Chapter 10, Mapping Models to Relational Schema
Lecture Plan

• Last lecture:
  • Operations on the object model:
    • Optimizations to address performance requirements
    • Implementation of class model components:
      • Realization of associations

This lecture:
• Realizing entity objects based on selected storage strategy, in particular
• Mapping the object model to a relational database
  • Mapping class diagrams to tables
Mapping an Object Model to a Database

- UML object models can be mapped to relational databases:
  - Some degradation occurs because all UML constructs must be mapped to a single relational database construct - the table
- Mapping of classes, attributes and associations
  - Each class is mapped to a table
  - Each class attribute is mapped onto a column in the table
  - An instance of a class represents a row in the table
  - A many-to-many association is mapped into its own table
  - A one-to-many association is implemented as buried foreign key
- Methods are not mapped.
Mapping a Class to a Table

User
+firstName:String
+login:String
+email:String

User table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Primary and Foreign Keys

• Any set of attributes that could be used to uniquely identify any data record in a relational table is called a **candidate key**

• The actual candidate key that is used in the application to identify the records is called the **primary key**
  • The primary key of a table is a set of attributes whose values uniquely identify the data records in the table

• A **foreign key** is an attribute (or a set of attributes) that references the primary key of another table.
### Example for Primary and Foreign Keys

#### User table

<table>
<thead>
<tr>
<th>firstName</th>
<th>login</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>“alice”</td>
<td>“am384”</td>
<td>“<a href="mailto:am384@mail.org">am384@mail.org</a>”</td>
</tr>
<tr>
<td>“john”</td>
<td>“js289”</td>
<td>“<a href="mailto:john@mail.de">john@mail.de</a>”</td>
</tr>
<tr>
<td>“bob”</td>
<td>“bd”</td>
<td>“<a href="mailto:bobd@mail.ch">bobd@mail.ch</a>”</td>
</tr>
</tbody>
</table>

**Primary key**

**Candidate key**

**Candidate key**

#### League table

<table>
<thead>
<tr>
<th>name</th>
<th>login</th>
</tr>
</thead>
<tbody>
<tr>
<td>“tictactoeNovice”</td>
<td>“am384”</td>
</tr>
<tr>
<td>“tictactoeExpert”</td>
<td>“bd”</td>
</tr>
<tr>
<td>“chessNovice”</td>
<td>“js289”</td>
</tr>
</tbody>
</table>

**Foreign key**

*referencing User table*
Buried Association

- Associations with multiplicity “one” can be implemented using a foreign key

For one-to-many associations we add the foreign key to the table representing the class on the “many” end.

For all other associations we can select either class at the end of the association.
Another Example for Buried Association

Transaction Table

<table>
<thead>
<tr>
<th>transactionID</th>
<th>portfolioID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Portfolio Table

<table>
<thead>
<tr>
<th>portfolioID</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Foreign Key

Transaction

transactionID

Portfolio

portfolioID

...
Mapping Many-To-Many Associations

In this case we need a separate table for the association

- **City**
  - cityName

- **Airport**
  - airportCode
  - airportName

- **Serves**
  - * cityName
  - * airportCode

**City Table**
- cityName
  - Houston
  - Albany
  - Munich
  - Hamburg

**Airport Table**
- airportCode
  - IAH: Intercontinental
  - HOU: Hobby
  - ALB: Albany County
  - MUC: Munich Airport
  - HAM: Hamburg Airport

**Serves Table**
- cityName
  - Houston
  - Albany
  - Munich
  - Hamburg
- airportCode
  - IAH
  - HOU
  - ALB
  - MUC
  - HAM
Another Many-to-Many Association Mapping

We need the Tournament/Player association as a separate table

We need the Tournament/Player association as a separate table
Realizing Inheritance

• Relational databases do not support inheritance
• Two possibilities to map an inheritance association to a database schema
  With a separate table ("vertical mapping")
    • The attributes of the superclass and the subclasses are mapped to different tables
  • By duplicating columns ("horizontal mapping")
    • There is no table for the superclass
    • Each subclass is mapped to a table containing the attributes of the subclass and the attributes of the superclass
Realizing inheritance with a separate table (Vertical mapping)

User table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>...</th>
<th>role</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>zoe</td>
<td></td>
<td>LeagueOwner</td>
</tr>
<tr>
<td>79</td>
<td>john</td>
<td></td>
<td>Player</td>
</tr>
</tbody>
</table>

LeagueOwner table

<table>
<thead>
<tr>
<th>id</th>
<th>maxNumLeagues</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

Player table

<table>
<thead>
<tr>
<th>id</th>
<th>credits</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>126</td>
<td></td>
</tr>
</tbody>
</table>
Realizing inheritance by duplicating columns (Horizontal Mapping)

- **User**
  - name

- **LeagueOwner**
  - maxNumLeagues

- **Player**
  - credits

**LeagueOwner table**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>maxNumLeagues</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>zoe</td>
<td>12</td>
</tr>
</tbody>
</table>

**Player table**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>john</td>
<td>126</td>
</tr>
</tbody>
</table>
Comparison: Separate Tables vs Duplicated Columns

• The trade-off is between modifiability and response time
  • How likely is a change of the superclass?
  • What are the performance requirements for queries?

• Separate table mapping (Vertical mapping)
  ☑️We can add attributes to the superclass easily by adding a column to the superclass table
  ☒Searching for the attributes of an object requires a join operation.

• Duplicated columns (Horizontal Mapping)
  ☒Modifying the database schema is more complex and error-prone
  ☑️Individual objects are not fragmented across a number of tables, resulting in faster queries
Summary

• Four mapping concepts:
  • Model transformation improves the compliance of the object design model with a design goal
  • Forward engineering improves the consistency of the code with respect to the object design model
  • Refactoring improves code readability/modifiability
  • Reverse engineering discovers the design from the code.

• Model transformations and forward engineering techniques:
  • Optimizing the class model
  • Mapping associations to collections
  • Mapping contracts to exceptions
  • Mapping class model to storage schemas.
Backup and Example Slides