



# Object Relational Mapping

---

Parsu Nurani, Softech Consulting, pnurani@bellsouth.net

Gopal Koratana, ASIA Systems, kgopalrao@gmail.com

2005 Georgia Oracle Users Conference

April 4<sup>th</sup>, 2005



# Speaker's Profile



Softech Consulting

- **Parsu Nurani, Softech Consulting**
  - 12+ years of software consulting experience
  - Architecture, design, and development of enterprise multi-tiered custom applications and EAI
  - Design and delivery of Business Intelligence and Data Warehouse applications
  - Sun Certified J2EE Architect
  - Adjunct Professor, Devry University



# Speaker's Profile

---

- **Gopal Koratana, ASIA Systems**
  - 7+ years of software consulting, product development and offshore development.
  - Architecture, design and development of large scale OO systems. Extensively worked
    - Java, J2EE and C++.
    - Open source frameworks, platforms, databases and application servers.



# Purpose of This Session

---

- Understand the role of object and relational technologies in software development
- Understand “Impedance Mismatch” between object and relational technologies
- Understand how to map objects to relational databases
- Understand how to implement these mappings



# Intended Audience

---

- Database people
- Object modelers
- Application developers
- Architects
- Managers
- Academic people in CIS
- Understanding of database concepts is assumed
- A basic understanding of OO concepts is needed
  - Class (Attributes and Behavior)
  - Class Model
  - Inheritance
  - Polymorphism
  - Associations (One to One, One to Many, Many to Many)



# Object Technology

---

- Used to model business domain (Inheritance, Polymorphism, Encapsulation, and Associations)
- It is the right technology to encapsulate business logic and create scalable software solutions
- We still need some way to persist and store the business objects and attributes (data)



# Relational Technology

---

- Well established technology that has been around for several years
- Used to store and query large amounts of data efficiently – object oriented databases failed miserably!!
  - Searching
  - Sorting
- Sharing
  - Concurrency
  - Transactions
- Integrity
  - Constraints
  - Transactions
- Not good for business modeling
- Not good to express business logic (Stored procedures not normally good for business logic)



# Object/Relational Differences

---

- Technical Differences
- Cultural Differences



# Object/Relational Technical Differences

---

- Object-oriented paradigm based on software engineering principles used to capture and model business domain
- Relational paradigm based on mathematical principles and used to store data
- Objects are traversed through relationships
- Relational paradigm joins data from tables



# Object/Relational Cultural Differences

---

- Object people and data people look at data differently
- Scott Ambler calls this the “object-data divide”



# Consequences of “Object-Data Divide”

---

- Software is not delivered on time and within budget
- The technical mismatch between object model and data model is worsened
- Data models often fail to be good drivers for the object model
- Frustration within the organization causes higher turnovers.

# Solving the Impedance Mismatch



---

- Technical mismatch can be overcome by educating yourself and team on both technologies
- Decouple object world from database access as much as possible
- Cultural mismatch is harder to solve, but solvable
- Use Object Relational Mapping

