Y)$ \textbf{then}
      $K := \text{GetReducedAttributes}(F, ((K \cup X) - Y) \rightarrow A)$;
      Keys := Keys \cup \{K\};
      Done := false;
  \textbf{endfor}
\textbf{endwhile}

return Keys;

Relational design – algorithms (NDBI025, Lect. 9)
Example – determining all keys

Contracts\((A, F)\)

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

\[ F = \{ c \rightarrow \text{all}, sd \rightarrow p, p \rightarrow d, jp \rightarrow c, j \rightarrow s \} \]

1. Determine first key – Keys = \{c\}
2. **Iteration 1**: take \(jp \rightarrow c\) 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. \(jp \rightarrow \text{all}\) is reduced (no redundant attribute), i.e.,
4. Keys = \{c, jp\}
5. **Iteration 2**: take \(sd \rightarrow p\) 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 \(jsd \rightarrow \text{all}\) we get redundant attribute \(s\), i.e.,
7. Keys = \{c, jp, jd\}
8. **Iteration 3**: take \(p \rightarrow d\), however, \(jp\) was already found so we do not add it
9. finishing as the iteration 3 resulted in no key addition

Relational design – algorithms (NDBI025, Lect. 9)
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.

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

**Definition 1:**
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 Definition 2:**
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 Definition 3:**
Decomposition of $R(A, F)$ into $R_1(A_1, F_1), ..., R_n(A_n, F_n)$ is lossless, if $R' = \bigstar_{i=1..n} R'[A_i]$.

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

Relational design – algorithms (NDBI025, Lect. 9)
Example – lossy decomposition

<table>
<thead>
<tr>
<th>Company</th>
<th>Uses DBMS</th>
<th>Data managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Oracle</td>
<td>50 TB</td>
</tr>
<tr>
<td>Sun</td>
<td>DB2</td>
<td>10 GB</td>
</tr>
<tr>
<td>Microsoft</td>
<td>MSSQL</td>
<td>30 TB</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Oracle</td>
<td>30 TB</td>
</tr>
</tbody>
</table>

Company, Uses DBMS

<table>
<thead>
<tr>
<th>Company</th>
<th>Uses DBMS</th>
<th>Data managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Oracle</td>
<td>50 TB</td>
</tr>
<tr>
<td>Sun</td>
<td>DB2</td>
<td>10 GB</td>
</tr>
<tr>
<td>Microsoft</td>
<td>MSSQL</td>
<td>30 TB</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Oracle</td>
<td>30 TB</td>
</tr>
</tbody>
</table>

Company, Data managed

<table>
<thead>
<tr>
<th>Company</th>
<th>Data managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>50 TB</td>
</tr>
<tr>
<td>Sun</td>
<td>10 GB</td>
</tr>
<tr>
<td>Microsoft</td>
<td>30 TB</td>
</tr>
</tbody>
</table>

Company, Uses DBMS

„reconstruction“ (natural join)

<table>
<thead>
<tr>
<th>Company</th>
<th>Uses DBMS, Data managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Oracle, 50 TB</td>
</tr>
<tr>
<td>Sun</td>
<td>Oracle, 10 GB</td>
</tr>
<tr>
<td>Sun</td>
<td>DB2, 10 GB</td>
</tr>
<tr>
<td>Sun</td>
<td>DB2, 50 TB</td>
</tr>
<tr>
<td>Microsoft</td>
<td>MSSQL, 30 TB</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Oracle, 30 TB</td>
</tr>
</tbody>
</table>

Company, Uses DBMS, Data managed

Relational design – algorithms (NDBI025, Lect. 9)
Example – lossless decomposition

<table>
<thead>
<tr>
<th>Company</th>
<th>HQ</th>
<th>Altitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Santa Clara</td>
<td>25 m</td>
</tr>
<tr>
<td>Oracle</td>
<td>Redwood</td>
<td>20 m</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Redmond</td>
<td>10 m</td>
</tr>
<tr>
<td>IBM</td>
<td>New York</td>
<td>15 m</td>
</tr>
</tbody>
</table>

