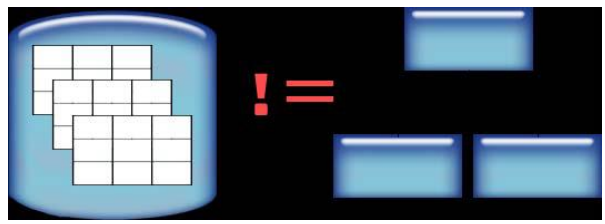
Software Engineering III

Object Relational Mapping (ORM)

Object – Relational Mismatch

The relational model does not equal the object-oriented model.



ORM Introduction

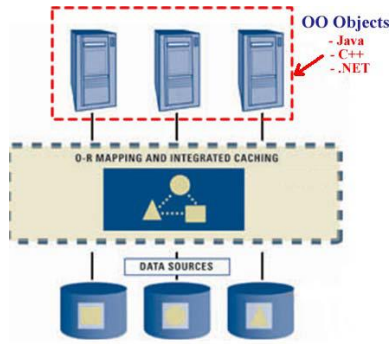
- A technique for converting data between a relational database and an object-oriented programming language.
- A data persistence strategy
- Data persistence code/package
- The result of an ORM implementation is often likened to that of a 'virtual object database'.
- ORM provides database independence
- The ability to seamlessly manipulate data stored in a relational database using an object programming language
- Contrast to:
  - A call interface used by ODBC or JDBC;
- **It provides a way to resolve the object-relational impedance mismatch.**
  - **This object-relational impedance mismatch is considered to be the core problem.**

Object-relational Mismatch Issues

- How to map columns, rows, and tables to objects?
- **How to deal with relationships?**
  - How to deal with associations?
- **How to map object inheritance to relational tables?**
- How to deal with the different design goals
  - The relational model is designed for data storage and retrieval. Its focus is in terms of how to best **manage data**.
  - The OO model is all about how to best **model behaviour**.
- How to make objects persistent?

The Goals of ORM

- Take advantages of the things that Relational Database technologies do well
- Use the rich features of Object Technologies

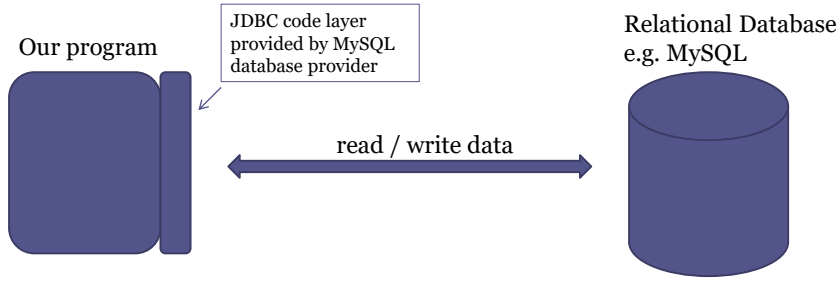


Database Connectivity

- **Open Database Connectivity**(ODBC) -standard software API for using RDBMS.
  - ODBC spec. offers a procedural API for using SQL queries to access data.
  - Programmer can write applications without concern for the specifics of each RDBMS encountered.
- **Java Database Connectivity**(JDBC)

Database Connectivity

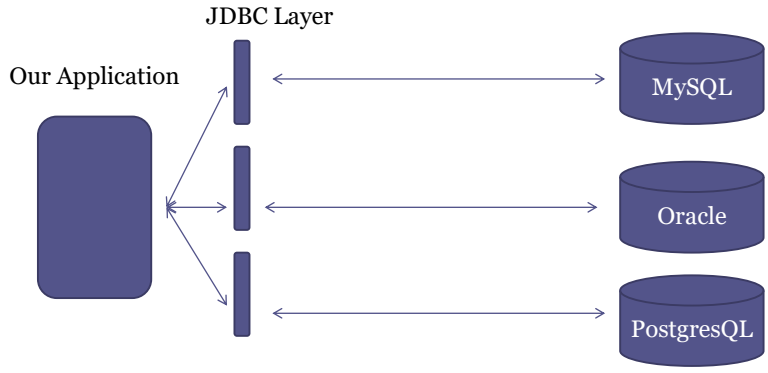
Java Database Connectivity (JDBC)



Most major RDBMS providers supply JDBC code libraries to allow java developers to easily connect to their database.

