Teil V

Relational Database Design Theory
Relational Database Design Theory

1. Target Model of the Logical Design
Relational Database Design Theory

1. Target Model of the Logical Design

2. Relational DB Design
Relational Database Design Theory

1. Target Model of the Logical Design
2. Relational DB Design
3. Normal Forms
Relational Database Design Theory

1. Target Model of the Logical Design
2. Relational DB Design
3. Normal Forms
4. Transformation Properties
Relational Database Design Theory

1. Target Model of the Logical Design
2. Relational DB Design
3. Normal Forms
4. Transformation Properties
5. Design Methods
Educational Objective for Today . . .

- Know how to refine the relational design
Educational Objective for Today . . .

- Know how to refine the relational design
- Understanding of normal forms
Educational Objective for Today . . .

- Know how to refine the relational design
- Understanding of normal forms
- Methodology and techniques for normalization
## Relation Model

### WINES

<table>
<thead>
<tr>
<th>WineID</th>
<th>Name</th>
<th>Color</th>
<th>Vintage</th>
<th>Vineyard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1042</td>
<td>La Rose Grand Cru</td>
<td>Rot</td>
<td>1998</td>
<td>Château La Rose</td>
</tr>
<tr>
<td>2168</td>
<td>Creek Shiraz</td>
<td>Rot</td>
<td>2003</td>
<td>Creek</td>
</tr>
<tr>
<td>3456</td>
<td>Zinfandel</td>
<td>Rot</td>
<td>2004</td>
<td>Helena</td>
</tr>
<tr>
<td>2171</td>
<td>Pinot Noir</td>
<td>Rot</td>
<td>2001</td>
<td>Creek</td>
</tr>
<tr>
<td>3478</td>
<td>Pinot Noir</td>
<td>Rot</td>
<td>1999</td>
<td>Helena</td>
</tr>
<tr>
<td>4711</td>
<td>Riesling Reserve</td>
<td>Weiß</td>
<td>1999</td>
<td>Müller</td>
</tr>
<tr>
<td>4961</td>
<td>Chardonnay</td>
<td>Weiß</td>
<td>2002</td>
<td>Bighorn</td>
</tr>
</tbody>
</table>

### PRODUCER

<table>
<thead>
<tr>
<th>Vineyard</th>
<th>District</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
<tr>
<td>Château La Rose</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
</tr>
<tr>
<td>Château La Pointe</td>
<td>Pomerol</td>
<td>Bordeaux</td>
</tr>
<tr>
<td>Müller</td>
<td>Rheingau</td>
<td>Hessen</td>
</tr>
<tr>
<td>Bighorn</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
</tbody>
</table>
### Terms of the Relational Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Informal Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Column of a table</td>
</tr>
<tr>
<td>Value domain</td>
<td>Possible values of an attribute</td>
</tr>
<tr>
<td>Attribute value</td>
<td>Element of a value domain</td>
</tr>
<tr>
<td>Relation schema</td>
<td>Set of attributes</td>
</tr>
<tr>
<td>Relation</td>
<td>Set of rows in a table</td>
</tr>
<tr>
<td>Tuple</td>
<td>Row in a table</td>
</tr>
<tr>
<td>Database schema</td>
<td>Set of relation schemas</td>
</tr>
<tr>
<td>Database</td>
<td>Set of relations (base relations)</td>
</tr>
</tbody>
</table>
## Terms of the Relational Model /2

<table>
<thead>
<tr>
<th>Term</th>
<th>Informal Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Minimal set of attributes, whose values uniquely identify a tuple in a table</td>
</tr>
<tr>
<td>Primary key</td>
<td>A key designated during database design</td>
</tr>
<tr>
<td>Foreign key</td>
<td>Set of attributes that are key in another relation</td>
</tr>
<tr>
<td>Foreign key constraint</td>
<td>All attribute values of the foreign key show up as keys in the other relation</td>
</tr>
</tbody>
</table>
Integrity Constraints

- Identifying set of attributes $K := \{B_1, \ldots, B_k\} \subseteq R:
  \[
  \forall t_1, t_2 \in r \ [t_1 \neq t_2 \implies \exists B \in K : t_1(B) \neq t_2(B)]
  \]

- **Key**: is minimal identifying set of attributes
  - \{Name, Vintage, Vineyard\} and
  - \{WineID\} for WINES

