7. Design Theory for Relational Databases

7.1 Introduction

- so far: methodical design from an E-R schema to a relational database schema

- Questions
  - What does a good conceptual database schema look like?
  - How can the quality of a database schema be assessed?

- hence: conceptual fine tuning of the generated relational database schema on the basis of formal methods
  - functional dependencies as generalization of the key concept
  - multi-valued dependencies as generalization of functional dependencies
  - normal forms for relation schemas on the basis of functional dependencies
    + goal: estimation of the “quality” of a relation schema
    + If a relation schema does not fulfil these normal forms, so-called normalization algorithms can be applied to it in order to decompose it into several new schemas where each resulting schema fulfils the corresponding normal form.
Class Discussion

- The task for you as database experts is to assess the “quality” of the following two database schemas given as

  - database schema 1
    - customer(cname, caddr, account)
    - order(cname, product, amount)
    - supplier(sname, saddr, product, price)

  - database schema 2
    - customer-addr(cname, caddr)
    - customer-account(cname, account)
    - order(cname, product, amount)
    - supplier(sname, saddr)
    - offer(sname, product, price)

Which quality criteria have you taken? Why? Which schema is better in your opinion? Why?
- problem discussion of database schema 1 by an example
  - supplier(sname, saddr, product, price)
    + redundancies
      For each product the address of the supplier is stored.
    + update anomalies
      The address of a supplier can be changed in one of its tuples but remains unchanged in another tuple (→ inconsistency).
    + insertion anomalies
      A supplier’s address cannot be inserted without a product.
    + deletion anomalies
      The deletion of the last product offered by a supplier leads to a loss of the supplier’s address.
  - intuitive improvement
    supplier(sname, saddr)
    offer(sname, product, price)
- advantages
  + no redundancy
  + no anomalies
- But: For finding the supplier’s address of a product, an expensive join is needed.

**Design goals**

- avoidance of redundancy and anomalies
- avoidance of information loss
- maybe consideration of efficiency aspects
7.2 Functional Dependencies

Proceeding

- Input: DB schema + functional dependencies
- decomposition of the given database schema into an equivalent schema without redundancy and anomalies (normalization)
- integrity constraints: conditions for the permitted instances of a database schema
- A functional dependency (FD) is a special integrity constraint.
- definition of functional dependency
  
  Let $R$ be the relation schema of a relation $R$, and let $A, B \subseteq R$. $B$ is functionally dependent on $A$, or $A$ determines $B$ functionally (uniquely), written $A \rightarrow B$ if, and only if, to each value in $A$ exactly one value in $B$ belongs:

  $A \rightarrow B \iff \forall t_1, t_2 \in R : t_1[A] = t_2[A] \Rightarrow t_1[B] = t_2[B]$ 

  for all possible relations $R$ over $R$.

- A functional dependency depends on the semantics of the schema, and not on the instance of a relation.
example:
- supplier(sname, saddr, product, price)
- functional dependencies:
  + \{sname\} → \{saddr\}
    A supplier's name determines uniquely the supplier's address.
  + \{sname, product\} → \{price\}
    The key \{sname, product\} determines uniquely the price.
  + \{sname\} → \{sname\}, \{sname, product\} → \{product\} are trivial.
  + \{sname, product\} → \{saddr\} is partial

A dependency is called **trivial** if \(B \subseteq A\) holds.

A functional dependency \(X \rightarrow Y\) is called **full**, if there is no proper subset \(Z \subset X\) so that \(Z \rightarrow Y\) holds. If such a subset exists, \(X \rightarrow Y\) is called **partial** dependency.

Let \(X, Y \subset R\), and let \(X \rightarrow Y\). Let \(A \subset R\) be an attribute set, and let \(Y \rightarrow A\). Then \(A\) is **transitively dependent** on \(X\), i.e., \(X \rightarrow A\).
Find the functional dependencies of \textit{StaffBranch} \textbf{from the data.}\n
<table>
<thead>
<tr>
<th>staffNo</th>
<th>sName</th>
<th>position</th>
<th>salary</th>
<th>branchNo</th>
<th>bAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John White</td>
<td>Manager</td>
<td>30000</td>
<td>B005</td>
<td>22 Deer Rd, London</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann Beech</td>
<td>Assistant</td>
<td>12000</td>
<td>B003</td>
<td>163 Main St, Glasgow</td>
</tr>
<tr>
<td>SG14</td>
<td>David Ford</td>
<td>Supervisor</td>
<td>18000</td>
<td>B003</td>
<td>163 Main St, Glasgow</td>
</tr>
<tr>
<td>SA9</td>
<td>Mary Howe</td>
<td>Assistant</td>
<td>9000</td>
<td>B007</td>
<td>16 Argyll St, Aberdeen</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan Brand</td>
<td>Manager</td>
<td>24000</td>
<td>B003</td>
<td>163 Main St, Glasgow</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie Lee</td>
<td>Assistant</td>
<td>9000</td>
<td>B005</td>
<td>22 Deer Rd, London</td>
</tr>
</tbody>
</table>
**StaffBranch**