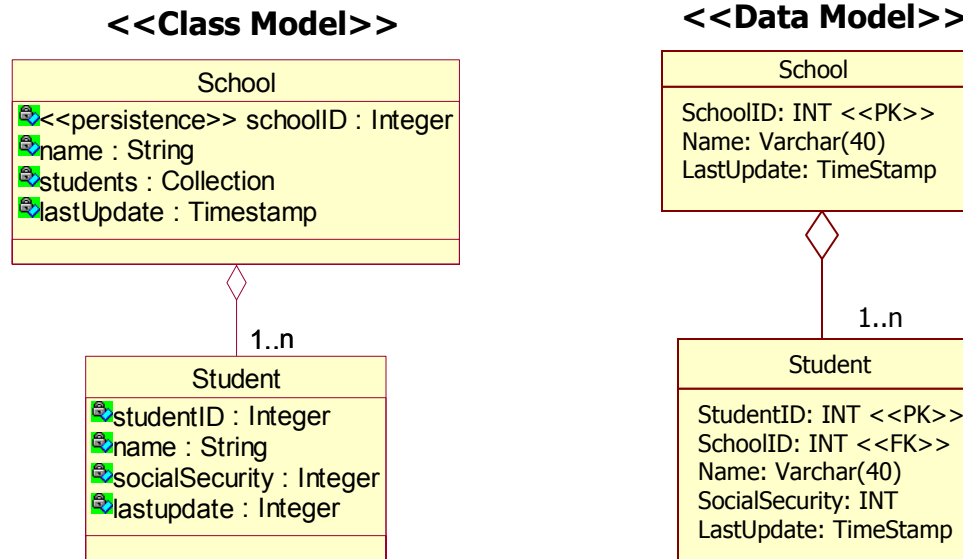


# Goal of Object Relation Mapping Technology

---

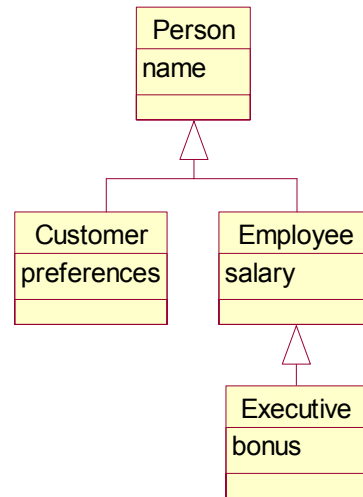
- Take advantage of those things that relational databases do well
- Keep using the rich features of object technology
- Automate and reduce work (tools/framework)
- Ensure DBAs and object people are both happy!

# Basic Mapping



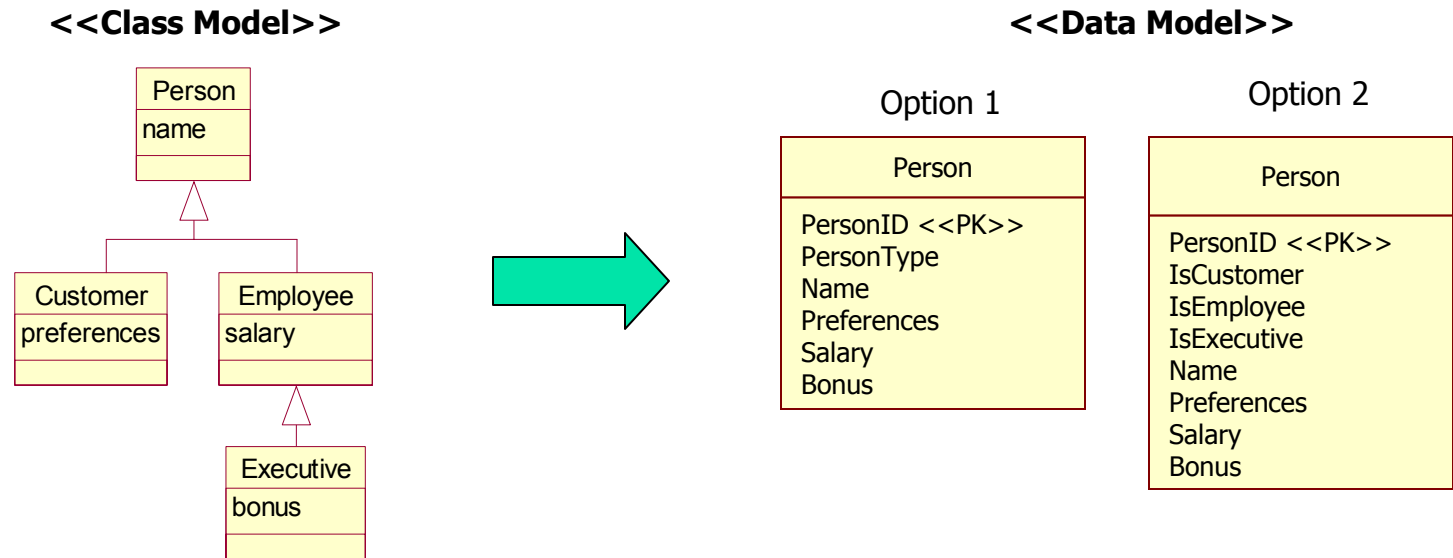
- One to one mapping of class attributes to table columns
- Shadow Information (Primary Key, Foreign Keys, Time Stamp)

# Mapping Inheritance



- Map the entire class hierarchy to a single table
- Map each concrete class to its own table
- Map each class to its own table