- **Prime attribute**: element of a key

- **Primary key**: designated key

- **Superkey**: every superset of a key (= identifying set of attributes)

- **Foreign key**: $X(R_1) \rightarrow Y(R_2)$
  \[
  \{t(X) | t \in r_1\} \subseteq \{t(Y) | t \in r_2\}
  \]
Relation with Redundancies

<table>
<thead>
<tr>
<th>WINES</th>
<th>WineID</th>
<th>Name</th>
<th>...</th>
<th>Vineyard</th>
<th>District</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINES</td>
<td>1042</td>
<td>La Rose Gr. Cru</td>
<td>...</td>
<td>Ch. La Rose</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
</tr>
<tr>
<td></td>
<td>2168</td>
<td>Creek Shiraz</td>
<td>...</td>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td></td>
<td>3456</td>
<td>Zinfandel</td>
<td>...</td>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
<tr>
<td></td>
<td>2171</td>
<td>Pinot Noir</td>
<td>...</td>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td></td>
<td>3478</td>
<td>Pinot Noir</td>
<td>...</td>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
<tr>
<td></td>
<td>4711</td>
<td>Riesling Res.</td>
<td>...</td>
<td>Müller</td>
<td>Rheingau</td>
<td>Hessen</td>
</tr>
<tr>
<td></td>
<td>4961</td>
<td>Chardonnay</td>
<td>...</td>
<td>Bighorn</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
</tbody>
</table>
Update Anomalies

- Insertion into the redundancy-containing relation WINES:

```sql
insert into WINES (WineID, Name, Color, Vintage,
                  Vineyard, District, Region)
values (4711, 'Chardonnay', 'Weiß', 2004,
        'Helena', 'Rheingau', 'California')
```

- WineID 4711 already assigned to another wine: violates FD
  WineID → Name
- Up to now, vineyard Helena was located in Napa Valley: violates FD
  Vineyard → District
- Rheingau is not located in California: violates FD
  District → Region

- Also: update- and delete anomalies
Functional Dependencies

- **Functional dependency** between two sets of attribute $X$ and $Y$ of a relation holds iff

  for each tuple of the relation, the attribute values of the $X$ components determine the attribute values of the $Y$ components.

- If two tuples have the same values for the $X$ attributes, they also have the same values for all $Y$ attributes.

- **Notation for functional dependency (FD):** $X \rightarrow Y$

- **Example:**
  - WineID $\rightarrow$ Name, Vineyard
  - District $\rightarrow$ Region

- **But not:** Vineyard $\rightarrow$ Name
Keys as a Special Case

- For example on Slide 5-7
  
  WineID → Name, Color, Vintage, Vineyard, District, Region

- Always: WineID → WineID,
  then whole schema on the right side

- If left side minimal: Key

- Formally: $X$ is key if FD $X \rightarrow R$ holds for relation schema $R$ and $X$ is minimal

Goal of database design: Transform all existing functional dependencies into “key dependencies”, without losing semantic information
Deriving FDs

<table>
<thead>
<tr>
<th>r</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td></td>
</tr>
<tr>
<td>a₂</td>
<td>b₁</td>
<td>c₁</td>
<td></td>
</tr>
<tr>
<td>a₃</td>
<td>b₂</td>
<td>c₁</td>
<td></td>
</tr>
<tr>
<td>a₄</td>
<td>b₁</td>
<td>c₁</td>
<td></td>
</tr>
</tbody>
</table>

- Satisfies $A \rightarrow B$ and $B \rightarrow C$
- Then $A \rightarrow C$ also holds
- Not derivable: $C \rightarrow A$ or $C \rightarrow B$
Deriving FDs /2

- If for $f$ over $R$, it holds that $\text{SAT}_R(F) \subseteq \text{SAT}_R(f)$, then $F$ implies the FD $f$ (short: $F \models f$)
- Previous example:

$$F = \{A \rightarrow B, B \rightarrow C\} \models A \rightarrow C$$

- Computing the closure: Determine all functional dependencies that can be derived from a given set of FDs
- Closure $F^+_R := \{f \mid (f \text{ FD over } R) \land F \models f\}$
- Example:

$$\{A \rightarrow B, B \rightarrow C\}^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C, AB \rightarrow C, A \rightarrow BC, \ldots, AB \rightarrow AB, \ldots\}$$
Derivation Rules

