Foundations of Information Systems

4 Improving Database Design
Goals of Database Design

Primary issue of database design

- How does a good conceptional schema look like?
- How can we measure the “quality” of database schemas?

Example

- Customer(CName, CAddr, Account#)
- Order(CName, Item, Amount)
- Supplier(SName, SAddr, Item, Price)

Creating “good” relational schemas → Normalization

<table>
<thead>
<tr>
<th>Supplier</th>
<th>SName</th>
<th>SAddr</th>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michl</td>
<td>Munich</td>
<td></td>
<td>DVD drives</td>
<td>110</td>
</tr>
<tr>
<td>Kohl</td>
<td>Frankfurt</td>
<td></td>
<td>DVD drives</td>
<td>115</td>
</tr>
<tr>
<td>Kohl</td>
<td>Frankfurt</td>
<td></td>
<td>CD drives</td>
<td>80</td>
</tr>
<tr>
<td>Keller</td>
<td>Stuttgart</td>
<td></td>
<td>CD drives</td>
<td>85</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Motivation

Drawback of denormalized relational schemas (e.g. supplier)

- **Redundancy**, e.g. for each item the address of the supplier is stored
- **Update anomaly**, e.g. it is not possible to change the address of a supplier in one tuple without changing all other tuples
- **Insert anomaly**, e.g. it is not possible to insert a supplier address without inserting an item
- **Delete anomaly**, e.g. deleting an item also deletes the supplier address → inconsistencies possible

Alternative relational schemas possible

- CustomerAddr(CName, CAddr)
- CustomerAccount(CName, Account#)
- Order(CName, Item, Amount)
- Supplier(SName, SAddr)
- Offer(SName, Item, Price)
Benefits after normalization?

- **Pro’s**
  - No redundancy, no anomalies
- **Con’s**
  - To find the supplier address for an item a join is necessary

<table>
<thead>
<tr>
<th>Supplier</th>
<th>SName</th>
<th>SAddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michl</td>
<td>Munich</td>
<td></td>
</tr>
<tr>
<td>Kohl</td>
<td>Frankfurt</td>
<td></td>
</tr>
<tr>
<td>Keller</td>
<td>Stuttgart</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offer</th>
<th>SName</th>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michl</td>
<td>DVD drives</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>Kohl</td>
<td>DVD drives</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>Kohl</td>
<td>CD drives</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Keller</td>
<td>CD drives</td>
<td>85</td>
<td></td>
</tr>
</tbody>
</table>

... ... ...
Normalization Goals

Several sometimes opposing goals

- Avoid redundancies and anomalies
- Avoid information losses
- (Inclusion of efficiency considerations)

Foundations

- DB schema + functional dependencies (definition on the following slides)

Process

- Split a given database schema in an equivalent schema without redundancies and anomalies
Theory of Functional Dependencies
**Integrity constraints**

- Constraints on the permitted instances of a database schema
- A functional dependency (FD) is a specific type of integrity constraints

**Definition of Functional Dependencies**

- $A$ and $B$ are attribute sets of the relational schema $RS$ with $A, B \subseteq RS$
- $B$ is functional dependent from an $A$
  (or $A$ functional determines $B$, or $A$ is determinant of $B$)
  iff for all possible relations $r(RS)$ each value in $A$ belongs to exactly one value in $B$

$$A \rightarrow B \iff \forall t_1, t_2 \in r(RS) : t_1[A] = t_2[A] \Rightarrow t_1[B] = t_2[B]$$

**Important:**

- Functional dependencies are derived from the schema semantics not from the current instances/tuples of a relation!
Example

- Supplier(SName, SAddr, Item, Price)
- List of functional dependencies
  - {SName} → {SAddr}  (a suppliers name determines his address)
  - {SName, Item} → {Price}  (the key {SName, Item} determines the price)
  - {SName} → {SName}  (trivial)
  - {SName, Item} → {Item}  (trivial)
  - {SName, Item} → {SAddr}  (partial)

Trivial, Full and Partial Functional Dependency

- An dependency A → B is called trivial, if B ⊆ A
- X → Y is a full functional dependency when no true subset Z ⊆ X exists, such that: Z → Y, we write X → Y and X is a candidate key
- If such a subset Z exists then X → Y is a partial dependency
**Computation of FDs**

*Transitive Dependencies*
- X and Y are attribute sets of RS (X, Y ⊂ RS) with X → Y. A ∈ RS is an attribute with A ∉ X, Y and Y → A. Then A is transitive dependent from X: X → A

*Transitive Closure of F*
- The Transitive Closure F⁺ is the set of all functional dependencies, which can be derived from the set of functional dependencies F.