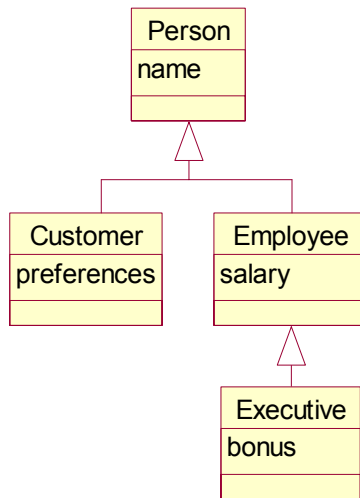
# Map Entire Class Hierarchy to a Table



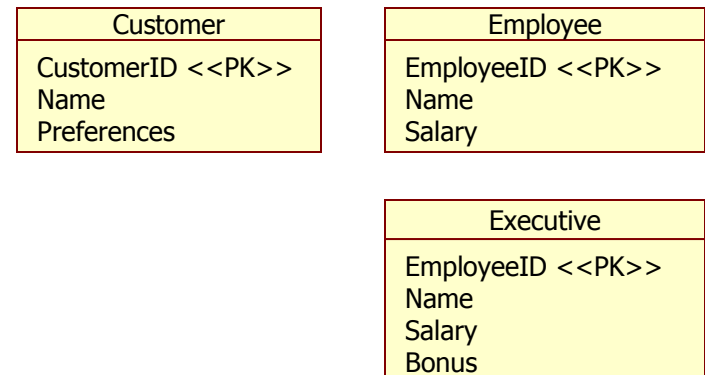
- PersonID – primary key
- PersonType – class discriminator
- isCustomer, isEmployee, isExecutive -alternate class discriminator

# Map each concrete class to its own table

<<Class Model>>

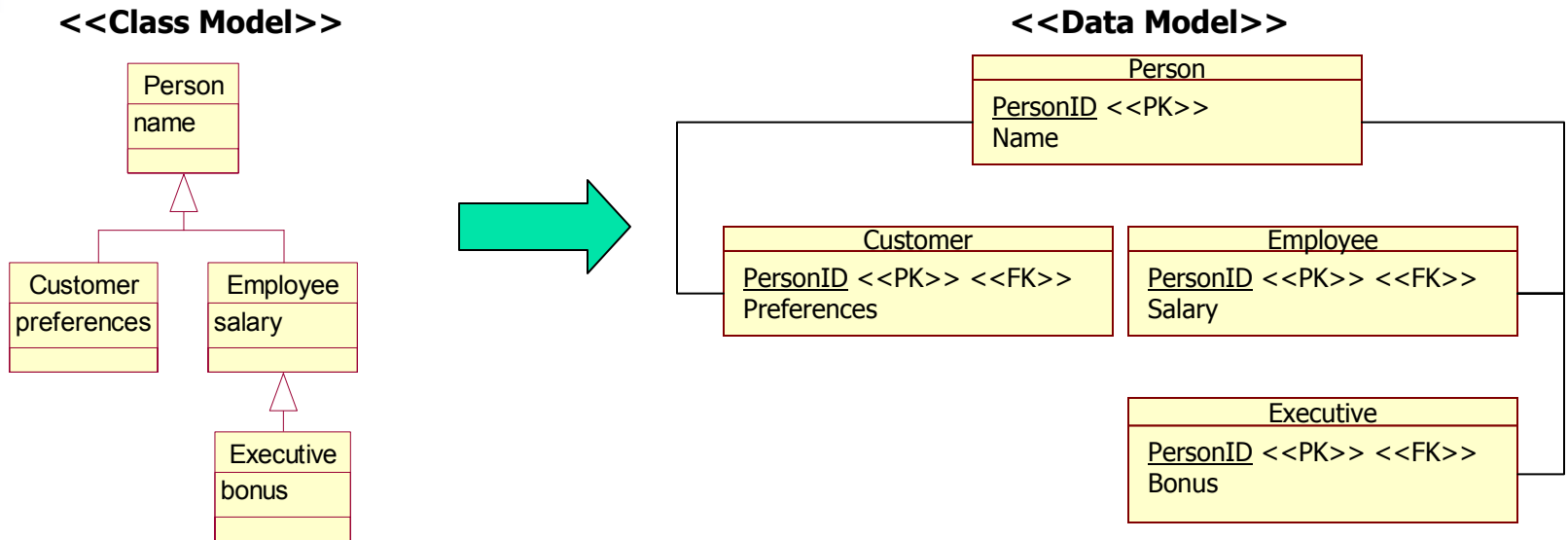


<<Data Model>>



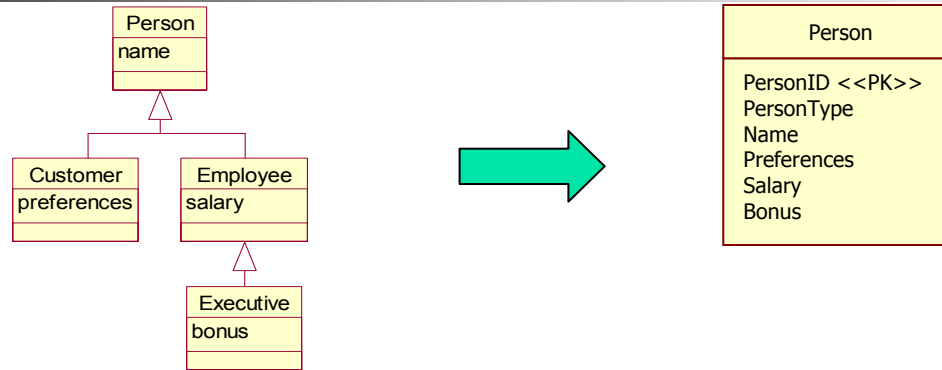
- A table for each concrete class containing attribute of the class and inherited attributes