Database Connectivity

Java Database Connectivity (JDBC)



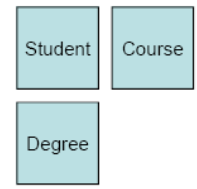
- Each RDBMS has its own implementation of the JDBC Java Interfaces
- **E.g.** Each JDBC library must have a class that implements the `Statement` interface
- `Statement` - The object used for executing a static SQL statement and returning the results it produces

JDBC Approaches

- **Write SQL conversion methods by hand**
  - Tedious and requires lots of code
  - Extremely error-prone
  - Non-standard SQL ties the application to specific databases
  - Vulnerable to changes in the object model
  - Difficult to represent associations between objects

```
public void addStudent( Student student )
{
    String sql = "INSERT INTO student ( name, address ) VALUES ( '" +
      student.getName() + "', '" + student.getAddress() + "' )";

    // Initiate a Connection, create a Statement, and execute the query
}
```



Other Approaches

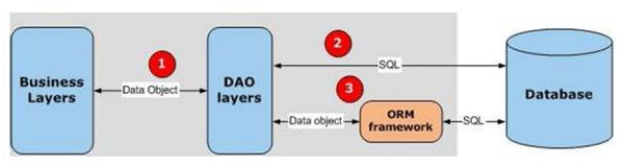
- Use Java serialization
  - Object persistence in Java
  - Write application state to a file:
    - Can only be accessed as a whole
- Object Oriented Database Systems
  - No complete query language implementation exists
  - RDMS standards more stable

The Preferred Solution

- Use an *Object-Relational Mapping System* (e.g. EclipseLink, Hibernate)
- Provides a simple API for storing & retrieving Java objects directly to and from the database
- *Transparent*: object model is unaware

ORM Architecture

- Middleware that manages persistence
- Provides an abstraction layer between the domain model and the database.



1) Data objects passed to DAO layer
2) Method: SQL conversion
3) Method: ORM using data/transfer objects

Advantages of ORM

- Make RDBMS look like ODBMS
- Data are accessed as objects, not rows and columns
- Simplify many common operations e.g.

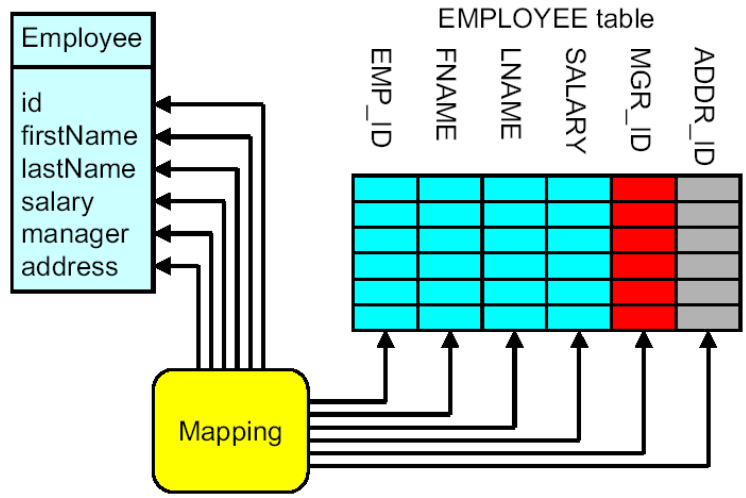
  manager.find(Company.class, "name = 'Mitsubishi'")
- Improve portability
  - Separate DB specific SQL statements from application code
- Productivity
  - Eliminates lots of repetitive code – focus on business logic
  - Database schema is generated automatically
- Maintainability
  - Fewer lines of code –easier to understand
  - Easier to manage change in the object model
- Performance
  - Lazy loading –associated/linked objects are fetched only when needed
  - Caching
- Database vendor independence
  - The underlying database is abstracted away
  - Can be configured outside the application

How do we Map the Objects to the Relational Database Tables?

Two Approaches to Mapping

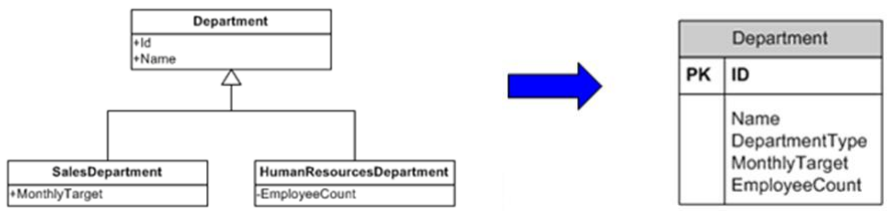
- Top down
  - Create a database schema given a class diagram
- Bottom up
  - Define a class diagram given a database schema

Basic ORM



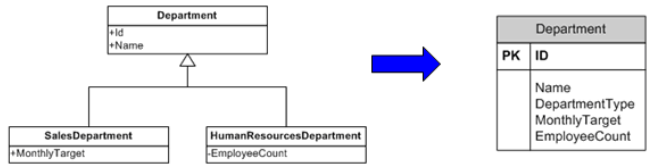
Map **Hierarchy** to a Single Table

- The attributes from all the classes are extracted into a single table.
- An additional attribute is added to indicate the type of Department

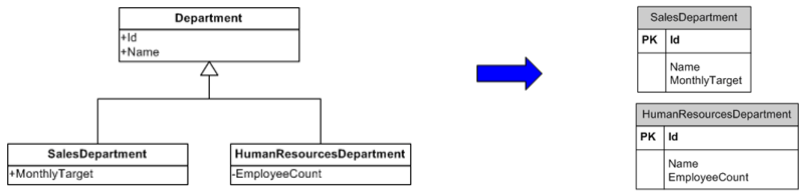


Map **Hierarchy** to a Single Table

- Advantages
  - approach is simple
  - easy to add new classes.
  - only one table - no need for table joins giving efficient data retrieval.
- Disadvantages
  - Not all attributes are relevant resulting in many null or empty attributes
  - The tables may not comply with normalization practices.