F1  Reflexivity  \[ \forall X \subseteq Y \Rightarrow X \rightarrow Y \]
F2  Augmentation  \[ \{X \rightarrow Y\} \Rightarrow XZ \rightarrow YZ \text{ and } XZ \rightarrow Y \]
F3  Transitivity  \[ \{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z \]
F4  Decomposition  \[ \{X \rightarrow YZ\} \Rightarrow X \rightarrow Y \]
F5  Union  \[ \{X \rightarrow Y, X \rightarrow Z\} \Rightarrow X \rightarrow YZ \]
F6  Pseudo-transitivity  \[ \{X \rightarrow Y, WY \rightarrow Z\} \Rightarrow WX \rightarrow Z \]

F1-F3 known as Armstrong axioms (sound, complete)

- **Sound**: Rules do not derive FDs that are not logically implied
- **Complete**: All implied FDs are derived
- **Independent** (i.e., minimal w.r.t. \( \subseteq \)): No rule can be omitted

\[^1\text{w.r.t. } = \text{ with respect to} \]
Alternative Set of Rules

- **B-Axioms or RAP-rules**

  **R** Reflexivity \[ \{\} \implies X \rightarrow X \]

  **A** Accumulation \[ \{X \rightarrow YZ, Z \rightarrow AW\} \implies X \rightarrow YZA \]

  **P** Projectivity \[ \{X \rightarrow YZ\} \implies X \rightarrow Y \]

- Rule set is complete because it allows to derive the Armstrong axioms
Membership Problem

Can a certain FD $X \rightarrow Y$ be derived from a given set $F$, i.e., is it implied by $F$?

Membership problem: “$X \rightarrow Y \in F^+$ ?”

- Closure over a set of attributes $X$ w.r.t. $F$ is $X_F^+ := \{ A \mid X \rightarrow A \in F^+ \}$
- Membership problem can be solved in linear time by solving the modified problem
  
  Membership problem (2): “$Y \subseteq X_F^+$ ?”
Algorithm **CLOSURE**

- Compute $X_F^+$, the closure of $X$ w.r.t. $F$

```plaintext
CLOSURE(F, X):
    X^+ := X
    repeat
        X~+~ := X^+  /* R-rule */
        forall FDs $Y \rightarrow Z \in F$
            if $Y \subseteq X^+$ then $X^+ := X^+ \cup Z$  /* A-rule */
    until $X^+ = \overline{X}^+$
    return $X^+$
```

**MEMBER**($F, X \rightarrow Y$): /* Test if $X \rightarrow Y \in F^+$ */
return $Y \subseteq \text{CLOSURE}(F, X)$  /* P-rule */

- Example: $A \rightarrow C \in \{A \rightarrow B, B \rightarrow C\}^+$?
Minimal Cover

... to minimize a set of FDs

\[
\text{forall } \text{FD } X \rightarrow Y \in F \quad /* \text{Left reduction} */ \\
\text{forall } A \in X \quad /* A \text{ superflous?} */ \\
\text{if } Y \subseteq \text{CLOSURE}(F, X - \{A\}) \\
\text{then } \text{replace } X \rightarrow Y \text{ with } (X - A) \rightarrow Y \text{ in } F
\]

\[
\text{forall } \text{remaining FD } X \rightarrow Y \in F \quad /* \text{Right reduction} */ \\
\text{forall } B \in Y \quad /* B \text{ superflous?} */ \\
\text{if } B \subseteq \text{CLOSURE}(F - \{X \rightarrow Y\} \cup \{X \rightarrow (Y - B)\}, X) \\
\text{then } \text{replace } X \rightarrow Y \text{ with } X \rightarrow (Y - B)
\]

Eliminate FDs of the form  \(X \rightarrow \emptyset\)

Combine FDs of the form  \(X \rightarrow Y_1, X \rightarrow Y_2, \ldots\) into  \(X \rightarrow Y_1Y_2\ldots\)
Normal Forms . . .

- . . . determine properties of relation schemata
- . . . forbid certain combinations of functional dependencies in relations
- . . . should prevent redundancies and anomalies
First Normal Form

- Allows only *atomic* attributes in relation schemas, i.e., only elements of standard datatypes, such as `integer` or `string`, are allowed as attribute values, but not `array` or `set`.