# Map Each Class To Its Own Table



- Each class mapped to a single table
- The data for the Customer (“joined subclass”) is stored in two tables
- PersonID is both a PK and FK in Customer, Employee, and Executive tables to maintain relationship with Person table

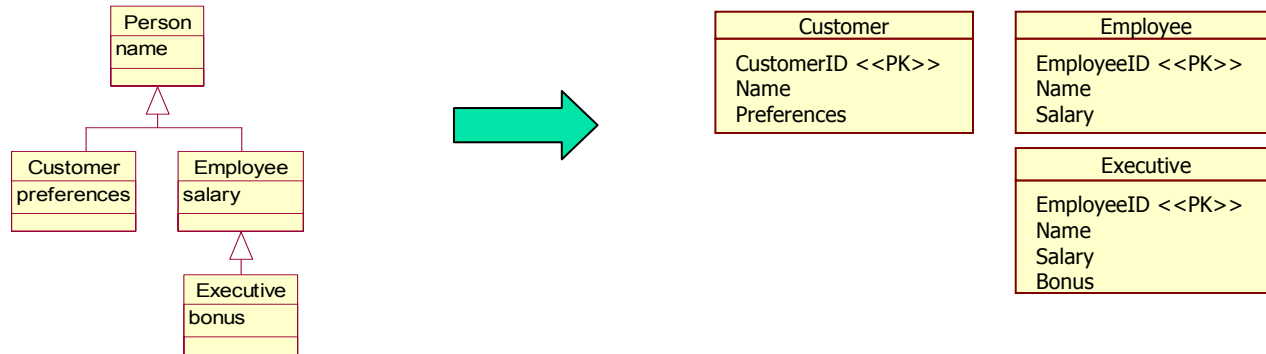
# One Table Per Hierarchy – Tradeoff



- Good strategy for simple and shallow hierarchies
  - Advantages
    - Simple
    - Easy to add classes
    - Easy support for polymorphism
    - Single table leads to good data access performance
  - Disadvantages
    - Changes to any class will affect schema
    - Space wasted in database
    - Table size can grow significantly with hierarchies

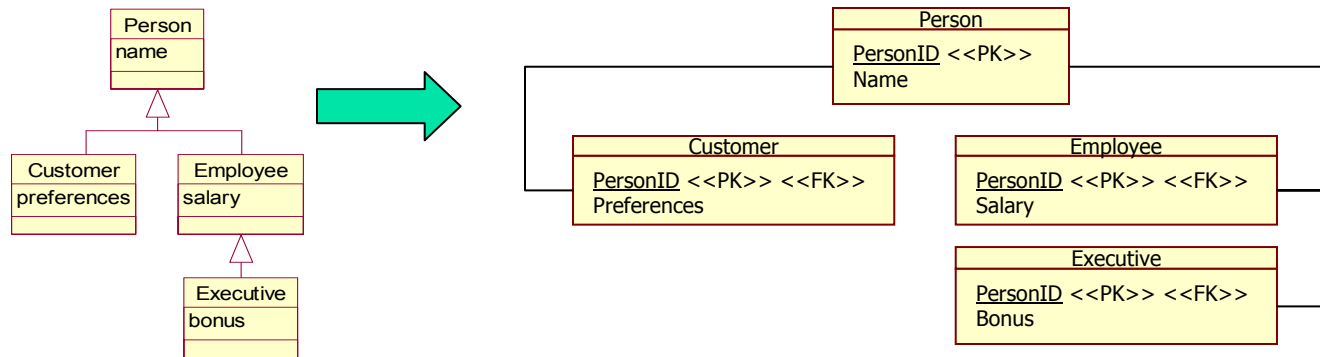
# One Table Per Concrete Class

## – Tradeoff



- Good strategy when changing types is rare
  - Advantages
    - Easy to do Ad-hoc reporting
    - Good performance
  - Disadvantages
    - Change to a class will affect its table and the table of its subclasses
    - Whenever an object changes its role, you need to copy data into the appropriate table

