Mapping of the E/R Model to the Relational Model
Life cycle of a database

Data processing requirements

Requirement analysis

Information requirements

Conceptional database design

Logical database design

Model properties

DBMS properties

Physical database design

HW/OS requirements
Design documentation

logical structure „on paper“

1st step
Map ER types systematically to relations:
From semantic concepts to syntactic structures!
Mapping from ER model to relational model:
- in principle very easy: per type one relation (table)
- in detail and for extensions: quite difficult

Mapping of entity types: rather obvious
- type $\Rightarrow$ table name
- attribute $\Rightarrow$ column name

Key attributes are mapped to primary key columns.
for relationship types:
- Use the same basic idea: one relation per type.
- But: How are participating entity types represented?

Obvious solution as well: The participating entities are represented by means of the values of their primary key attributes (possibly after renaming).
In presence of special functionalities (1 : N, N : 1, 1 : 1 resp.), a separate relationship table is not necessary, as the relationship information can be embedded into the table of the entity type on the N-side:

```
city
  name
  inhabitants
  ...
```

```
state
  name
  area
  ...
```

```
city_in_state
  city
  name
  state
  since
```

```
city
  city.name
  inhabitants
  ...  state.name
  since
```

Foreign key
Relational representation of generalization hierarchies

How to realize inheritance and sub-/super type relationships relationally?

- relational model: does not know any inheritance!
- Inheritance thus has to be "simulated".

Relational representation of a super type $E_1$ is obvious:

$$
\begin{array}{ccc}
E_1 & A_1 & A_2 \\
\hline
A_1 & A_2 & \ldots \\
\end{array}
$$
Generalization hierarchies (2)

- **Obvious** relational realization of a subtype: subtype relation $E_2$ owns "native" and inherited attributes.

- **But:** Values of the inherited attributes of all $E_2$-instances have to be (redundantly) repeated in the $E_1$-relation in this case!

- **Reason:** Each $E_2$-instance is an $E_1$-instance, too!
Generalization hierarchies (3)

Avoiding duplication: Store only „native“ attributes (+ key for joining) in the subtype relation!

But in this case relation $E_2$ does no longer contain all attributes of $E_1$-entities!
way out: E2-population is completely realized by means of a view joining the inherited and the native attributes.

CREATE VIEW E2-global AS
(SELECT *
FROM E1 INNER JOIN E2-local
Generalization hierarchies (5)

third alternative (also free of redundancies and using a view):
Distribute values of the **inherited** attributes to **different** relations –
super types are reconstructed via views.

<table>
<thead>
<tr>
<th>E1-local</th>
<th>E1-global</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>A2</td>
</tr>
<tr>
<td>a21</td>
<td>a22</td>
</tr>
</tbody>
</table>

CREATE VIEW `E1-global` AS
(TABLE `E1-local`)
UNION
(SELECT `A1`, `A2`, . . .
FROM `E2`)

\[ \pi \]

\[ \text{"native" attributes} \]
Which of the three alternatives is „the best“?

1. Relations $E_1$ and $E_2$, no views:

   + short access time (without any join of tables)
   
   - high requirements for space (due to redundant storage)

2. Relations $E_1$ and $E_2$-local, view $E_2$-global (JOIN):

   + Only key attribute values are stored redundantly.
   
   - Access to $E_2$-attributes is slower (due to join).

3. Relations $E_2$ and $E_1$-local, view $E_1$-global (PROJECT-UNION):

   + No duplication of any attribute values.
   
   - Access to $E_1$-attributes is slower (due to projection and union).
Generalization and updates

What happens if an $E_2$-entity is deleted?

- relational variant 1 (inherited attributes duplicated): Deletion from both relations is necessary.
- relational variant 2 (inherited and native A. separated): Deletion from both relations is necessary.
- relational variant 3 ($E_2$-attributes only in one relation): no propagation of deletions required

⇒ In variants 1 and 2: referential integrity constraints with delete cascade is required.

- For insertions and modifications: Changes in several relations may be necessary, too (depending on the chosen strategy).