- Not in 1NF:

<table>
<thead>
<tr>
<th>Vineyard</th>
<th>District</th>
<th>Region</th>
<th>WName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ch. La Rose Creek</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
<td>La Rose Grand Cru</td>
</tr>
<tr>
<td>Helena</td>
<td>Barossa Valley</td>
<td>South Australia</td>
<td>Creek Shiraz, Pinot Noir</td>
</tr>
<tr>
<td>Müller</td>
<td>Napa Valley</td>
<td>California</td>
<td>Zinfandel, Pinot Noir</td>
</tr>
<tr>
<td>Bighorn</td>
<td>Rheingau</td>
<td>Hessen</td>
<td>Riesling Reserve</td>
</tr>
<tr>
<td></td>
<td>Napa Valley</td>
<td>California</td>
<td>Chardonnay</td>
</tr>
</tbody>
</table>
First Normal Form /2

In first normal form:

<table>
<thead>
<tr>
<th>Vineyard</th>
<th>District</th>
<th>Region</th>
<th>WName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ch. La Rose</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
<td>La Rose Grand Cru</td>
</tr>
<tr>
<td>Creek Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
<td>Creek Shiraz Pinot Noir</td>
</tr>
<tr>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
<td>Creek Shiraz Pinot Noir</td>
</tr>
<tr>
<td>Helena Helena</td>
<td>Napa Valley</td>
<td>California</td>
<td>Zinfandel Pinot Noir</td>
</tr>
<tr>
<td>Bighorn Müller</td>
<td>Rheingau</td>
<td>Hessen</td>
<td>Riesling Reserve Chardonnay</td>
</tr>
</tbody>
</table>
## Second Normal Form

- **Partial dependency**: An attribute functionally depends on only part of the key

<table>
<thead>
<tr>
<th>Name</th>
<th>Vineyard</th>
<th>Color</th>
<th>District</th>
<th>Region</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>La Rose Grand Cru</td>
<td>Ch. La Rose</td>
<td>Rot</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
<td>39.00</td>
</tr>
<tr>
<td>Creek Shiraz</td>
<td>Creek</td>
<td>Rot</td>
<td>Barossa Valley</td>
<td>South Australia</td>
<td>7.99</td>
</tr>
<tr>
<td>Pinot Noir</td>
<td>Creek</td>
<td>Rot</td>
<td>Barossa Valley</td>
<td>South Australia</td>
<td>10.99</td>
</tr>
<tr>
<td>Zinfandel</td>
<td>Helena</td>
<td>Rot</td>
<td>Napa Valley</td>
<td>California</td>
<td>5.99</td>
</tr>
<tr>
<td>Pinot Noir</td>
<td>Helena</td>
<td>Rot</td>
<td>Napa Valley</td>
<td>California</td>
<td>19.99</td>
</tr>
<tr>
<td>Riesling Reserve</td>
<td>Müller</td>
<td>Weiß</td>
<td>Rheingau</td>
<td>Hessen</td>
<td>14.99</td>
</tr>
<tr>
<td>Chardonnay</td>
<td>Bighorn</td>
<td>Weiß</td>
<td>Napa Valley</td>
<td>California</td>
<td>9.90</td>
</tr>
</tbody>
</table>

- $f_1$: Name, Vineyard $\rightarrow$ Price
- $f_2$: Name $\rightarrow$ Color
- $f_3$: Vineyard $\rightarrow$ District, Region
- $f_4$: District $\rightarrow$ Region

- Second normal form eliminates such partial dependencies for non-key attributes
Elimination of Partial Dependencies

Key K

Part of Key X

dependent Attribute A

Sattler / Saake

Database Systems

Last Edited: October 2014 5–22
Second Normal Form /2

- Example relation in 2NF
  - \( R_1(\text{Name}, \text{Vineyard}, \text{Price}) \)
  - \( R_2(\text{Name}, \text{Color}) \)
  - \( R_3(\text{Vineyard}, \text{District}, \text{Region}) \)
Second Normal Form /3

- Note: Partially dependent attribute is only problematic if it is not a prime attribute.
- 2NF formally: Extended relation schema $\mathcal{R} = (R, \mathcal{K})$, FD set $F$ over $R$.