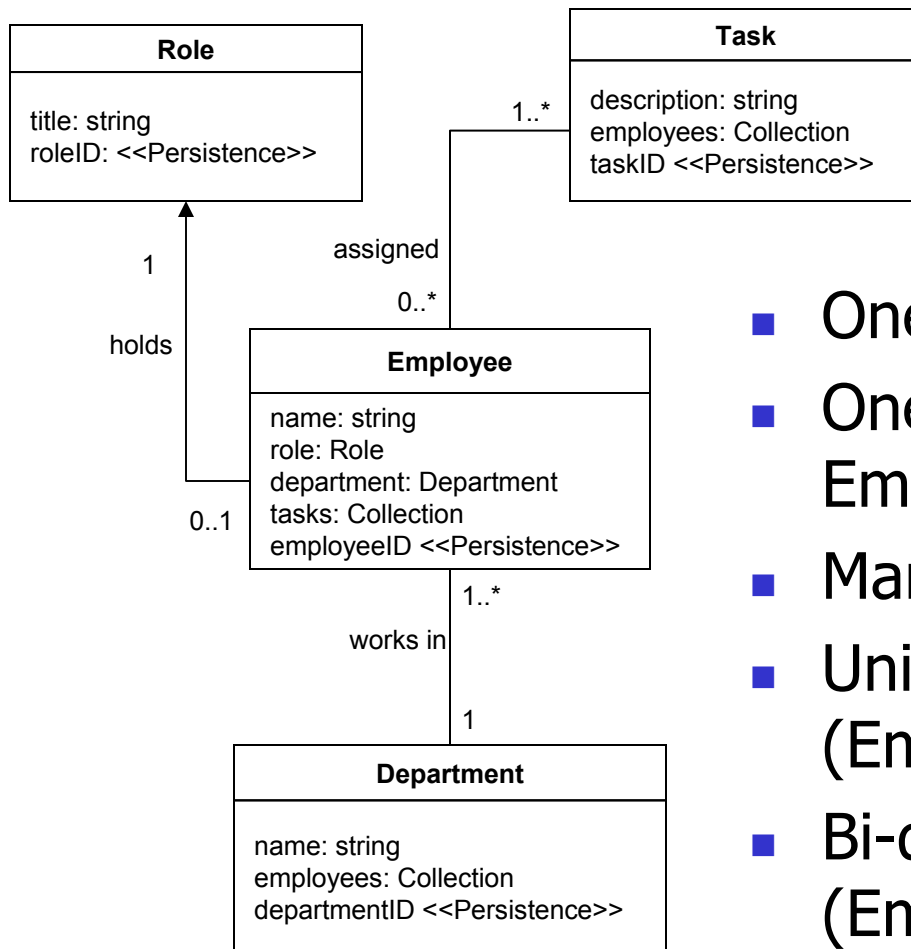
# One Table Per Class - Tradeoff



- Good strategy when types change often
  - Advantages
    - Easy to understand because of one-to-one mapping
    - Supports polymorphism easily – each type in a separate table
    - Changes to subclass and addition to superclass affect addition/modification of one table
  - Disadvantages
    - Extra joins are needed to get to a subclass – could lead performance issue
    - Ad-hoc reporting is difficult unless you add views

# Mapping Object Relationships

<<Class Model>>



- One to One (Employee-Role)
- One to Many (Department-Employee)
- Many to Many (Employee-Tasks)
- Uni-directional Association (Employee-Role)
- Bi-directional Association (Employee-Department)



