Object-relational mapping: An introduction
Object-relational mapping (ORM): what do we mean?

Context:

• Development of data-driven software systems
• DATABASE LAYER vs APPLICATION LAYER

Traditionally, two different design paradigms:

• Database: RELATIONAL PARADIGM, from relational algebra, an offshoot of first-order logic and of algebra of sets
• Application: OO (Object Orientation) PARADIGM

ORM = the process of mapping “objects” to “relational databases”
Mapping “objects” to “relational databases” (1): property mapping

• ATTRIBUTES, PROPERTIES
  – Persistent data only (a subset of all possible attributes: for example, an Employee object may have an averageSalary attribute that is needed within your application but isn’t saved to the database because it is calculated by the application)
  – “Shadow information”: data that objects need to maintain, above and beyond their normal domain data. This typically includes primary key information, particularly when it has no business meaning, concurrency control markings such as timestamps or incremental counters, and versioning numbers.

• Mapping how
  – One persistent attribute -> one column (more columns)
  – One class / object -> one table
Mapping “objects” to “relational databases” (2): relationship mapping

• RELATIONSHIPS BETWEEN OBJECTS [from different classes]
  – A mapping that describes and supports how to make a relationship (inheritance, aggregation, or composition, association) between two or more objects [classes] persistent at Application level

• METHODS AND METADATA WITHIN OBJECTS (controversial? However RARE!!!)
  – Sometimes, it may be necessary to save data about two or more “methods” related to a single property of an objects (“setters” and “getters”)
Relationship mapping at DB level: inheritance

Inheritance creates a semantic "is a" relationship

There are usually three primary solutions for mapping inheritance into a relational database, and a fourth supplementary technique that goes beyond inheritance mapping.

These techniques are:

- Map the entire class hierarchy to a single table
- Map each concrete class to its own table
- Map each class to its own table
- Map the classes into a generic table structure
Relationship mapping at DB level: inheritance (example from [1])

Let’s assume four classes related by inheritance:
- **Person**, an abstract class,
- and three concrete classes:
  - Customer
  - Employee
  - Executive
each with one attribute only.

Be aware that the shown hierarchy could be better modelled via the Party ([3], [4]) or the Business Entity [2] patterns. For example, if someone can be both a customer and an employee you would have two objects in memory for them, which may be problematic for your application.
Relationship mapping at DB level: inheritance: how to map it

Map the entire class hierarchy to a single table (Figure 1)
Map each concrete class to its own table (Figure 2)
Map each class to its own table (Figure 3)
Map the classes into a generic table structure (Figure 4)
Relationship mapping at DB level: aggregation, composition, association

A simple rule to follow with relationship mapping is that you should keep the multiplicities the same, within the Application and the DB level.

Still, remember that “a one-to-one data relationship is a subset of a one-to-many data relationship and a one-to-many relationship is a subset of a many-to-many relationship”. [1]

This means that you could implement a one-to-one object relationship with to a one-to-many or even a many-to-many data relationship...

... Which needs to be normalised (usually by creating a join table!)
Object-relational “mappers”!!!

• Once you have normalised your DB
• You still want to use OO for developing your application
• Where Objects should relate to each other without needing any “join objects” (i.e., any similar solution to the “join tables”)
• Someone (mainly Fawler, [5]) specified the first O/R Mapper, as a design pattern...
The ActiveRecord pattern (and its implementation as a class)

The **active record pattern** is a design pattern found in software that stores its data in relational databases.

When implemented into a class, the interface to such an object would include functions such as Insert, Update, and Delete (the “CRUD” operations), plus properties that correspond more-or-less directly to the columns in the underlying database table(s)
The ActiveRecord class (Jake Grimley’s PHP implementation)

Implementations of the concept can be found in various development frameworks for many programming environments. One of simplest implementation in PHP is from Jake Grimley (we adopt his MyActiveRecord.0.4.php implementation, a newer version is available from [7]):

* This class achieves simplicity of use and implementation through the following 'by-design' limitations:
  * 1. This class talks to MySQL only.
  * 2. Table/Class mapping is achieved by each database table being named IDENTICALLY to the MyActiveRecord subclass that will represent it (but in lowercase for compatibility reasons).
  * 3. Every database table mapped by MyActiveRecord MUST have an autoincrementing primary-key named `id`.
How to use MyActiveRecord class
(a Driver-Car example in PHP)

The class is defined within a PHP file, as follows:

```php
class MyActiveRecord {
    /* ... Many functions have been defined, out of which... */
    function FindBySql( $strClass, $strSQL, $strIndexBy='id' ) { ... }
    function find_attached($strClass, $strCondition=NULL) {
        return $this->find_linked($strClass, $strCondition);
    }
}
```

In order to use it, each class with persistent data (and its table) should extend the MyActiveRecord class (some PHP specific syntax constraints apply)

```php
class Driver extends MyActiveRecord {
}

$driver = MyActiveRecord::FindBySql('Driver', 'SELECT * FROM driver WHERE id > 1');
foreach($driver as $driver_key => $driver_value) {
    $driver_value->drives = $driver_value->find_attached('Car');
    echo "<P> - "$driver_value->last." (id: ".$driver_value->id., class: ".$driver_value->class.")";
}
```
ORM for the Driver-Car example

- Two classes in a many-to-many relationship.
- Three tables
- MyActiveRecord methods can populate the “Driver/Car” join table, and, generally speaking, perform all CRUD operations on all the tables.
References