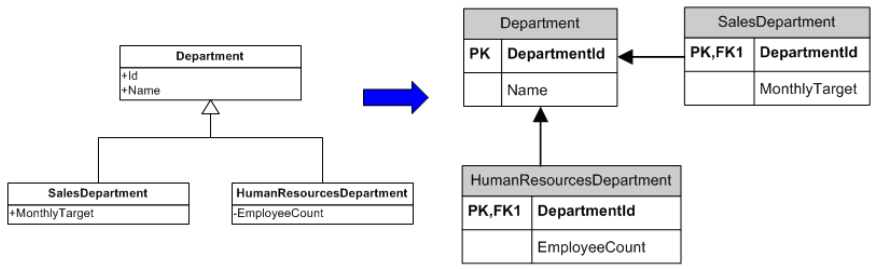
Map **Each Concrete Class** to a Table



- A table is created for each concrete class.
  - The attributes of the base class are included for each table.
- Advantages
  - Good performance in terms of accessing a single object's data.
- Disadvantages
  - A class change requires a change to its corresponding table and any corresponding tables of its child classes.
  - Can result in much repeating data

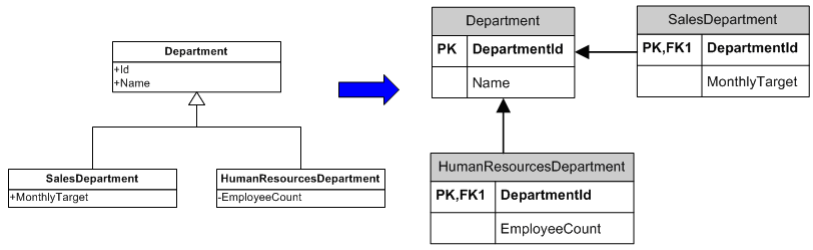
Map **All Classes** to a Table

- Every class has an associative table in the database.
- Advantages
  - Easy to add or modify subclasses
  - Easy to modify base class
  - The one-to-one mapping makes this approach easy to understand



Map All Classes to a Table

- Disadvantages
  - Data access in terms of a reads and writes can be slow because there are more tables involved.
  - More table joins may be required in order to perform a data related operation.
  - Ad-hoc reporting can be difficult

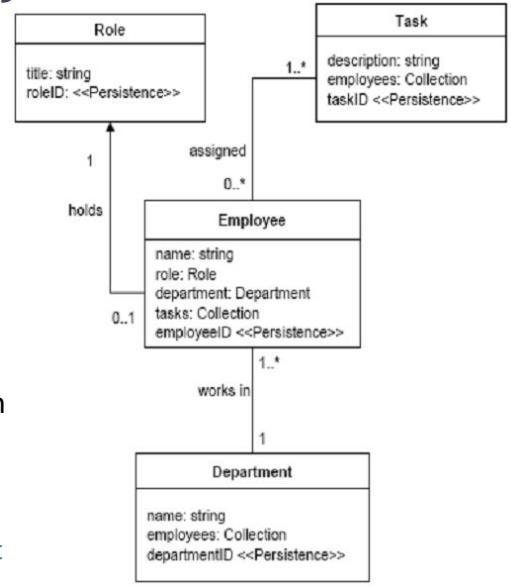


Mapping Object Relationships in Tables

- Class Associations
  - One to One
  - One to Many
  - Many to Many
  - Uni-directional Association
  - Bi-directional Association

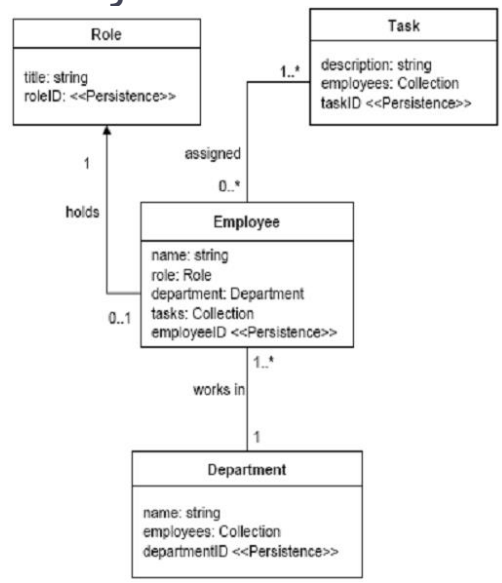
Types of Class/Object Associations

- One to One
  - Employee to Role
- One to Many
  - Department to Employee
- Many to Many
  - Employee to Task
- Uni-directional Association
  - Employee to Role
- Bi-directional
  - Employee <-> Department