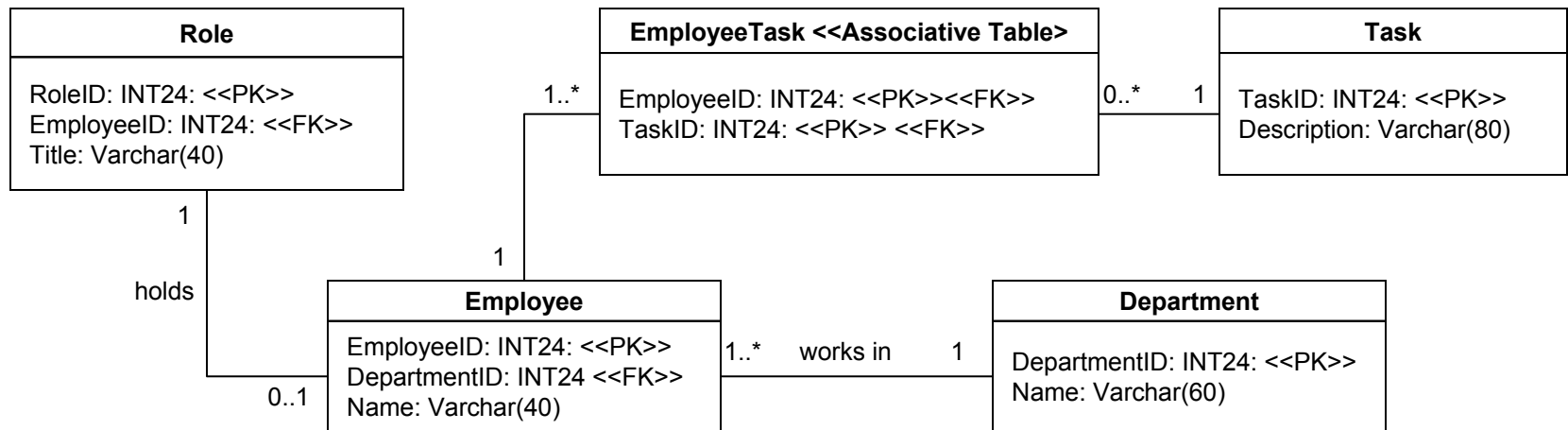
# How Object Relationships are Implemented

---

- Relationships implemented using reference and operations
- Multiplicity of one (0..1 or 1) implemented with reference to an object and getters/setters (*department*, *getDepartment()*, *setDepartment()*)
- Multiplicity of many (N, 0..\*, 1..\*) implemented with a collection attribute e.g., Array, Collection and operation to manipulate the array/collection (*tasks*, *addTask()*, *removeTask()*)

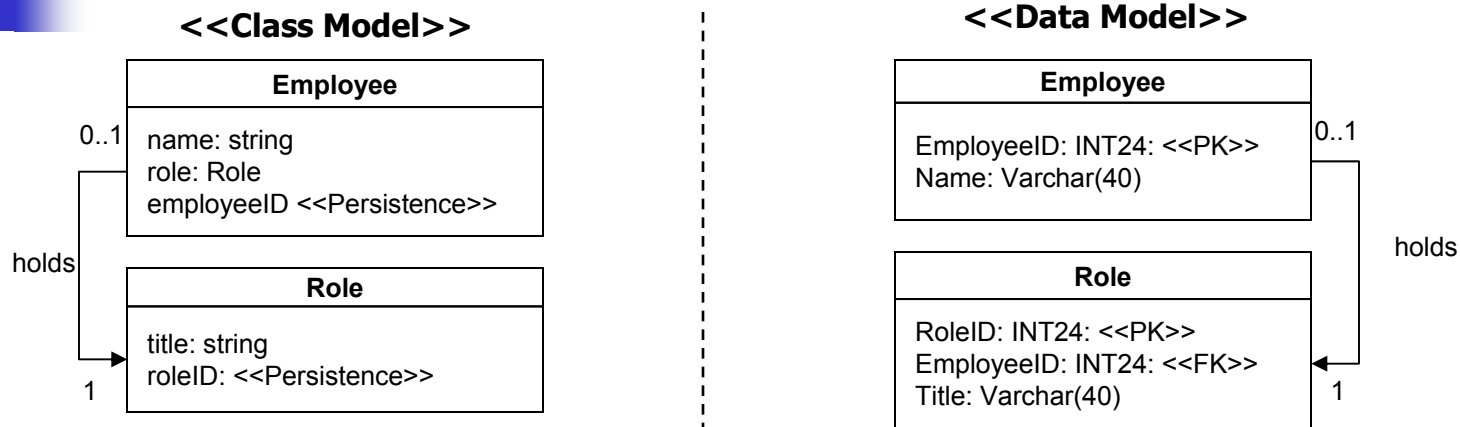
# How Relational Database Relationships Are Implemented

## <<Data Model>>



- Relationships in databases implemented using FKs
- One to many – FK from “one table” to “many table” (DepartmentID)
- Many to Many – use associative table and convert into two “one to many” associations