<table>
<thead>
<tr>
<th>staffNo</th>
<th>sName</th>
<th>position</th>
<th>salary</th>
<th>branchNo</th>
<th>bAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John White</td>
<td>Manager</td>
<td>30000</td>
<td>B005</td>
<td>22 Deer Rd, London</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann Beech</td>
<td>Assistant</td>
<td>12000</td>
<td>B003</td>
<td>163 Main St, Glasgow</td>
</tr>
<tr>
<td>SG14</td>
<td>David Ford</td>
<td>Supervisor</td>
<td>18000</td>
<td>B003</td>
<td>163 Main St, Glasgow</td>
</tr>
<tr>
<td>SA9</td>
<td>Mary Howe</td>
<td>Assistant</td>
<td>9000</td>
<td>B007</td>
<td>16 Argyll St, Aberdeen</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan Brand</td>
<td>Manager</td>
<td>24000</td>
<td>B003</td>
<td>163 Main St, Glasgow</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie Lee</td>
<td>Assistant</td>
<td>9000</td>
<td>B005</td>
<td>22 Deer Rd, London</td>
</tr>
</tbody>
</table>

Find the functional dependencies of *StaffBranch* from the data.

\[
\{\text{staffNo}\} \rightarrow \{\text{sName, position, salary, branchNo, bAddress}\} \\
\{\text{sName}\} \rightarrow \{\text{staffNo, position, salary, branchNo, bAddress}\} \\
\{\text{branchNo}\} \rightarrow \{\text{bAddress}\} \\
\{\text{bAddress}\} \rightarrow \{\text{branchNo}\} \\
\{\text{branchNo, position}\} \rightarrow \{\text{salary}\} \\
\{\text{bAddress, position}\} \rightarrow \{\text{salary}\}\]
Find the functional dependencies of StaffBranch that make sense.

\[
\begin{align*}
\{\text{staffNo}\} & \rightarrow \{\text{sName, position, salary, branchNo, bAddress}\} \\
\{\text{sName}\} & \rightarrow \{\text{staffNo, position, salary, branchNo, bAddress}\} \\
\{\text{branchNo}\} & \rightarrow \{\text{bAddress}\} \\
\{\text{bAddress}\} & \rightarrow \{\text{branchNo}\} \\
\{\text{branchNo, position}\} & \rightarrow \{\text{salary}\} \\
\{\text{bAddress, position}\} & \rightarrow \{\text{salary}\}
\end{align*}
\]
A functional dependency (FD) on a relation \( R \) is a statement of the form:

If two or more tuples of \( R \) agree on the attribute values \( A_1, A_2, \ldots, A_n \) (i.e., the tuples have the same values for each of these attributes), then they must also agree on another attribute \( B \). We write \( \{A_1, A_2, \ldots, A_n\} \rightarrow \{B\} \)

Consider the schema Movies (title, year, length, filmType, studioName, starName)
Possible FDs:
\( \{\text{title, year}\} \rightarrow \{\text{length}\}, \{\text{title, year}\} \rightarrow \{\text{filmType}\}, \{\text{title, year}\} \rightarrow \{\text{studioName}\} \)

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
<th>filmType</th>
<th>studioName</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>124</td>
<td>color</td>
<td>Fox</td>
<td>Carrie Fisher</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>124</td>
<td>color</td>
<td>Fox</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>124</td>
<td>color</td>
<td>Fox</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Might Ducks</td>
<td>1991</td>
<td>104</td>
<td>color</td>
<td>Disney</td>
<td>Emilio Estevez</td>
</tr>
<tr>
<td>Wayne’s World</td>
<td>1992</td>
<td>95</td>
<td>color</td>
<td>Paramount</td>
<td>Dana Carvey</td>
</tr>
<tr>
<td>Wayne’s World</td>
<td>1992</td>
<td>95</td>
<td>color</td>
<td>Paramount</td>
<td>Mike Meyers</td>
</tr>
</tbody>
</table>
We see that we can summarize the right sides:
\[ \{\text{title, year}\} \rightarrow \{\text{length, filmType, studioName}\} \]

What about the validity of \( \{\text{title, year}\} \rightarrow \{\text{starName}\} \)? Is it a FD?

A set of one or more attributes \( \{A_1, A_2, \ldots, A_n\} \) is a **key** for a relation schema \( R \) if

1. \( \{A_1, A_2, \ldots, A_n\} \rightarrow R \ \backslash \ \{A_1, A_2, \ldots, A_n\} \)
   
   These attributes functionally determine all other attributes of \( R \). We can also say:
   
   \[ \{A_1, A_2, \ldots, A_n\} \rightarrow R \]

2. There is no proper subset \( C \subset \{A_1, A_2, \ldots, A_n\} \) such that \( C \rightarrow R \).

Determine an attribute set that forms a key for Movies.
Checking the preservation of a functional dependency

- Alternative characterization of a FD $A \rightarrow B$

  Let $A = \{A_1, ..., A_n\}$, and let $\text{dom}(A) = \text{dom}(A_1) \times ... \times \text{dom}(A_n)$.

  The FD $A \rightarrow B$ holds on $R$ if $\forall \ v \in \text{dom}(A) : |\pi_B(\sigma_{A=v}(R))| \leq 1$

  ($A = v$ stands for $A_1 = v_1 \land ... \land A_n = v_n$)

- This leads to a simple algorithm which computes whether a given relation $R$ satisfies a given FD $A \rightarrow B$:

  ![Algorithm for checking the preservation of a functional dependency](algorithm.png)

  ```
  algorithm FDPreservation($R$, $A \rightarrow B$)
  // input: relation $R$ and FD $A \rightarrow B$
  // output: true, if $A \rightarrow B$ holds on $R$; false otherwise
  sort $R$ with respect to $A$-values
  if all groups consisting of tuples with equal $A$-values also have equal $B$-values then
      return true
  else
      return false
  end.
  ```