Part IV

Database Design
Database Design

1. Phases of Database Design
Database Design

1 Phases of Database Design

2 Further Steps During Design
Database Design

1. Phases of Database Design

2. Further Steps During Design

3. Capacity-preserving Transformations
Database Design

1. Phases of Database Design

2. Further Steps During Design

3. Capacity-preserving Transformations

4. ER-to-RM Transformation
Educational Objective for Today . . .

- Goals and steps of the database design process
Educational Objective for Today . . .

- Goals and steps of the database design process
- Rules to transform ER schemata into relational schemata
Goal of Design

- Data management for multiple application systems, for multiple years
- Therefore: special importance
- Design requirements
  - For every application, it should be possible to derive application data from data in the database — efficiently
  - Only store “sensible” (actually needed) data
  - Avoid redundancies
Phase Model

- Requirement Analysis
- Conceptional Design
- Distribution Design
- Logical Design
- Data Definition
- Physical Design
- Implementation & Maintenance
Requirements Analysis

- **Approach:** Collecting information needs from all specialist divisions

- **Result:**
  - Informal description (text, tabular lists, forms, etc.) of the problem domain
  - Separation of the information about data (data analysis) from the information about functions (functional analysis)

- **“Classical” DB design:**
  - Only data analysis and following steps

- **Functional design:**
  - See methods of software engineering
Conceptual Design

- First formal description of the problem domain
- **Language means**: semantical data model
- **Process**:
  - Modeling of views, e.g., for different specialist divisions
  - Analysis of existing views with respect to conflicts
  - Integration of views into a full schema
- **Result**: full conceptual schema, e.g., ER diagram
Phases of Conceptual Design

Concepcional Design

View Design

View Analysis

View Integration
Further Steps During Design

- ER modeling of different views of the complete information, e.g., for different specialist divisions of a company \( \sim \) conceptual design
  - Analysis and integration of views
  - Result: full conceptual schema
- Distribution design when using distributed storage
- Transformation to concrete implementation model (e.g., relational model) \( \sim \) logical design
- Data definition, implementation and maintenance \( \sim \) physical design
View Integration

- Analysis of existing view with respect to conflicts
- Integration of views into a full schema
Integration Conflicts

- **Naming conflicts**: Homonyms / synonyms
  - Homonyms: bank (money / river); order (command / request for goods)
  - Synonyms: car, vehicle, automobile

- **Typing conflicts**: different structures for the same element

- **Domain mismatch**: different domains for an element

- **Identifier conflicts**: e.g., different keys for the same element

- **Structural conflicts**: same fact expressed in different ways
Distribution Design

- If data should be distributed to several machines, a way of distributed storage must be determined.
- E.g., for a relation
  CUSTOMER (CNo, Name, Address, Zipcode, Account)
  - **Horizontal** distribution:
    CUSTOMER_1 (CNo, Name, Address, Zipcode, Account)
    where Zipcode < 50000
    CUSTOMER_2 (CNo, Name, Address, Zipcode, Account)
    where Zipcode >= 50000
  - **Vertical** distribution (connection via attribute CNo):
    CUSTOMER_Adr (CNo, Name, Address, Zipcode)
    CUSTOMER_Account (CNo, Account)
Logical Design

**Language means:** Data model of the chosen “implementation” DBMS, e.g., relational model

**Process:**

1. (Automatical) transformation of the conceptual schema, e.g., ER → relational model
2. Improvement of the relational schema based on quality criteria (normalization, see Chapter 6):
   Design goals: avoid redundancies, ...

**Result:** logical schema, e.g., collection of relation schemata
Data Definition

- Translation of logical schema into a concrete schema
- **Language means:** DDL and DML of DBMS (e.g., Oracle, DB2, SQL Server)
  - Database declaration in the DDL of the DBMS
  - Realization of integrity constraints
  - Definition of views
Physical Design

- Supplement physical design with support for efficient access, e.g., by defining indexes

- Index
  - Access path: data structure for additional, key-based access to tuples (⟨key attribute value, tuple address⟩)
  - Usually implemented as a B*-tree