- Deletion of instances of the super type $E_1$: Cascading deletion if the resp. instance is an $E_2$-instance, too (again referential integrity).
Each relationship type induces FOREIGN KEY-constraints as well:

- with N : M-functionality:

  ![Diagram of N : M relationship]

  ```sql
  CREATE TABLE river_through_state
  ( River String REFERENCES river,
    State String REFERENCES state )
  ```

- with N : 1-functionality:

  ![Diagram of N : 1 relationship]

  ```sql
  CREATE TABLE city
  ( . . .,
    State String REFERENCES state )
  ```
Functionalities as constraints

- **Uniqueness** of the state in the city_in_state-relationship has been expressed indirectly using key constraints!

- **CHECK-constraints** can be employed in order to restrict the number of state instances in a more general way:

```
CREATE TABLE city
    ( . . . ,
    CHECK COUNT (SELECT State
        FROM city S
        WHERE S.Name = Name) <= 2
    )
```

An implicit universal quantifier ranging over each city row!
Step 1: Translation of Entities
- One to one mapping to relations

Step 2: Translation of Attributes
- Simple attributes
  - Direct mapping to attributes in the relational model
- Composite attributes (e.g.: addresses)
  - Composite attributes as independent attributes
  - One attribut that contains all the information
  - Using object-relational techniques

```
a(keyOfA, attrOfA, attrA1, attrA2, attrA3)
```

Model as a record with:
```plaintext
attrA1, attrA2, and attrA3
```

`A`

```plaintext
keyOfA
attrOfA
compositeOfA
derivedAttr
```

plus Attribut (, derivedAttr)
Step 2: Translation of Attributes (cont’d)

- Calculated attributes (e.g.: turnover = price*quantity)
  - Store the calculated column in relation and using a trigger for updates
  - Encapsulation of the relation using a view with an additional attribute
  - Using generated columns: GENERATED ALWAYS AS

- Multi-value attributes (e.g.: telephone number)
  - Additional relation with foreign key to the original relation

\[
c(keyOfC, attrOfC) \\
\text{multiC}(keyOfC, \text{multivaluedAttr}) \text{ foreign key (keyOfC) references } c(keyOfC)
\]
Step 3: Translation of 1:1 Relationships

- **Case 1:**
  - One relation with attributes from E₁, E₂ and Rel primary key is key for E₁ and E₂

- **Case 2:**
  - One relation with all attributes and primary key of E₁, NULL values for key of E₂ possible
  - Two relations for E₁ and E₂: E₁ has an additional foreign key which references the primary key of E₂, foreign key does not allow NULL

- **Case 3:**
  - Two relations for E₁ and E₂: E₁ has an additional foreign key which references the primary key of E₂, foreign key allows NULL
  - Three relations: relation Rel contains foreign keys which point to E₁ and E₂
Step 4: Translation of 1:N Relationships

- **Case 1:**
  - Two relations: relation $E_2$ additionally contains primary key of $E_1$ and the attributes of Rel, foreign key does not allow NULL

- **Case 2:**
  - Two relations: relation $E_2$ additionally contains primary key of $E_1$ and the attributes of Rel, foreign key allows NULL
  - Three relations: primary key of Rel is combined key from $E_1$ and $E_2$
Step 5: Translation of N:M Relationships

- **Three** relations, primary key of Rel is combined key from $E_1$ and $E_2$

Step 6: Translation of Relationships with more than two relations

- *(n+1)* relations $E_1, ..., E_n$ and Rel,
  Primary key of Rel is combined key from $E_1, ..., E_n$
Step 7: Translation of recursive Relationships

- **One** relation $E_1$ with foreign key to itself
- **Two** relations $E_1$ and $Rel$,
  $Rel$ consists of two primary key columns of $E_1$ with different attribute names

- Example
  
  $b(keyOfB, attrOfB)$
  
  $bb(childKeyOfB, parentKeyOfB)$
  
  foreign key (childKeyOfB) references $b(keyOfB)$
  
  foreign key (parentKeyOfB) references $b(keyOfB)$
Step 8: Translation of attributes at relationships

- Simple or composite attribute
  - See entities

- Multi-valued attributes
  - Additional relation R with multi-valued attributes of Rel and key attributes of E₁ and E₂

Step 9: Translation of inheritance relationships

- Using object-relational techniques (sub type and sub tables)

