Object-Relational Modeling

Holger Pirk
Purpose of this Lecture

Introduce Object-Relational-Mapping

- Understand the problem it solves
- Understand the problems it does not solve
- Understand the problems it creates
- Learn about the implementations

Understand how ORM work

- Be able to manually perform the mapping
  - There are multiple ways of doing it – understand the tradeoffs
- Be able to create a class system for a relational schema
Normalization
Unnormalized Data - Everything goes

The Schema

OrderedBooks(Number, CustomerID, Customer, ShippingAddress, Titles, Authors)

<table>
<thead>
<tr>
<th>Number</th>
<th>CustID</th>
<th>Customer</th>
<th>ShippingAddress</th>
<th>Titles</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Holger</td>
<td>180 Queens Gate</td>
<td>Database Management Systems; A Game of Thrones</td>
<td>Ramakrishnan &amp; Gehrke; Martin</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Sam</td>
<td>32 Vassar Street</td>
<td>Database Management Systems</td>
<td>Ramakrishnan &amp; Gehrke</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Peter</td>
<td>180 Queens Gate</td>
<td>Distributed Systems</td>
<td>van Steen &amp; Tanenbaum</td>
</tr>
</tbody>
</table>
Fix it by breaking the Schema

The Schema

OrderedBooks(Number, Customer, ShippingAddress, Title 1, Title 2, Author 1, Author 2)

The Data

<table>
<thead>
<tr>
<th>Number</th>
<th>CustID</th>
<th>Customer</th>
<th>ShippingAddress</th>
<th>Title 1</th>
<th>Title2</th>
<th>Author 1</th>
<th>Author 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Holger</td>
<td>180 Queens Gate</td>
<td>Database Management Systems</td>
<td>A Game of Thrones</td>
<td>Ramakrishnan &amp; Gehrke</td>
<td>Martin</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Sam</td>
<td>32 Vassar Street</td>
<td>Database Management Systems</td>
<td></td>
<td>Ramakrishnan &amp; Gehrke</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Peter</td>
<td>180 Queens Gate</td>
<td>Distributed Systems</td>
<td></td>
<td>van Steen &amp; Tanenbaum</td>
<td></td>
</tr>
</tbody>
</table>
# First Normal Form

## The Schema

OrderedBooks(OrderNumber, Customer, ShippingAddress, BookNumber)

Books(Number, Title, Author)

## OrderedItem

<table>
<thead>
<tr>
<th>Number</th>
<th>CustomerID</th>
<th>Customer</th>
<th>ShippingAddress</th>
<th>BookNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Holger</td>
<td>180 Queens Gate</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Holger</td>
<td>180 Queens Gate</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Sam</td>
<td>32 Vassar Street</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Peter</td>
<td>180 Queens Gate</td>
<td>3</td>
</tr>
</tbody>
</table>

## Book

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database Management Systems</td>
<td>Ramakrishnan &amp; Gehrke</td>
</tr>
<tr>
<td>2</td>
<td>A Game of Thrones</td>
<td>Martin</td>
</tr>
<tr>
<td>3</td>
<td>Distributed Systems</td>
<td>van Steen &amp; Tanenbaum</td>
</tr>
</tbody>
</table>
### Second Normal Form

#### The Schema

- Orders (OrderNumber, Customer, ShippingAddress)
- OrderedBooks (OrderNumber, BookNumber)
- Books (Number, Title, Author)

#### Order

<table>
<thead>
<tr>
<th>Number</th>
<th>CustomerID</th>
<th>Customer</th>
<th>ShippingAddress</th>
<th>OrderNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Holger</td>
<td>180 Queens Gate</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Sam</td>
<td>32 Vassar Street</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Peter</td>
<td>180 Queens Gate</td>
<td>3</td>
</tr>
</tbody>
</table>

#### OrderedItem

<table>
<thead>
<tr>
<th>OrderNumber</th>
<th>BookNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

#### Book

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database Management Systems</td>
<td>Ramakrishnan &amp; Gehrke</td>
</tr>
<tr>
<td>2</td>
<td>A Game of Thrones</td>
<td>Martin</td>
</tr>
<tr>
<td>3</td>
<td>Distributed Systems</td>
<td>van Steen &amp; Tanenbaum</td>
</tr>
</tbody>
</table>
# Third Normal Form

## The Schema

Customers(ID, Customer, ShippingAddress)
Orders(OrderID, CustomerID)
OrderedBooks(OrderID, BookID)
Books(ID, Title, Author)

<table>
<thead>
<tr>
<th>ID</th>
<th>Customer</th>
<th>ShippingAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Holger</td>
<td>180 Queens Gate</td>
</tr>
<tr>
<td>2</td>
<td>Sam</td>
<td>32 Vassar Street</td>
</tr>
<tr>
<td>3</td>
<td>Peter</td>
<td>180 Queens Gate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>CustomerID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OrderId</th>
<th>BookID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database Management Systems</td>
<td>Ramakrishnan &amp; Gehrke</td>
</tr>
<tr>
<td>2</td>
<td>A Game of Thrones</td>
<td>Martin</td>
</tr>
<tr>
<td>3</td>
<td>Distributed Systems</td>
<td>van Steen &amp; Tanenbaum</td>
</tr>
</tbody>
</table>
Object-Relational Mapping
What do we want from a DBMS?