- Language means: *storage structure (definition) language* SSL
Indexes in SQL

```
create [ unique ] index indexname
    on relname ( attrname [ asc | desc ],
    attrname [ asc | desc ],
    ...
 )
```

Example

```
create index WineIdx on WINES (Name)
```
Necessity of Access Paths

- Example: Table with 100 GB of data, hard disk transfer rate of ca. 50 MB/s
- Operation: Search for a tuple (selection)
- Implementation: Sequential search
- Cost: \(\frac{102.400}{50} = 2.048\) sec. \(\approx\) 34 min.
Implementation and Maintenance

- Phases of ...
  - Maintenance,
  - Further optimization of the physical layer,
  - Adaptation to new requirements or operating system platforms,
  - Porting to new database management systems
  - etc.
Transformation of the Conceptual Schema

- Translation to logical schema
  - Example: ER → RM
  - Correct?
  - Quality of transformation?

- Preservation of information capacity
  - Is it possible, after the transformation, to store exactly the same data as before?
  - ... or more?
  - ... or less?
Capacity-increasing Transformation

- Transformation into

\[ R = \{\text{LicenseNo}, \text{Vineyard}\} \]

with exactly one key

\[ K = \{\{\text{LicenseNo}\}\} \]

- Possible invalid relation:

<table>
<thead>
<tr>
<th>Has LicenseNo</th>
<th>Vineyard</th>
</tr>
</thead>
<tbody>
<tr>
<td>007</td>
<td>Helena</td>
</tr>
<tr>
<td>42</td>
<td>Helena</td>
</tr>
</tbody>
</table>
Capacity-preserving Transformation

- Correct instantiation

<table>
<thead>
<tr>
<th>Has</th>
<th>LicenseNo</th>
<th>Vineyard</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>007</td>
<td>Helena Müller</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td></td>
</tr>
</tbody>
</table>

- Correct set of keys

\[ K = \{\{\text{LicenseNo}\}, \{\text{Vineyard}\}\} \]
Capacity-decreasing Transformation

- Relation schema with one key \( \{ \text{WName} \} \)
- Instantiation that is no longer valid:

<table>
<thead>
<tr>
<th>ConsistsOf</th>
<th>WName</th>
<th>GrapeName</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Zinfandel Red Blossom</td>
<td>Zinfandel</td>
</tr>
<tr>
<td></td>
<td>Bordeaux Blanc</td>
<td>Cabernet Sauvignon</td>
</tr>
<tr>
<td></td>
<td>Bordeaux Blanc</td>
<td>Muscadelle</td>
</tr>
</tbody>
</table>

- Capacity-preserving when using the keys of both entity types as the new key in the relation schema

\[ K = \{ \{ \text{WName}, \text{GrapeName} \} \} \]
Example Transformation ER-RM: Input

Database Concepts

Last Edited: May 2017
Example Transformation ER-RM: Result

1. GRAPE = \{\text{Color, GrapeName}\} with \( K_{\text{GRAPE}} = \{\{\text{GrapeName}\}\} \)
2. ConsistsOf = \{\text{GrapeName, WName, Amount}\} with
   \( K_{\text{ConsistsOf}} = \{\{\text{GrapeName, WName}\}\} \)
3. WINE = \{\text{Color, WName, Vintage, Res.Sugar}\} with
   \( K_{\text{WINE}} = \{\{\text{WName}\}\} \)
4. PRODUCE = \{\text{WName, Vineyard}\} with \( K_{\text{PRODUCE}} = \{\{\text{WName}\}\} \)
5. PRODUCER = \{\text{Vineyard, Address}\} with \( K_{\text{PRODUCER}} = \{\{\text{Vineyard}\}\} \)
ER Transformation into Relations

- **Entity types and relationship types**: both transformed into relation schemata
- **Attributes**: attributes of the relation schema, keys are adopted
- **Cardinalities** of the relationships: expressed in respective relation schemata by choice of keys
- In some cases: **merge** of the relation schemata of entity and relationship types
- Introduce foreign key constraints between the remaining relation schemata
Transformation of Relationship Types

- New relation schema with all attributes of the relationship type; additionally, adopt all primary keys of the participating entity types