*Armstrong Axioms*
- F⁺ is computed by applying the following rules (F is a set of FDs and A, B, C ⊂ RS):
  - **Reflexivity:** If B ⊆ A then A → B is always true (special case: A → A)
  - **Augmentation:** If A → B, then also A ∪ C → B ∪ C
  - **Transitivity:** If A → B and B → C, then also A → C

- It can be shown that the Armstrong Axioms are correct and complete
Extension of the Armstrong Axioms

- **Union:** If \( A \rightarrow B \) and \( A \rightarrow C \) is true, then also \( A \rightarrow B \cup C \) applies
- **Decomposition:** If \( A \rightarrow B \cup C \) is true, then also \( A \rightarrow B \) and \( A \rightarrow C \) applies
- **Pseudo transitivity:** If \( A \rightarrow B \) and \( B \cup C \rightarrow D \) is true, then also \( A \cup C \rightarrow D \) applies

Example

- Given a relation Supplier(SName, SAddr, Item, Price) and FD \( \{\text{SName}\} \rightarrow \{\text{SAddr}\} \)
- We want to show that \( \{\text{SName, Item}\} \rightarrow \{\text{SAddr}\} \) is also true
  - Starting point: \( \{\text{SName}\} \rightarrow \{\text{SAddr}\} \)
  - Applying the 2nd Armstrong Axiom: \( \{\text{SName, Item}\} \rightarrow \{\text{SAddr, Item}\} \)
  - Applying the Decomposition rule: \( \{\text{SName, Item}\} \rightarrow \{\text{SAddr}\} \)
Membership Problem

**Problem**

- Given a set of functional dependencies $F$ and $A \rightarrow B$
- Is $A \rightarrow B \in F^+$ true? (Or: is $A \rightarrow B$ a member of $F^+$?)
- Computation of $F^+$ very costly

**Solution**

- Compute the transitive closure $A^+$ of the attribute set $A$ regarding $F$
  - $A^+$ consists of all attributes that are functional determined by $A$
  - If $B \subseteq A^+$ is true, then also $A \rightarrow B \in F^+$ applies

**Algorithm: Closure($F, A$)**

```plaintext
res := A; // because A \rightarrow A
WHILE (Changes to res) DO
    FOREACH FD (B \rightarrow C) \in F DO
        IF B \subseteq res THEN res := res \cup C;
    RETURN A+ = res;
```

**Application**

- Check whether $A_k$ is a primary key candidate $\rightarrow$ Closure($F, A_k$) = RS
Equivalence of Functional Dependencies

- Two FD sets F and G of a relational schema R are equivalent, if $F^+ = G^+$ is true.

Problem

- Find the minimal set of FDs
- To minimize the overhead for checking whether a tuple violates a constraint

Canonical Cover

- The set of FDs $F_c$ is called the canonical cover of F, if following conditions are met:
  - $F_c^+ = F^+$
  - For all FDs $A \rightarrow B$ in $F_c$ there are no redundant attributes in $A$ and in $B$, i.e.
    - for all attributes $C$ from $A$: $(F_c - \{A \rightarrow B\} \cup \{(A - \{C\}) \rightarrow B\})^+ \neq F^+$
    - for all attributes $D$ from $B$: $(F_c - \{A \rightarrow B\} \cup \{(A \rightarrow (B - \{D\}))\})^+ \neq F^+$
  - Each left side of FDs in $F_c$ occurs only once, i.e.,
    - if $A \rightarrow B$ and $A \rightarrow C$, then $F_c$ contains just $A \rightarrow B \cup C$ (Union Axiom)
  - Casually: $F_c$ is the canonical cover of F,
    - if no FD in $F_c$ can be reduced either left or right without giving up the $F^+$
    - and no FDs in F share a left side
Determine the Canonical Cover

**Step 1: Left-reduction**
- Apply left-reduction to all FD \( (A \rightarrow B) \in F \) by checking for each \( X \in A \) whether attribute \( X \) is redundant, i.e. following is true:
  \[
  B \subseteq \text{Closure}(F, A \setminus \{X\})
  \]
  If that applies replace \( A \rightarrow B \) with \( A \setminus \{X\} \rightarrow B \)

**Step 2: Right-reduction**
- Apply right-reduction to all FD \( (A \rightarrow B) \in F \) by checking for each \( Y \in B \) whether attribute \( Y \) is redundant, i.e. following is true:
  \[
  Y \in \text{Closure}(F \setminus (A \rightarrow B) \cup (A \rightarrow B \setminus \{Y\}), A)
  \]
  If that applies replace \( A \rightarrow B \) with \( A \rightarrow B \setminus \{Y\} \)

**Step 3: Cleanup**
- Remove all FDs with \( A \rightarrow \emptyset \)
- Replace all FDs like \( A \rightarrow B_1, A \rightarrow B_2, \ldots, A \rightarrow B_k \) with \( A \rightarrow B_1 \cup B_2 \cup \ldots \cup B_k \)
Example