- Using relational methods → 3 different alternatives (next slide...
Translating Inheritance Relationships

Overview

Horizontal Partitioning

Vertical Partitioning

Universal relation
(typed partitioning)

Comparison

- Different behavior of sub and super class queries
- Universal relation
  - Typically used in practice, limitation of #columns
Translating Inheritance Relationships

*Horizontal Partitioning*

- Each object is exactly one tuple in one relation, i.e. same OID does not mean that is the same object

*Example*

- All tuples with general attributes

  ```sql
  (SELECT "Product", id, price, stock FROM products)
  UNION
  (SELECT "Book", id, price, stock FROM book)
  UNION
  (SELECT "CD", id, price, stock FROM cd)
  ```

- Reconstructing the universal relation

  ```sql
  (SELECT "Product", id, price, stock, NULL AS isbn,
   NULL AS title, NULL AS album, NULL AS artist FROM products)
  UNION
  (SELECT "Book", id, price, stock, isbn,
   title, NULL AS album, NULL AS artist FROM book)
  UNION
  (SELECT "CD", id, price, stock, NULL AS isbn,
   NULL AS title, album, artist FROM cd)
  ```
Translating Inheritance Relationships

**Vertical Partitioning**
- All attributes of an object can be determined only by applying joins (expensive operation in databases)

**Example**
- Whole table with all common attributes

```
SELECT id, price, stock FROM product
```

- Reconstructing the universal relation

```
SELECT id, price, stock, isbn, title, album, artist
FROM product
    LEFT OUTER JOIN book ON product.id = book.id
    LEFT OUTER JOIN cd ON product.id = cd.id
```
### General Example

<table>
<thead>
<tr>
<th>Type</th>
<th>Restriction $E_1$</th>
<th>Restriction $E_2$</th>
<th>Resulting Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1</td>
<td>(1,1)</td>
<td>(1,1)</td>
<td>$E_{12}(S_1, A_1, S_2 \text{ NOT NULL &amp; UNIQUE, } A_2, A_3)$</td>
</tr>
<tr>
<td></td>
<td>(0,1)</td>
<td>(1,1)</td>
<td>$E_1(S_1, A_1) \text{ und } E_2(S_2, A_2, S_1 \text{ NOT NULL &amp; UNIQUE, } A_3)$</td>
</tr>
<tr>
<td></td>
<td>(0,1)</td>
<td>(0,1)</td>
<td>$E_1(S_1, A_1, S_2 \text{ NULL</td>
</tr>
<tr>
<td>1:N</td>
<td>(0,N)</td>
<td>(0,1)</td>
<td>$E_1(S_1, A_1) \text{ und } E_2(S_2, A_2, S_1, A_3)$</td>
</tr>
<tr>
<td></td>
<td>(1,N)</td>
<td>(0,1)</td>
<td>$E_1(S_1, A_1, S_2 \text{ NOT NULL}) \text{ und } E_2(S_2, A_2, S_1, A_3)$</td>
</tr>
<tr>
<td></td>
<td>(0,N)</td>
<td>(1,1)</td>
<td>$E_1(S_1, A_1) \text{ und } E_2(S_2, A_2, S_1 \text{ NOT NULL, } A_3)$</td>
</tr>
<tr>
<td>N:M</td>
<td>(0,M)</td>
<td>(0,N)</td>
<td>$E_1(S_1, A_1) \text{ und } E_2(S_2, A_2) \text{ und } R(S_1, S_2, A_3)$</td>
</tr>
<tr>
<td></td>
<td>(1,M)</td>
<td>(0,N)</td>
<td>$E_1(S_1, A_1, S_2 \text{ NOT NULL}) \text{ und } E_2(S_2, A_2) \text{ und } R(S_1, S_2, A_3)$</td>
</tr>
</tbody>
</table>
Summary

Database design is an important and complex process

- E/R modeling is a well-known and prevailed approach in industry

Important E/R components

- Entity  Entity Set  Entity Type
- Relationship  Relationship Set  Relationship Type

Characterization of relationship types

- 1:1, 1:m, n:m relations
- IS-A relation
- Strong versus weak relations

Mapping E/R to Relational Model

- Different translation rules for entities, attributes, relationships and inheritance