course:
Database Systems (A7B36DBS)

lecture 9:
Relational design – algorithms

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

- 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 iteratively derive all attributes „F-determined“ by attributes in $X$
  - complexity $O(m \times n)$, where $n$ is the number of attributes and $m$ is number of FDs

```
algorism 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[$i$] $\subseteq$ ClosureX and RS[$i$] $\notin$ ClosureX) then
        ClosureX := ClosureX $\cup$ RS[$i$];
        DONE := false;
      endif
    endfor
  endwhile
  return ClosureX;
```

The trivial FD is used (algorithm initialization) and then transitivity (test of left-hand 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. Closure_X := \{b,c\} \quad \text{(initialization)}
2. Closure_X := \text{Closure}_X \cup \{d\} = \{b,c,d\} \quad (bc \rightarrow d)
3. Closure_X := \text{Closure}_X \cup \{a\} = \{a,b,c,d\} \quad (bd \rightarrow a)

\{b,c\}^+ = \{a,b,c,d\}
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 \texttt{IsDependencyInClosure}(set of FDs $F$, FD $X \rightarrow Y$)
\hspace{1cm} \textbf{return} $Y \subseteq \text{AttributeClosure}(F, X)$;
The membership test can be easily used when testing redundancy of
- FD $X \rightarrow Y$ in $F$
- attribute $a$ in $X$ (according to $F$ and $X \rightarrow Y$)

algorithm $\text{IsDependencyRedundant}(\text{set of FDs } F, \text{ FD } X \rightarrow Y \in F)$
  return $\text{IsDependencyInClosure}(F - \{X \rightarrow Y\}, X \rightarrow Y)$;

algorithm $\text{IsAttributeRedundant}(\text{set of FDs } F, \text{ FD } X \rightarrow Y \in F, \text{ attr. } a \in X)$
  return $\text{IsDependencyInClosure}(F, X - \{a\} \rightarrow Y)$;

In the following slides we find useful the algorithm for reduction of the left-hand side of a FD:

algorithm $\text{ReduceAttributes}(\text{set of FDs } F, \text{ FD } X \rightarrow Y \in F)$
  $X' := X$;
  for each $a \in X$ do
    if $\text{IsAttributeRedundant}(F, X' \rightarrow Y, a)$ 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 FD in F into elementary FDs
  for each \( X \rightarrow Y \) in F do
    F := (F – \{X \rightarrow Y\}) \cup \{ReduceAttributes(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\};
    endif
  endfor
  return F;

Relational design – algorithms (A7B36DBS, Lect. 9)
the algorithm for attribute redundancy testing could be used directly for determining a key

redundant attributes are iteratively removed from left-hand side of trivial FD $A \rightarrow A$

algorithm \texttt{GetFirstKey}(set of deps. $F$, set of attributes $A$):

: \texttt{returns} a key $K$;

\texttt{return} \texttt{ReduceAttributes}($F$, $A \rightarrow A$);

\textbf{Note:} Because multiple keys can exists, the algorithm finds only one of them.

Which one? It depends on the traversing of the attribute set within the algorithm \texttt{ReduceAttributes}.
Let us have a schema $S(A, F)$. Simplify $F$ to minimal cover.

1. Find any key $K$ (see the 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-hand 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 continue from step 2. Otherwise we finish.
Formally: **Lucchesi-Osborn algorithm**

- having 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 FDs $F$, set of attr. $A$)
- returns set of all keys $Keys$

let all dependencies in $F$ be non-trivial

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

for each $K$ in $Keys$ do
  for each $X \rightarrow Y$ in $F$ do
    if $(Y \cap K \neq \emptyset$ and $\exists K' \in Keys : K' \subseteq (K \cup X) - Y)$ then
      $N := ReduceAttributes(F, ((K \cup X) - Y) \rightarrow A)$;
      $Keys := Keys \cup \{N\}$;
    endif
  endfor
endfor

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 the first key – Keys = {c}
2. **Iteration 1**: take jp → c that has a part of the last key on the right-hand 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.,
   Keys = {c, jp}
4. **Iteration 2**: take sd → p that has a part of the last key on the right-hand side (jp),
   {jsd} is not a super-set of c nor jp, i.e., it is a key candidate
5. in jsd → all we get redundant attribute s, i.e.,
   Keys = {c, jp, jd}
6. **Iteration 3**: take p → d, however, jp was already found so we do not add it
7. Finish 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 determination of keys is fast
  - thanks to limited size of $F$ and “separability“ of FDs
Design of database schemas

Two ways of modelling a relational database:

1. 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)

2. considering the whole database as a bag of (global) attributes results in a single universal database schema – i.e., one big table + a 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

- both approaches could be combined – i.e.,
  - create a conceptual database model
  - convert it to relational schemas
  - merge and/or normalize some of the schemas

Relational design – algorithms (A7B36DBS, Lect. 9)
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 into several single-column ones that surely are all in BCNF.

<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>

Clearly, there is something wrong with such a decomposition... ...it is lossy and it does not preserve dependencies
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 \( R_1(A_1, F_1), \ldots, R_n(A_n, F_n) \) is lossless, if \( R = \star_{i=1..n} R_i[A_i] \).
### 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>

<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**
- **Company, Data managed**
- **Company, Uses DBMS**
- **Company, Uses DBMS, Data managed**

"reconstruction" (natural join)
### 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>

**“reconstruction” (natural join)**
a decomposition property that ensures **no FD will be lost**

**Definition:**
Let $R_1(A_1, F_1), R_2(A_2, F_2)$ be 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

<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>

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>

Relational design – algorithms (A7B36DBS, Lect. 9)
The “Decomposition” algorithm

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

algorithm *Decomposition* (set of elem. FDs $F$, 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 $\exists R_i(F_{i_{-A}}, A_i) \in$ Result not being in BCNF then
    Let $X \rightarrow Y \in F_i$ such that $X \rightarrow A_i \not\in F^+$.
    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))\}
    // if there is a schema in the result violating BCNF
    // we remove the schema being decomposed
    // we add the schema being decomposed without attributes $Y$
    // we add the schema with attributes $XY$
  else
    Done := *true*;
  endwhile
return Result;

Note: Function $\text{cover}(X, F)$ 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 $F^+$. 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).
Example – decomposition

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}
The “Synthesis” algorithm

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

algorithm Synthesis(set of elem. FDs F, set of attributes A) : returns set \{R_i(F_i, A_i)\}
  
  G = minimal cover of F
  
  compose FDs having equal left-hand 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)!
  - i.e., we can try to minimize the number of relations
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.
- Reundant FDs \( c \rightarrow s \) and \( c \rightarrow p \) were removed.
- \( G = \{ c \rightarrow j, c \rightarrow d, c \rightarrow q, c \rightarrow v, sd \rightarrow p, p \rightarrow d, jp \rightarrow c, j \rightarrow s \} \)

Composition:

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

Result:

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

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

\( R_2(\{sdp\}, \{sd \rightarrow p, p \rightarrow d\}), R_5(\{js\}, \{j \rightarrow s\}) \)

merging \( R_4 \) and \( R_5 \)

(has however, now \( p \rightarrow d \) violates BCNF)