- $Y$ partially depends on $X$ w.r.t. $F$ if the FD $X \rightarrow Y$ is not left-reduced.
- $Y$ fully depends on $X$ if the FD $X \rightarrow Y$ is left-reduced.
- $\mathcal{R}$ is in 2NF if $\mathcal{R}$ is in 1NF and every non-prime attribute of $R$ fully depends on every key of $\mathcal{R}$.
Third Normal Form

- Eliminates transitive dependencies (in addition to the other kinds of dependencies)
- For instance, Vineyard $\rightarrow$ District and District $\rightarrow$ Region in relation on Slide 5-21
- Note: 3NF only considers non-key attributes as endpoints of transitive dependencies
Elimination of Transitive Dependencies

Key K

Set of Attributes X

dependent Attribute A
Third Normal Form /2

- Transitive dependency in R3, i.e., R3 violates 3NF
- Example relation in 3NF
  
  R3_1(Vineyard, District)
  R3_2(District, Region)
Third Normal Form: Formally

- Relation schema $R$, $X \subseteq R$ and $F$ is an FD set over $R$

  - $A \in R$ is called **transitively dependent** on $X$ w.r.t. $F$ if and only if there is a $Y \subseteq R$ for which it holds that $X \rightarrow Y$, $Y \not\rightarrow X$, $Y \rightarrow A$, $A \notin XY$

  - Extended relation schema $\mathcal{R} = (R, \mathcal{K})$ is in 3NF w.r.t. $F$ if and only if

    \[
    \forall A \in R : \quad A \text{ is non-prime attribute in } R \\
    \quad \land \ A \text{ transitively dependent on a } K \in \mathcal{K} \text{ w.r.t. } F.
    \]
Boyce-Codd Normal Form

- Stronger version of 3NF: Elimination of transitive dependencies also between prime attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Vineyard</th>
<th>Händler</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>La Rose Grand Cru</td>
<td>Château La Rose</td>
<td>Weinkontor</td>
<td>39.90</td>
</tr>
<tr>
<td>Creek Shiraz</td>
<td>Creek</td>
<td>Wein.de</td>
<td>7.99</td>
</tr>
<tr>
<td>Pinot Noir</td>
<td>Creek</td>
<td>Wein.de</td>
<td>10.99</td>
</tr>
<tr>
<td>Zinfandel</td>
<td>Helena</td>
<td>GreatWines.com</td>
<td>5.99</td>
</tr>
<tr>
<td>Pinot Noir</td>
<td>Helena</td>
<td>GreatWines.com</td>
<td>19.99</td>
</tr>
<tr>
<td>Riesling Reserve</td>
<td>Müller</td>
<td>Weinkeller</td>
<td>19.99</td>
</tr>
<tr>
<td>Chardonnay</td>
<td>Bighorn</td>
<td>Wein-Dealer</td>
<td>9.90</td>
</tr>
</tbody>
</table>

- FDs:

  \[
  \text{Name, Vineyard} \rightarrow \text{Price} \\
  \text{Vineyard} \rightarrow \text{Dealer} \\
  \text{Dealer} \rightarrow \text{Vineyard}
  \]

- Candidate keys: \{ Name, Vineyard \} und \{ Name, Dealer \}

- Example relation meets 3NF but not BCNF
Boyce-Codd-Normalform /2

• Extended relation schema \( \mathcal{R} = (R, \mathcal{K}) \), FD set \( F \)
• BCNF formally:

\[ \forall A \in R : A \text{ transitively depends on a } K \in \mathcal{K} \text{ w.r.t. } F. \]

• Schema in BCNF:
  
  WINES(Name, Vineyard, Price)
  
  WINE_TRADE(Vineyard, Dealer)

• However, BCNF may violate dependency preservation, therefore often stop at 3NF
Minimality

- Avoid global redundancies
- Meet other criteria (such as normal forms) with as few schemas as possible
- Example: Set of attributes $ABC$, set of FDs $\{A \rightarrow B, B \rightarrow C\}$
- Database schema in third normal form:

$$ S = \{(AB, \{A\}), (BC, \{B\})\} $$

$$ S' = \{(AB, \{A\}), (BC, \{B\}), (AC, \{A\})\} $$

Redundancies in $S'$
# Schema Properties

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Schema Property</th>
<th>Key Points</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1NF</td>
<td>Only atomic attributes</td>
</tr>
<tr>
<td></td>
<td>2NF</td>
<td>No non-prime attribute that partially depends on a key</td>
</tr>
<tr>
<td>S1</td>
<td>3NF</td>
<td>No non-prime attribute that transitively depends on a key</td>
</tr>
<tr>
<td></td>
<td>BCNF</td>
<td>No attribute that transitively depends on a key</td>
</tr>
<tr>
<td>S2</td>
<td>Minimality</td>
<td>Minimal number of relation schemas that satisfies the other properties</td>
</tr>
</tbody>
</table>
Transformation Properties

When decomposing a relation in multiple relations, care must be taken that . . .