- Set \( F = \{A \rightarrow B, B \rightarrow C, A \cup B \rightarrow C\} \)

Determining the canonical cover (alternative 1)

- Step 1: Left-reduction
  - \( C \subseteq \text{Closure}(\{A \rightarrow B, B \rightarrow C, B \rightarrow C\}, A \cup B)? \rightarrow \text{yes (pseudo transitivity)} \)

- Step 2: Right-reduction
  - \( B \subseteq \text{Closure}(\{A \rightarrow \emptyset, B \rightarrow C, B \rightarrow C\}, A)? \rightarrow \text{no!} \)
  - \( C \subseteq \text{Closure}(\{A \rightarrow B, B \rightarrow \emptyset, B \rightarrow C\}, B)? \rightarrow \text{yes (trivial)} \)

- Step 3: Cleanup
  - Removing \( B \rightarrow \emptyset \), keeping \( \{A \rightarrow B, B \rightarrow C\} \)
  - No aggregations possible
Example

- Set $F = \{A \rightarrow B, B \rightarrow C, A \cup B \rightarrow C\}$

Determining the canonical cover (alternative 2)

- Step 1: Left-reduction
  - $C \subseteq \text{Closure}(\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}, A \cup B)$? $\rightarrow$ yes (pseudo transitivity)

- Step 2: Right-reduction
  - $B \subseteq \text{Closure}(\{A \rightarrow \emptyset, B \rightarrow C, A \rightarrow C\}, A)$? $\rightarrow$ no!
  - $C \subseteq \text{Closure}(\{A \rightarrow B, B \rightarrow \emptyset, A \rightarrow C\}, B)$? $\rightarrow$ no!
  - $C \subseteq \text{Closure}(\{A \rightarrow B, B \rightarrow C, A \rightarrow \emptyset\}, A)$? $\rightarrow$ yes (transitivity)

- Step 3: Cleanup
  - Removing $A \rightarrow \emptyset$, keeping $\{A \rightarrow B, B \rightarrow C\}$
  - No aggregations possible
Decomposing Relational Schemas

Segmentation properties

- To avoid anomalies the relational schema $R$ is decomposed into smaller relational schemas $R_1, ..., R_n$
- Without information loss, i.e. relation $r(R)$ must be reconstructible from relations $r(R_1), ..., r(R_n)$
- All FDs valid for schema $R$, must be valid for $R_1, ..., R_n$, too

Information Loss

- A segmentation of schemas $R$ into $R_1, ..., R_n$ is lossless, if: $R = \pi_{R_1}(R) \bowtie ... \bowtie \pi_{R_n}(R)$

Lossless Splitting

- Given a relational schema $R$ and a set of FD’s $F_R$
- A decomposition of $R$ into $R_1$ and $R_2$ is lossless, if:
  \[(R_1 \cap R_2 \rightarrow R_1) \in F_R^+ \quad \text{or} \quad (R_1 \cap R_2 \rightarrow R_2) \in F_R^+\]
- In other words: $R = \alpha \cup \beta \cup \gamma$ is split into $R_1 = \alpha \cup \beta$ and $R_2 = \alpha \cup \gamma$, then:
  \[\beta \subseteq \text{Closure}(F_R, \alpha) \quad \text{or} \quad \gamma \subseteq \text{Closure}(F_R, \alpha)\]
Example

- Decomposition of $R(SName, SAddr, Item, Price)$ into
  - $Supplier(SName, SAddr, Item)$
  - $Offer(Item, Price)$
- Not lossless, i.e. $R \neq Supplier \bowtie Offer$
- Reasons
  - Item does not functionally determine Price
  - Item does not functionally determine $SName, Saddr$
- Casually: Two suppliers may offer the same item at different prices

Challenge

- All FDs, valid for schema $R$, should be easy checkable on the decomposed, local schemas $R_1, ..., R_n$
  $\Rightarrow$ preservation of functional dependencies
**Dependency Preservation**

- The decomposition of a relational schema $R$ into $R_1, \ldots, R_n$ preserves the functional dependencies, if

\[ F_R^+ = (F_{R_1} \cup \ldots \cup F_{R_n})^+ \]

**Example**

- Given the schema $R(Street, City, State, Zip)$ with following constraints
  - Cities are uniquely defined through City (their name) and State
  - Within a street the zip code always keeps the same
  - Zip code areas do not exceed city borders and cities do not exceed state borders

- FDs: $\{Zip\} \rightarrow \{City, State\}$ and $\{Street, City, State\} \rightarrow \{Zip\}$
- Properties of the decomposition $\{Zip, Street\}$ and $\{Zip, City, State\}$
  - Lossless
  - Not dependency preserving