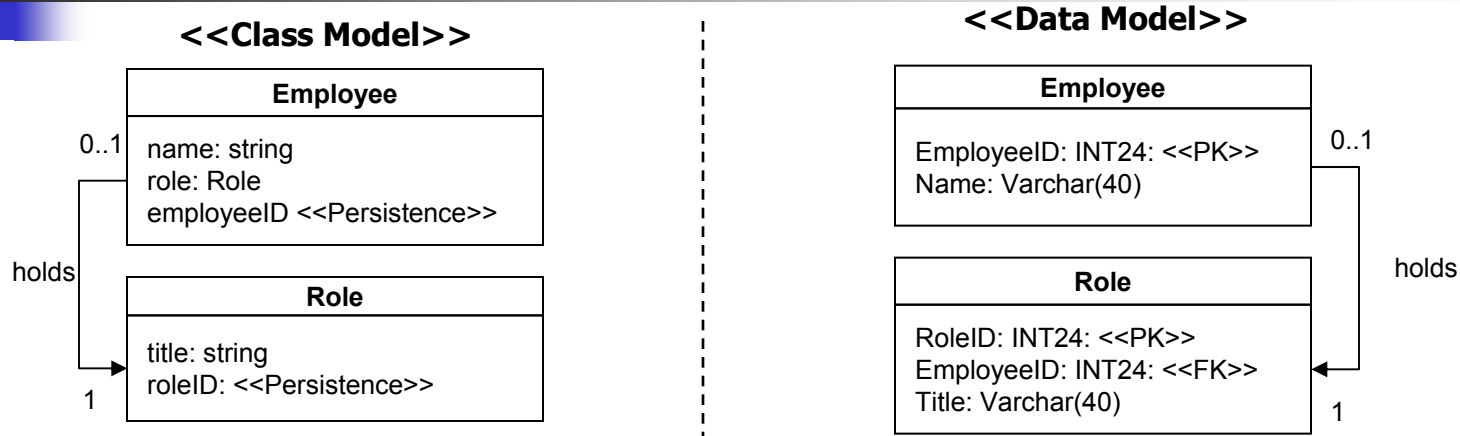
# Reading Object(s) and Relationship(s)



Retrieving single *Employee* object, step-by-step

1. The *Employee* object is read into memory.
2. Join *Employee* and *Role* table on *EmployeeID* to identify row in *Role* table
3. The *Role* object is read in and instantiated.
4. The value of the *Employee.role* attribute is set to reference *Role* object.

# Saving Object(s) and Relationship(s)



1. Create a transaction to ensure referential integrity
2. Add update statements for each object to the transaction
3. Update statement includes both the attributes and mapped key values Relationships are persisted by updating the keys
4. The transaction is submitted to the database
5. Transaction is committed



# Tools & Examples

---

## ■ What do you get out of them?

- Mapping (The “Object world” to the “Relational world”).
- Clean API level access to data.
  - Flexibility in defining query.
    - Object based queries and/or traditional SQL based queries.
    - Transparent and highly optimized conversion “records set” to “object” and vice versa.
- Clean transaction management
  - Declarative transaction management with AOP frameworks like spring (fully integrated with hibernate).
- Cleaner Maintainable code
- Easy to change and well tuned applications
- Forces to build organized Entity Relation