Company, HQ → Altitude

---

<table>
<thead>
<tr>
<th>Company</th>
<th>HQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Santa Clara</td>
</tr>
<tr>
<td>Oracle</td>
<td>Redwood</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Redmond</td>
</tr>
<tr>
<td>IBM</td>
<td>New York</td>
</tr>
</tbody>
</table>

„reconstruction“ (natural join)

<table>
<thead>
<tr>
<th>HQ</th>
<th>Altitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Santa Clara</td>
<td>25 m</td>
</tr>
<tr>
<td>Redwood</td>
<td>20 m</td>
</tr>
<tr>
<td>Redmond</td>
<td>10 m</td>
</tr>
<tr>
<td>New York</td>
<td>15 m</td>
</tr>
</tbody>
</table>

Relational design – algorithms (NDBI025, Lect. 9)
Dependency preserving

• a decomposition property that ensures no FD will be lost

• **Definition:**
  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 $F^+ = (\bigcup_{i=1..n} F_i)^+$.

• 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 → Altitude
- Dependencies preserved

<table>
<thead>
<tr>
<th>Company</th>
<th>HQ</th>
<th>Altitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Santa Clara</td>
<td>25 m</td>
</tr>
<tr>
<td>Oracle</td>
<td>Redwood</td>
<td>20 m</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Redmond</td>
<td>10 m</td>
</tr>
<tr>
<td>IBM</td>
<td>New York</td>
<td>15 m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company</th>
<th>HQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Santa Clara</td>
</tr>
<tr>
<td>Oracle</td>
<td>Redwood</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Redmond</td>
</tr>
<tr>
<td>IBM</td>
<td>New York</td>
</tr>
</tbody>
</table>

Relational design – algorithms (NDBI025, Lect. 9)
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 \( X \rightarrow A_i \not\in F^+ \). // X is not (super)key and so \( X \rightarrow Y \) violates BCNF
    Result := \( (\text{Result} - \{R_i(A_i, F_i)\}) \cup \{R_i(A_i - Y, \text{cover}(F, A_i - Y))\}) \cup \{R_j(X \cup Y, \text{cover}(F, X \cup Y))\} \) // we remove the schema being decomposed
      // we add the schema being decomposed without attributes
  else
    Done := true;
endwhile
return Result;

**Note**: Function \( \text{cover}(X, F) \) returns all FDs valid on attributes from \( X \) in a superset of \( F^+ \) that contains only

- Relational design – algorithms (NDBI025, Lect. 9)
Example – decomposition

Contracts \((A, F)\)

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

\[ F = \{c \rightarrow \text{all}, sd \rightarrow p, p \rightarrow d, jp \rightarrow c, j \rightarrow s\} \]

Relational design – algorithms (NDBI025, Lect. 9)
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 \( \bigcup_{i=1..n} \{ R_i(A_i, F_i) \} \)

- 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\((A, F)\)

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

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

**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:**
- \(R_1(\{cqjdv\}, \{c \rightarrow jdqv\}), R_2(\{sdp\}, \{sd \rightarrow p\}), R_3(\{pd\}, \{p \rightarrow d\}), R_4(\{jpc\}, \{jp \rightarrow c\}), \)

\(R_5(\{js\}, \{j \rightarrow s\})\)

\(\text{Equivalent keys: } \{c, jp, jd\}\)

Relational design – algorithms (NDBI025, Lect. 9)
Bernstein’s extension

• if merging the schemas using equivalent keys $K_1, K_2$ violated 3NF, we perform the decomposition again

1. $F_{\text{new}} = F \cup \{K_1 \rightarrow K_2, K_2 \rightarrow K_1\}$
2. we determine redundant FDs in $F_{\text{new}}$ 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