Object-Relational Mapping
Object-Relational Mapping

- Software Architecture
- ORM Problems
- ORM Solutions
- Demo
Software Architecture

- **Separation of Concerns**
  - A design principle that comprises the process of separating a computer program into distinct features that overlap in functionality as little as possible

- **Architecture style**
  - A set of principals that shape an application
  - Improves SoC and promotes reusable solutions

- **Examples**
  - Service-Oriented Architecture
  - Component-Based Architecture
  - 3-Tier Architecture
Presentation Layer

Logic Layer

Data Access Layer

Database
Data Access Layer

• Provides simplified access to data stored in persistent storage, such as a relational database.

• Might return an object to the business layer when the business layer makes a request for some data.

• This layer is in charge of communicating to the database.
  - Should encapsulate the data storage type and data access method

• This is the layer provided by Object-Relational Mapping
Data Access Layer using Object-Relational Mapping

Table: Course

<table>
<thead>
<tr>
<th>cnum</th>
<th>title</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CS4123</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>CS3321</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>CS3333</td>
<td>3</td>
</tr>
</tbody>
</table>

```cpp
class Course {
    private:
        string cnum;
        int credits;
        string title;
    public:
        // getters
        // & setters
};
```
Object-Relational Mapping

- Two Paradigms
  - Object-Oriented Programming
  - Relational Database Model
- The don't always play nice together
  - Problem of Granularity
  - Problem of Inheritance
  - Problem of Identity
  - Problems of Association
  - Problem of Data Navigation
Object-Relational Mapping

- Problem of Granularity
  - Fine grain classes can be a part of coarse grain classes (class composition). Relations do not have this feature
  - The granularity problem comes when the number of classes mapping to number of tables in the database do not match.
Problem of Granularity

For example, let's say we have the User class which has an Address object.

```java
public class User{
    private String Name;
    private Address address;

    //Setters and getters
}
```

```java
public class Address{
    private String city;
    private String country;

    //Setters and getters
}
```
Problem of Granularity

Suppose the table structure for User is

Table USER:

NAME, CITY, COUNTRY

One way to capture this relation using the User and Address classes.

There is one table but the data is sitting in two objects. The same problem can come the other way round also where you have two tables and one class containing all the data points. ORM has to take care of this mismatch.
• **Problem of Inheritance**
  • Classes can inherit from other classes, this can be difficult to represent in a database
  • How should class inheritance be represented with relations?
  • There is also a polymorphism problem. A reference of User type can refer to an object of Student or Teacher. The ORM has to somehow differentiate between data that is belonging to User or Student or Teacher.
Object-Relational Mapping

• **Problem of Identity**
  • In the relational model, the primary key determines identity.
  • Default equality of objects is usually reference
  • Must implement an equals method
  • Situation gets more complicated when one table is represented by more than one class and in multithreaded applications.
Object-Relational Mapping

• Problem of Association
  • Object association is directional, one object contains another
  • There is no notion of directionality with Relational associations (foreign keys)
    • There are just tables and keys
  • Many-to-many relationships can be represented as 3 tables, how should this be represented with classes
Object-Relational Mapping

- Problem of Data Navigation
  - Accessing data in an object is different from accessing data in a table
  - With objects we use getters
    - e.g. order.getLineItem().getProduct();
  - With relations we use joins
  - How do we map between these?
Object-Relational Mapping

- Recap
  - Problem of Granularity
  - Problem of Inheritance
  - Problem of Identity
  - Problems of Association
  - Problem of Data Navigation
Object-Relational Mapping

• Solutions
  • Write a Data Access Layer from scratch
  • Use an ORM software
    • Converts data stored relationally into objects
    • There are many software tools that can do this automatically
      – Relational Database => Classes
      – Classes=> Relational Database
Object-Relational Mapping Software

- **Java**
  - Java Persistence API (JPA)
  - Hibernate, open source ORM framework, widely used

- **PHP**
  - CakePHP
  - Zend Framework

- **C++**
  - ODB
  - QxORM

- **.NET**
  - ADO.NET Entity Framework, included in .NET Framework 3.5 SP1 and above
  - NHibernate, open source
Object-Relational Mapping Pros

- Provide a way to persist objects using a relational database
- reduces the amount of code that needs to be written
- Allow the programmer to query data in a programming language and not with SQL
  - Query with code completion!
Object-Relational Mapping Cons

- Can be slow
- Often lead to learning ORM specific features instead of just learning SQL
- Some complex queries cannot be done without SQL
- Can produce poorly designed databases
ADO.NET Entity Framework