1. . . only semantically sensible and consistent application data is presented (dependency preservation), and
2. . . all application data can be derived from the base relations (lossless-join decomposition)
Dependency Preservation

- **Dependency preservation**: A set of dependencies can be transformed into an equivalent second set of dependencies
  - More specifically: into the set of key dependencies because these can be validated efficiently by the database system
    - The set of dependencies shall be equivalent to the set of key constraints in the resulting database schema.
    - Equivalence ensures that, on a semantic level, the key dependencies express the exact same integrity constraints as the functional and other dependencies did before.
Dependency Preservation: Example

- Decomposition of the relation schema WINES (Slide 5-21) into 3NF:
  
  R1(Name, Vineyard, Price)
  R2(Name, Color)
  R3_1(Vineyard, District)
  R3_2(District, Region)

  with key dependencies

  Name, Vineyard → Price
  Name        → Color
  Vineyard    → District
  District    → Region

- Equivalent to FDs $f_1 \ldots f_4$ (Slide 5-21) $\rightsquigarrow$ dependency-preserving
Dependency Preservation: Example /2

- Zip code (a.k.a. postal code) structure of the Deutsche Post
  \( \text{ADDRESS}(\text{ZIP} \ (Z), \ \text{City} \ (C), \ \text{Street} \ (S), \ \text{Street Number} \ (N)) \)
  and functional dependencies \( F \):
    \[
    \text{CSN} \rightarrow Z, \ Z \rightarrow C
    \]

- Candidate keys: CSN and ZSN \( \leadsto \) 3NF

- Does not meet BCNF (because ZSN \( \rightarrow \) Z \( \rightarrow \) C): therefore decomposition of ADDRESS

- But: every decomposition would destroy CSN \( \rightarrow \) Z

- Set of resulting FDs is not equivalent to \( F \), the decomposition is therefore not dependency-preserving
Dependency Preservation: Formally

Locally extended database schema $S = \{(R_1, K_1), \ldots, (R_p, K_p)\};$

a set $F$ of local dependencies

$S$ fully characterizes $F$ (or: is dependency-preserving w.r.t. $F$) if and only if

$$F \equiv \{K \rightarrow R \mid (R, K) \in S, K \in \mathcal{K}\}$$
Lossless-Join Decomposition

- In order to satisfy the criteria of the normal forms, relation schemas sometimes have to be decomposed into smaller relation schemas.
- In order to restrict to “sensible” decomposition, require that the original relation can be recreated from the decomposed relations using a natural join.

\[ \rightarrow \text{lossless-join decomposition} \]
Lossless-Join Decomposition: Examples

- Decompose the relation schema $R = ABC$ into

  $$R_1 = AB \text{ und } R_2 = BC$$

- Decomposition is not join-lossless given the dependencies

  $$F = \{A \rightarrow B, C \rightarrow B\}$$

- In contrast, the decomposition is join-lossless given the dependencies

  $$F' = \{A \rightarrow B, B \rightarrow C\}$$
Lossless-Join Decomposition

- **Original relation:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Decomposition:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Join (join-lossless):**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Non-Join-Lossless Decomposition

- Original relation:
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

- Decomposition:
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

- Join (not join-lossless):
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Lossless-Join Decomposition: Formally

The decomposition of a set of attributes $X$ in $X_1, \ldots, X_p$ with $X = \bigcup_{i=1}^{p} X_i$ is called a lossless-join decomposition under a set of dependencies $F$ over $X$ if and only if

$$\forall r \in \text{SAT}_X(F) : \pi_{X_1}(r) \Join \cdots \Join \pi_{X_p}(r) = r$$

holds.