- **Determining keys:**
  - **m:n relationship:** both primary keys together form the key in the new relation schema
  - **1:n relationship:** primary keys of the n-side (in the functional notation, this is the side without the arrowhead) form key in the new relation schema
  - **1:1 relationship:** both primary keys become a key in the new relation schema; the primary key is then chosen from these keys
n:m Relationships

Transformation

1. \( \text{GRAPE} = \{\text{Color}, \text{GrapeName}\} \) with \( K_{\text{GRAPE}} = \{\{\text{GrapeName}\}\} \)
2. \( \text{ConsistsOf} = \{\text{GrapeName}, \text{WName}, \text{Amount}\} \) with \( K_{\text{ConsistsOf}} = \{\{\text{GrapeName}, \text{WName}\}\} \)
3. \( \text{WINE} = \{\text{Color}, \text{WName}, \text{Vintage}, \text{Res. Sugar}\} \) with \( K_{\text{WINE}} = \{\{\text{WName}\}\} \)

- Attributes GrapeName and WName together are key
1:n Relationships

(Preliminary) transformation
- PRODUCER with the attributes Vineyard and Address, 
- AREA with the attributes Name and Region, and 
- LocatedIn with the attributes Vineyard and Name and the primary key of the $n$-side Vineyard as primary key of this schema.
Possible Merges

- **Optional relationships** ([0,1] or [0,n]) are not merged
- With cardinalities [1,1] or [1,n] (**mandatory relationships**), merge is possible:
  - 1:n relationship: the entity-relation schema of the n-side can be integrated into the relation schema of the relationship
  - 1:1 relationship: both entity-relation schemata can be integrated into the relation schema of the relationship
### 1:1 Relationships

- **Licence**
  - LicenceNo
  - Hectoliter

- **Producer**
  - Vineyard
  - Address

**Has**

- **(Preliminary) transformation**
  - **PRODUCER** with the attributes Vineyard and Address
  - **LICENSE** with the two attributes LicenseNo and Hectoliters
  - **Has** with the primary keys of both participating entity types each as key of this schema, that is LicenseNo and Vineyard
1:1 Relationships: Merge

- Transformation with merge
  - Merged relation:

<table>
<thead>
<tr>
<th>PRODUCER</th>
<th>Vineyard</th>
<th>Address</th>
<th>LicenseNo</th>
<th>Hectoliters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotkäppchen</td>
<td>Freiberg</td>
<td>42-007</td>
<td>10 000</td>
<td></td>
</tr>
<tr>
<td>Vineyard Müller</td>
<td>Dagstuhl</td>
<td>42-009</td>
<td>250</td>
<td></td>
</tr>
</tbody>
</table>

- Producers without license require null values:

<table>
<thead>
<tr>
<th>PRODUCER</th>
<th>Vineyard</th>
<th>Address</th>
<th>LicenseNo</th>
<th>Hectoliters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotkäppchen</td>
<td>Freiberg</td>
<td>42-007</td>
<td>10 000</td>
<td></td>
</tr>
<tr>
<td>Vineyard Müller</td>
<td>Dagstuhl</td>
<td>42-009</td>
<td>250</td>
<td></td>
</tr>
</tbody>
</table>

- Free Licenses lead to additional null values:

<table>
<thead>
<tr>
<th>PRODUCER</th>
<th>Vineyard</th>
<th>Address</th>
<th>LicenseNo</th>
<th>Hectoliters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotkäppchen</td>
<td>Freiberg</td>
<td>42-007</td>
<td>10 000</td>
<td></td>
</tr>
<tr>
<td>Vineyard Müller</td>
<td>Dagstuhl</td>
<td></td>
<td>42-003</td>
<td>100 000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dependent Entity Types

- **Transformation**
  1. \( \text{WINEVINTAGE} = \{\text{WName, Res. Sugar, Year}\} \) with \( K_{\text{WINEVINTAGE}} = \{\{\text{WName, Year}\}\} \)
  2. \( \text{WINE} = \{\text{Color, WName}\} \) with \( K_{\text{WINE}} = \{\{\text{WName}\}\} \)

- Attribute \text{WName} in \text{WINEVINTAGE} is foreign key to relation \text{WINE}
IS-A Relationship