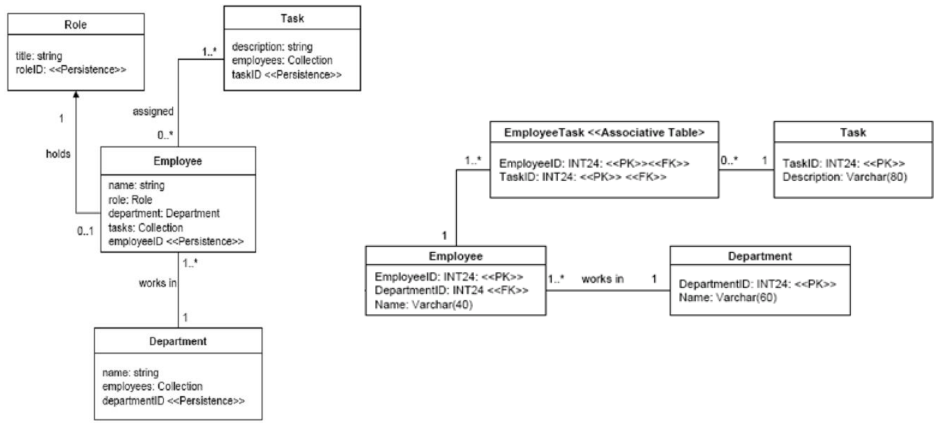
Implementing Class/Object Associations

- Use of References and operations
- One-to-one uses a reference with getters/setters
- One to many uses a collection (ArrayList, Vector) with add(), remove(), etc.



Mapping Object Associations as Joins in Relational DB

- Joins implemented using foreign keys FKs
- One to Many implemented by FK in the Many table
- Many to many implemented using a join or associative table



Retrieving an Object and Relationships from Database

- Example – Retrieving a Department from the DB

1. Read *Department* row from the database
2. Instantiate a *Department* object and set the attributes – remember one of the attributes is a *Collection*
3. Read all related *Employees* from the database
4. Instantiate an *Employee* object for each and set the attributes
5. Add a reference to each *Employee* object to the *Collection* (attribute of the *Department* object)

Saving Objects and Relationships

- Example
  - Create a SQL transaction to ensure referential integrity
  - Add update/insert statements for each object to the transaction.
    - Update/insert statement includes both the attributes and the key values.
  - Submit and commit the transaction.

Some ORM Tool Vendors

- **Hibernate**
  - http://www.hibernate.org/
- **Oracle TopLink**
  - http://www.oracle.com/technetwork/middleware/toplink/overview/index.html
- **CocoBase**
  - http://www.thoughtinc.com/cber_index.html
- **MyBatis**
  - www.mybatis.org
- **EclipseLink**
  - http://www.eclipse.org/eclipselink/

- Java Specifications
  - Java Data Object (JDO)
    - One of the Java specifications
    - Flexible persistence options: RDBMS, OODBMS, files etc.
  - Java Persistence API (JPA)
    - Hibernate 3.2 onwards supports the JPA

ORM Tools Features

- Basic features
  - Be able to use inheritance, create hierarchies between entities
  - Handle any type of relations (1-1, 1-n, n-n)
  - Support for transactions
- Support various databases.
  - A big advantage of mapping tools is that they provide an abstraction of the underlying database engine.
  - Most of them allow switching easily between RDBMSs
- Be able to map a single object to data coming from multiple tables (joins, views).
  - Most of the tools handle a direct mapping of a class to one table.
- GUI to set up the mapping.
  - Such a graphical tool presents the relational data model and lets you specify the objects to be created or at least the links between the objects and the tables.
- Generation of the classes.
  - This can speed up the development
  - In some cases the database is mapped to hand-coded classes which may be a preference
- Generation of the database schema.
  - Some tools work only with a database they generated.
  - Big issue for legacy databases
- Support for stored procedures in SQL

ORM Tools Features

- **Lazy loading**
  - the loading of some of the related data as it is needed
- **Cache dynamically generated queries**,
  - so that they don't get rebuilt at each call.
- **Cache some data**
  - to avoid too many calls to the data source.
- **Optimized queries**
  - update only the modified columns;
- **Bulk updates or deletions**.
  - When thousands of records to be updated, avoid loading all the objects in memory,
- Use a query (DELETE FROM Customer WHERE Balance < 0).