- Simple criterion for a join-lossless decomposition into two relation schemas: Decomposition of $X$ into $X_1$ and $X_2$ is join-lossless under $F$, if $X_1 \cap X_2 \rightarrow X_1 \in F^+$ or $X_1 \cap X_2 \rightarrow X_2 \in F^+$
## Transformation Properties

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Transformation Property</th>
<th>Key Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Dependency Preservation</td>
<td>All given dependencies are represented by keys</td>
</tr>
<tr>
<td>T2</td>
<td>Lossless-Join Decomposition</td>
<td>Original relations can be recreated by joining base relations</td>
</tr>
</tbody>
</table>
Design Methods: Goals

- Given: Universe $U$ and set of FDs $F$
- Locally extended database schema $S = \{(R_1, K_1), \ldots, (R_p, K_p)\}$
  - compute with
    - **T1**: $S$ fully characterizes $F$
    - **S1**: $S$ is in 3NF under $F$
    - **T2**: Decomposition of $U$ in $R_1, \ldots, R_p$ is dependency-preserving under $F$
    - **S2**: Minimality, i.e.,
      $\forall S' : S'$ satisfies **T1**, **S1**, **T2** and $|S'| < |S|$
Design Methods: Example

- Database schemas badly designed if only one of these four criteria is not fulfilled.

- Example: \( S = \{(AB, \{A\}), (BC, \{B\}), (AC, \{A\})\} \) fulfills \( T_1, S_1 \) and \( T_2 \) under \( F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\} \) in third relation \( AC \) tuple redundant or inconsistent.

- Correct: \( S' = \{(AB, \{A\}), (BC, \{B\})\} \)
Decomposition

- Given: Initial universal relation schema $\mathcal{R} = (\mathcal{U}, \mathcal{K}(F))$ with all attributes and a set of implied keys implied by FDs $F$ over $R$
  - Set of attributes $\mathcal{U}$ and set of FDs $F$
  - Find all $K \rightarrow \mathcal{U}$ with $K$ minimal, for which $K \rightarrow \mathcal{U} \in F^+ (\mathcal{K}(F))$
- Wanted: Decomposition into $D = \{\mathcal{R}_1, \mathcal{R}_2, \ldots \}$ of 3NF-relation schemas
Decomposition: Algorithm

**DECOMPOSE**(\(\mathcal{R}\))

Set \(D := \{\mathcal{R}\}\)

while \(\mathcal{R}' \in D\), does not meet 3NF

/* Find attribute \(A\) that is transitively dependent on \(K\) */

if Key \(K\) with \(K \rightarrow Y\), \(Y \notightarrow K\), \(Y \rightarrow A\), \(A \not\in KY\) then

/* Decompose relation schema \(R\) w.r.t. \(A\) */

\(R_1 := R - A\), \(R_2 := YA\)

\(\mathcal{R}_1 := (R_1, K)\), \(\mathcal{R}_2 := (R_2, K_2 = \{Y\})\)

\(D := (D - \mathcal{R}') \cup \{\mathcal{R}_1\} \cup \{\mathcal{R}_2\}\)

end if

end while

return \(D\)
Decomposition: Example

- Initial relation schema $R = ABC$
- Functional dependencies $F = \{A \rightarrow B, B \rightarrow C\}$
- Keys $K = A$
Decomposition: Example /2

- Initial relation schema $R$ with Name, Vineyard, Price, Color, District, Region
- Functional dependencies

$f_1$: Name, Vineyard $\rightarrow$ Price
$f_2$: Name, Vineyard $\rightarrow$ Vineyard
$f_3$: Name, Vineyard $\rightarrow$ Name
$f_4$: Name $\rightarrow$ Color
$f_5$: Vineyard $\rightarrow$ District, Region
$f_6$: District $\rightarrow$ Region
Decomposition: Assessment

- Advantages: 3NF, lossless-join decomposition
- Disadvantages: other criteria not fulfilled, depends on order, NP-hard (search for keys)
Synthesis Method