Transformation

1. \( \text{WINE} = \{\text{Color}, \text{WName}\} \) with \( K_{\text{WINE}} = \{\{\text{WName}\}\} \)
2. \( \text{SPARKLING\_WINE} = \{\text{WName}, \text{Production}\} \) with 
   \( K_{\text{SPARKLING\_WINE}} = \{\{\text{WName}\}\} \)

- \text{WName} in \( \text{SPARKLING\_WINE} \) is foreign key with respect to relation \( \text{WINE} \)
Recursive Relationships

Transformation

1. $\text{AREA} = \{\text{Name, Region}\}$ with $K_{\text{AREA}} = \{\{\text{Name}\}\}$
2. $\text{ADJOINS} = \{\text{to, from}\}$ with $K_{\text{ADJOINS}} = \{\{\text{to, from}\}\}$
Recursive Functional Relationships

Transformation

1. \( \text{CRITIC} = \{ \text{Name, Organization, Mentorname} \} \) with 
   \( K_{\text{CRITIC}} = \{ \{ \text{Name} \} \} \)

   Mentorname is foreign key to attribute Name of relation CRITIC.
Every participating entity type is treated according to the rules stated above.

For relationship Recommends, the primary keys of the three participating entity types are included in the resulting relation schema.

Relationship has a generic type (k:m:n relationship): all primary keys together form the key.
N-ary Relationships: Result

1. RECOMMENDS = \{WName, DName, Name\} with 
   \(K_{RECOMMENDS} = \{\{WName, DName, Name\}\}\)

2. DISH = \{DName, Side_Dish\} with \(K_{DISH} = \{\{DName\}\}\)

3. WINE = \{Color, WName, Vintage, Res_Sugar\} with 
   \(K_{WINE} = \{\{WName\}\}\)

4. CRITIC = \{Name, Organization\} with \(K_{CRITIC} = \{\{Name\}\}\)

The three key attributes of RECOMMENDS are foreign keys to the respective source relations (CRITIC, WINE, DISH).
## Overview of Transformations

<table>
<thead>
<tr>
<th>ER Concept</th>
<th>Is Translated into Relational Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity type $E_i$</td>
<td>Relation schema $R_i$</td>
</tr>
<tr>
<td>Attributes of $E_i$</td>
<td>Attributes of $R_i$</td>
</tr>
<tr>
<td>Primary key $P_i$</td>
<td>Primary key $P_i$</td>
</tr>
<tr>
<td>Relationship type</td>
<td>Relation schema</td>
</tr>
<tr>
<td>Its attributes</td>
<td>Attributes: $P_1$, $P_2$</td>
</tr>
<tr>
<td>$1 : n$</td>
<td>Further attributes</td>
</tr>
<tr>
<td>$P_2$ becomes primary key of the relationship</td>
<td></td>
</tr>
<tr>
<td>$1 : 1$</td>
<td>$P_1$ and $P_2$ become key of the relationship</td>
</tr>
<tr>
<td>$m : n$</td>
<td>$P_1 \cup P_2$ becomes primary key of the relationship</td>
</tr>
<tr>
<td>IS-A relationship</td>
<td>$R_1$ gets an additional key $P_2$</td>
</tr>
</tbody>
</table>

$E_1$, $E_2$: Entity types participating in a relationship,

$P_1$, $P_2$: Their primary keys,

$1 : n$ relationship: $E_2$ is $n$-side,

IS-A relationship: $E_1$ is a special entity type
Summary

- Phases of database design
Summary

- Phases of database design
- Capacity-preserving transformations
Summary

- Phases of database design
- Capacity-preserving transformations
- Transformation ER → relational
Control Questions

- Which steps does the database design process comprise?
Control Questions

- Which steps does the database design process comprise?
- Which requirements do the transformations between each design step have to fulfill? Why?
Control Questions

- Which steps does the database design process comprise?
- Which requirements do the transformations between each design step have to fulfill? Why?
- How are concepts of the ER model translated into concepts of the relational model?
Control Questions

- Which steps does the database design process comprise?
- Which requirements do the transformations between each design step have to fulfill? Why?
- How are concepts of the ER model translated into concepts of the relational model?
- How are the different cardinalities of relationship types accounted for during transformation?