Insertion, change and deletion

- commands “I.” and “D.” are used for the insertion and deletion of tuples.

- example: Smith orders 20 pieces of an article which has been ordered by Jones more than hundred times.

<table>
<thead>
<tr>
<th>order</th>
<th>cname</th>
<th>product</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>Smith</td>
<td>_W</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Jones</td>
<td>_W</td>
<td>&gt; 100</td>
</tr>
</tbody>
</table>

- Command “U.” is used for updating the values in non-key attributes.

- Empty fields are not changed.

- Values of the key attributes must be specified.

- example: Jones prices milk to $1.30.

<table>
<thead>
<tr>
<th>vendor</th>
<th>vname</th>
<th>vaddr</th>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.</td>
<td>Jones</td>
<td></td>
<td>milk</td>
<td>1.30</td>
</tr>
</tbody>
</table>
example: Benson increases all his prices by 10%.

<table>
<thead>
<tr>
<th>vendor</th>
<th>vname</th>
<th>vaddr</th>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.</td>
<td>Benson</td>
<td>_W</td>
<td>1.1 * _P</td>
<td></td>
</tr>
<tr>
<td>Benson</td>
<td>_W</td>
<td>_P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Creation of relation and views

A relation is specified as follows:

<table>
<thead>
<tr>
<th>l. customer l.</th>
<th>cname</th>
<th>caddr</th>
<th>account</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY l.</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>TYPE l.</td>
<td>CHAR</td>
<td>CHAR</td>
<td>FIXED</td>
</tr>
<tr>
<td>INVERSION l.</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

- KEY: definition of key attributes
- TYPE: data type of an attribute
- INVERSION: index structure for this attribute created?
creation of a view

<table>
<thead>
<tr>
<th>customer</th>
<th>cname</th>
<th>caddr</th>
<th>account</th>
</tr>
</thead>
<tbody>
<tr>
<td>_N</td>
<td>_A</td>
<td>_K</td>
<td></td>
</tr>
</tbody>
</table>

I.VIEW good_customers I.

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>account</th>
</tr>
</thead>
<tbody>
<tr>
<td>_N</td>
<td>_A</td>
<td>_K &gt; 0</td>
</tr>
</tbody>
</table>
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.
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:

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

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:
- \textit{supplier(sname, saddr, product, price)}
- functional dependencies:
  + \{sname\} \rightarrow \{saddr\}
    A supplier’s name determines uniquely the supplier’s address.
  + \{sname, product\} \rightarrow \{price\}
    The key \{sname, product\} determines uniquely the price.
  + \{sname\} \rightarrow \{sname\}, \{sname, product\} \rightarrow \{product\} are trivial.
  + \{sname, product\} \rightarrow \{saddr\} is partial

A dependency is called \textbf{trivial} if \(B \subseteq A\) holds.

A functional dependency \(X \rightarrow Y\) is called \textbf{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 \textbf{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 \textbf{transitively dependent} on \(X\), i.e., \(X \rightarrow A\).