Persistence!!

- When do we want it?
  - All the time!
- Databases are persistent
- Processes are not
Object-Oriented Databases

Programs work with objects, not tuples

- **Granularity**
  - Reconstruction overhead (Data navigation)

- **Inheritance**

- **Identity vs. Equality**
  - Identity is an inherent property
  - Equality has to be defined

- **Associations**
Why OO DBMSs failed

A couple of factors
- No standardized interface (for a long time)
- No real story for OLAP
- Non-technical reasons: IBM and Oracle backed relations
- Coincidence

ORMs are a compromise
- Given the illusion of an OODBMS
- On top of an RDBMS
Some Nomenclature

**Wording**

- **ORM** Object-Relational Mapping
- **Relation** A set of tuples (i.e., the relational sense)
- **Class** A template for object creation (i.e., the object-oriented sense)
- **Association** A rule defining the relationship between the objects of two classes (i.e., the object-oriented sense)
This part of the class used to be about ER-Modelling

- Very related to ORM
- Popular in the 90s as a higher-level modeling paradigm
  - (Something management would understand)
- Rarely used today because it did not catch on in programming languages...
  - ... which is why we cover ORM instead
- ORMs are actually quite popular in practice
Functionality of an ORM

**Must have**
- Mapping
  - Abstraction of syntax/oddities
  - Integrity Constraints
- Querying
- Schema
- Inserts

**Nice to have**
- Non-Intrusive
- Version control
- Faithfulness: object-oriented semantics should be represented
Let's dive into it
Poll: Is anybody seeing this for the first time?
classes

Classes in UML

class Customer{
std::string name;
std::string shipping_address;
std::set<std::shared_ptr<Customer>> orderedBooks;
}
class Book {
std::string author;
std::string title;
}

Poll: Is anybody seeing this for the first time?
Classes

Classes in UML

class Customer{
    std::string name;
    std::string shipping_address;
    std::set<std::shared_ptr<Customer>> orderedBooks;
}
class Book {
    std::string author;
    std::string title;
}

- Poll: Is anybody seeing this for the first time?
- Question: Why are there no IDs here?
Develop an object-oriented model to describe the following domain

**Domain**

The payroll system for *BIG Inc* records the salaries, status, joining date, name, and payroll number for all of the corporation’s 30,000 employees. Each employee works for one division, and each division has an account number for paying its staff. We identify divisions by their name, and record the address where the division’s HQ is located.

For employees sent abroad by *BIG Inc*, we record the address, country and telephone number of the foreign tax office that will handle the employee. It is assumed that each country has one central tax office that we have to deal with. All other employees have their tax affairs dealt with by the Inland Revenue.
Develop an object-oriented model to describe the following domain

**Domain**

The payroll system for *BIG Inc* records the salaries, status, joining date, name, and payroll number for all of the corporation’s 30,000 employees. Each employee works for one division, and each division has an account number for paying its staff. We identify divisions by their name, and record the address where the division’s HQ is located.

For employees sent abroad by *BIG Inc*, we record the address, country and telephone number of the foreign tax office that will handle the employee. It is assumed that each country has one central tax office that we have to deal with. All other employees have their tax affairs dealt with by the Inland Revenue.
Mapping Classes

Mapping Rules
- Every class is mapped to a table
- Every scalar attribute is mapped to an attribute

Relations
Customer(Name, ShippingAddress)
Book(Title, Author)
Mapping Relations

Multiplicities

- One-to-One
- One to Many
- Many to Many
One-to-One

**In UML**

- Class `Book` with an association to `Person`
- `Person author` in `Book`
- `Person` has one instance

**In C++**

```cpp
class Person {
}
class Book {
    Person author;
}
```
One-to-One

Translation rules
- Both classes get mapped into one (i.e., the same) relation
- Attributes of the relation is the concatenation of both
  - Conflicts can be resolved using, e.g., prefixing with class name

A note on One-to-One
- This is a very rare case
  - Two entities that are intricately linked but still semantically different
- Many practical frameworks do not treat one-to-one properly
  - Often modeled the same as One-to-Many
One-to-Many

In UML

![UML Diagram]