# Some Good Tools

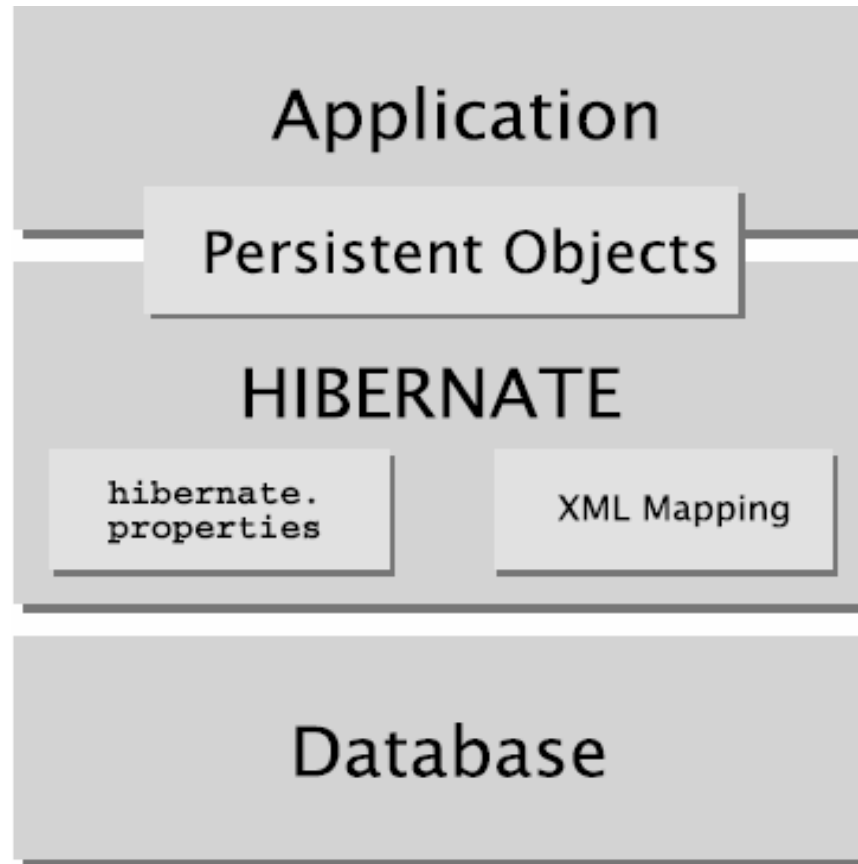
---

- Oracle TopLink (Oracle)
- Hibernate (Opensource)
  - We will discuss this in more detail.
- Cocobase (Thought)



# Typical Architecture

---





# Typical Usage

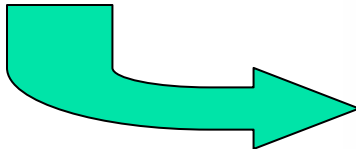
---

- **Configure** (One time)
  - Configure Hibernate. (Involves DB connection Info \*)
  - Prepare persistence class. (Java classes representing entities)
  - Prepare mapping for the class. (Mapping entities in relational database)
- **Runtime**
  - Build session. (Analogues to starting transaction)
  - Build query. (Object or SQL based queries)
  - Process results.(In Objects or records)
  - Close session. (End of transaction)

\* Database (DB2 to Oracle) can be changed by changing the "dialect" at configuration level. This of course depends on the need and if the application is designed as such.

# Typical Mapping

```
public class Cat {
    private Long id;
    private String name;
    private Date birthdate;
    private Cat mate;
    private Set kittens;
    private Color color;
    private char sex;
    private float weight;
}
```



```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping package="eg">

    <class name="Cat" table="CATS" discriminator-value="C">
        <id name="id" column="uid" type="long">
            <generator class="hilo"/>
        </id>
        <discriminator column="subclass" type="character"/>
        <property name="birthdate" type="date"/>
        <property name="color" not-null="true"/>
        <property name="sex" not-null="true" update="false"/>
        <property name="weight"/>
        <many-to-one name="mate" column="mate_id"/>
        <set name="kittens">
            <key column="mother_id"/>
            <one-to-many class="Cat"/>
        </set>
        <subclass name="DomesticCat" discriminator-value="D">
            <property name="name" type="string"/>
        </subclass>
    </class>

    <class name="Dog">
        <!-- mapping for Dog could go here -->
    </class>

</hibernate-mapping>
```



# Typical Transaction

---

```
Public void foo() {  
    Cat cat = new Cat();  
    cat.setName("Princess");  
    cat.setColor("white");  
    session = Util.getCurrentSession();  
    session.save(cat); // force insert / synchronize with db  
    session.commit(); // flush & commit.  
}
```



# More Goodies!!!!

---

- Who will maintain one more mapping document?
  - Good that you asked. Use xdoclet, MiddleGen and the mapping file gets generated when the App is being build.
  - Or use tools like hibernate to generate java objects and mapping files from the database schema.
- I am tired of defining transaction boundaries.
  - Good!!!!. Sometimes being lazy also helps. You still have to define them somewhere. How about using declarative transaction management.
    - Using AOP frameworks like Spring (Fully integrated with hibernate and other frameworks) you can do that.
- Can I do anything I used to do before with these tools.
  - Yes. Not only that, you can do it far better and cleaner, and the person coming after you will also thank you.



# More Goodies!!!!!!

---

- We come one step closer to bridge the “Object-Data Divide”.
- Helps build cohesiveness and better collaboration among the team.
- Helps reduce “Horse work” and thus increases efficiency and reduces errors.
  - Imagine a perfect system where all those complex outer join are
    - Generated, executed, optimized and converted to and fro from objects to tables and vice versa “Transparently”.
    - Now imagine a “OO geek” trying to do this by hand, and getting in to trouble with the DBA.
- Best part
  - With this approach both parties, Object and Relational get to do what they do the best.



# Acknowledgments & References

---

- <http://hibernate.org/>
- <http://springframework.org/>
- <http://www.oracle.com/technology/products/ias/toplink/index.html>
- <http://www.thoughtinc.com/>
- <http://www.ambysoft.com>
- Agile Database Techniques By Scott W. Ambler – This is a very good book. Concepts and examples from this book have been used in this presentation.



# Thank You! – Any Questions?

---