- Principle: Synthesis transforms original set of FDs $F$ into a resulting set of key dependencies $G$ such that $F \equiv G$
- “Dependency Preservation” built into the method
- 3NF and minimality also achieved, independent of order
- Computational complexity: quadratic
Comparison Decomposition — Synthesis

\[ \mathcal{U}, \text{FDs } F \]

\[ R, \mathcal{K} \]

\[ R_1, \mathcal{K}_1 \ldots R_m, \mathcal{K}_m \]

\[ R_1, \mathcal{K}_1 \ldots R_n, \mathcal{K}_n \]

\[ \text{FDs } F' \]

\[ \text{FDs } F'' \]

\[ \text{Decomposition} \quad \text{Synthesis} \]
Synthesis Method: Algorithm

- Given: Relation schema $R$ mit FDs $F$
- Wanted: Join-lossless and dependency-preserving decomposition into $R_1, \ldots R_n$ where all $R_i$ are in 3NF
- Algorithm:

```
SYNTHESIZE(F):
  \hat{F} := MINIMALCOVER(F) /* Determine minimal cover */
  Compute equivalence classes $C_i$ of FDs from $\hat{F}$ with equal or equivalent left sides, i.e., $C_i = \{X_i \rightarrow A_{i1}, X_i \rightarrow A_{i2}, \ldots \}$
  For each equivalence class $C_i$ create a schema of the form $R_{C_i} = \{X_i \cup \{A_{i1}\} \cup \{A_{i2}\} \cup \ldots \}$
  if none of the schemas $R_{C_i}$ contains a key from $R$
  then create additional relation schema $R_K$ with attributes from $R$, which form the key
  return $\{R_K, R_{C_1}, R_{C_2}, \ldots \}$
```
**Equivalence Classes**

- Class of FDs whose left sides are equal or equivalent
- Left sides are equivalent if they determine each other functionally
- Relation schema $R$ with $X_i, Y \subset R$, set of FDs $X_i \rightarrow X_j$ and $X_i \rightarrow Y$ with $1 \leq i, j \leq n$ can be expressed as

\[(X_1, X_2, \ldots, X_n) \rightarrow Y\]
Equivalence Classes: Example

- Set of FDs
  \[ F = \{A \rightarrow B, AB \rightarrow C, A \rightarrow C, B \rightarrow A, C \rightarrow E\} \]

- Minimal cover
  \[ \hat{F} = \{A \rightarrow B, B \rightarrow C, B \rightarrow A, C \rightarrow E\} \]

- Aggregation into equivalence classes
  \[ C_1 = \{A \rightarrow B, B \rightarrow C, B \rightarrow A\} \]
  \[ C_2 = \{C \rightarrow E\} \]

- Result of synthesis
  \( (ABC, \{\{A\}, \{B\}\}), (CE, \{C\}) \)
Achieving a Lossless-Join Decomposition

Achieve a lossless-join decomposition by a simple “trick”:

- Extend the original set of FDs \( F \) with \( U \rightarrow \delta \), where \( \delta \) is a dummy attribute
- \( \delta \) is removed after synthesis

Example: \{\( A \rightarrow B \), \( C \rightarrow E \)\}

- Result of synthesis \((AB, \{A\}), (CE, \{C\})\) is not lossless, because the universal key is not part of any schema
- Dummy-FD \( ABCE \rightarrow \delta \); reduced to \( AC \rightarrow \delta \)
- Yields third relation schema

\[(AC, \{AC\})\]
Synthesis: Example

Relation schema and set of FDs from Slide 5-49

Steps

1. Minimal cover: removal of $f_2, f_3$ as well as Region in $f_5$
2. Equivalence classes:

   - $C_1 = \{\text{Name}, \text{Vineyard} \rightarrow \text{Price}\}$
   - $C_2 = \{\text{Name} \rightarrow \text{Color}\}$
   - $C_3 = \{\text{Vineyard} \rightarrow \text{District}\}$
   - $C_4 = \{\text{District} \rightarrow \text{Region}\}$

3. Derivation of relation schemas
Summary

- Functional dependencies
- Normal forms (1NF – 3NF, BCNF)
- Dependency preservation and lossless-join decomposition
- Design methods
Control Questions

- What is the goal of normalizing relational schemas?

Which properties of relational schemas do the normal forms take into account?

What is the difference between 3NF and BCNF?

What does it mean for a decomposition to be dependency-preserving?

What is a lossless-join decomposition?
Control Questions

- What is the goal of normalizing relational schemas?
- Which properties of relational schemas do the normal forms take into account?
Control Questions

- What is the goal of normalizing relational schemas?
- Which properties of relational schemas do the normal forms take into account?
- What is the difference between 3NF and BCNF?
Control Questions

- What is the goal of normalizing relational schemas?
- Which properties of relational schemas do the normal forms take into account?
- What is the difference between 3NF and BCNF?
- What does it mean for a decomposition to be dependency-preserving?
- What is a lossless-join decomposition?