In C++

class Book {
    Person* author; // <- This is optional
    // Don't play with raw pointers, children!
}

class Person {
    set<Book> authoredBooks;
}

*Question: could a vector be used instead of a set?*
One-to-Many

Translation rules

- Each class gets mapped into one (i.e., its own) relation
- The many-side receives a foreign-key attribute referencing the one-side

Example

```sql
create table Person(
  id int;
  ...;
  primary key(id);
)

create table Book(
  Person_id int not null;
  foreign key (Person_id) references Person(id);
)
```
Many-to-Many

In UML

In C++

class Book {
    set<Person*> authors; // <- Both are needed
}
class Person {
    set<Book*> authoredBooks; // <- Both are needed
}
Many-to-Many

Translation rules

- Each class gets mapped into one (i.e., its own) relation
- A third table is created containing the mapping
  - It contains two columns, each of which is a foreign key

Example

```sql
create table Person(
    id int; primary key(id);
);
create table Book(
    id int; primary key(id);
);

create table Book_to_Person(
    Person_id int not null; foreign key (Person_id) references Person(id);
    Book_id int not null; foreign key (Book_id) references Book (id);
);
```
Mapping Inheritance
Here comes the mismatch

An Example

Questions:

- How do we map this to a relation?
- Why would this be a problem?
Resolving inheritance

Let’s give it a shot

- Every class maps to a relation
  - Person, Customer, Author, Book
- The associations are all many-to-many
  - How do we map those?
- What about the inheritance
  - Let's say we create foreign keys
  - Is the schema in 3NF?
  - Is the schema faithful? Are the object-oriented semantics represented?
Let’s give it another shot

- Every concrete class maps to a relation
  - `CustomerWithPerson`, `AuthorWithPerson`, `Book`
- The associations are all many-to-many
  - How do we map those?
- What about the inheritance
  - Is the schema in 3NF?
  - Is the schema faithful? Are the object-oriented semantics represented?
Resolving inheritance 2

Let’s complicate the example

Question

- Is there a problem here?
Let’s give it a last shot

- Every class hierarchy maps to a relation
  - CustomerWithCustomerWithPerson, Book, TwitterAccount
- The associations are all many-to-many
  - How do we map those?
- What about the inheritance
  - Is the schema in 3NF?
  - Is the schema faithful? Are the object-oriented semantics represented?
## Conclusion

### Three methods
- table per class (abstract or concrete)
- table per concrete class
- Single table per hierarchy

### None are perfect
- Per class tables break object-oriented semantics
- Per concrete class table may break foreign-key constraints
- Single table per hierarchy violates normal form (produces sparse tables)
Continuing the previous example

- Map your previously defined class hierarchy to a relational schema
Reverse-engineering the OO model

Relatively simple rules

- Every relation is mapped to a class
  - Exception: relations with only two foreign-key columns
- Every non-foreign key attribute is mapped to an attribute
- Every PK/FK constraint gets mapped to
  - a collection on the PK side
  - a reference-pointer on the FK side

Question:

- Is ORM mapping/reverse-mapping a bijective process?
- Let’s find out!
Worksheet: Reverse-ORM

Continuing the previous example

- Reverse-Map your previously defined class hierarchy to a relational table
Null-Semantics in relational DBMSs

<table>
<thead>
<tr>
<th>Quiz</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unknown</strong></td>
<td>We don’t know the value of this attribute, may not even have one</td>
</tr>
<tr>
<td><strong>Missing</strong></td>
<td>The value exists but has not been entered into the system</td>
</tr>
<tr>
<td><strong>Uncertain</strong></td>
<td>The value exists but I have conflicting information about what it is</td>
</tr>
<tr>
<td><strong>Invalid</strong></td>
<td>The value that was entered in the system does not conform to the rules</td>
</tr>
<tr>
<td><strong>Not an option</strong></td>
<td>This column is here for technical reasons but is not to be set</td>
</tr>
</tbody>
</table>
Implementations
Example

@Entity
class User {
  @Id
  int id;
  String Name = null;
  String Login = null;
  String Password = null;
}
#include <memory>
#include <set>
#include <string>
#include <vector>

// never do this:
using namespace std;

class Author;

#pragma db object
class Book {  
public:
#pragma db id   
    int id;
    string title;
    Author* author;
};

#pragma db object
class Person {  
public:
#pragma db id
    int id;
    string name;
    set<shared_ptr<Book>> orderedBooks;
};

#pragma db object
class Author {  
public:
#pragma db id
    int id;
    string name;
#pragma db inverse(author)
    set<Book*> authoredBooks;
};
C++: ODB

Querying

```cpp
transaction ct2(db.begin());
auto folks = db.query<Person>(odb::query<Person>::name == "holger");
for(auto& person : folks)
    for(auto& book : person.orderedBooks)
        cout << book->title;
ct2.commit();
```
The end