LUXOFT TRAINING

JVA-000
Java Persistence with Hibernate

Module 1
Introduction

Objectives

- Understand Object and Relational models specifics
- Understand ORM and its problems
- Understand what is JPA and where to use it

The paradigm mismatch

Database normalization is typically optimized for storage

OO design is optimized for readability and maintainability

Sometimes the two conflict resulting in object-relational impedance mismatch... makes persistence challenging

The paradigm mismatch

Object-Relational Impedance Mismatch (*paradigm mismatch*) is a fancy way of saying that object and relational models don’t work very well together

- RDBMS represents data in tabular format
- OOP languages represents data as interconnected graph of objects

Loading/storing graphs of objects using RDBMS exposes us to 5 mismatch problems:

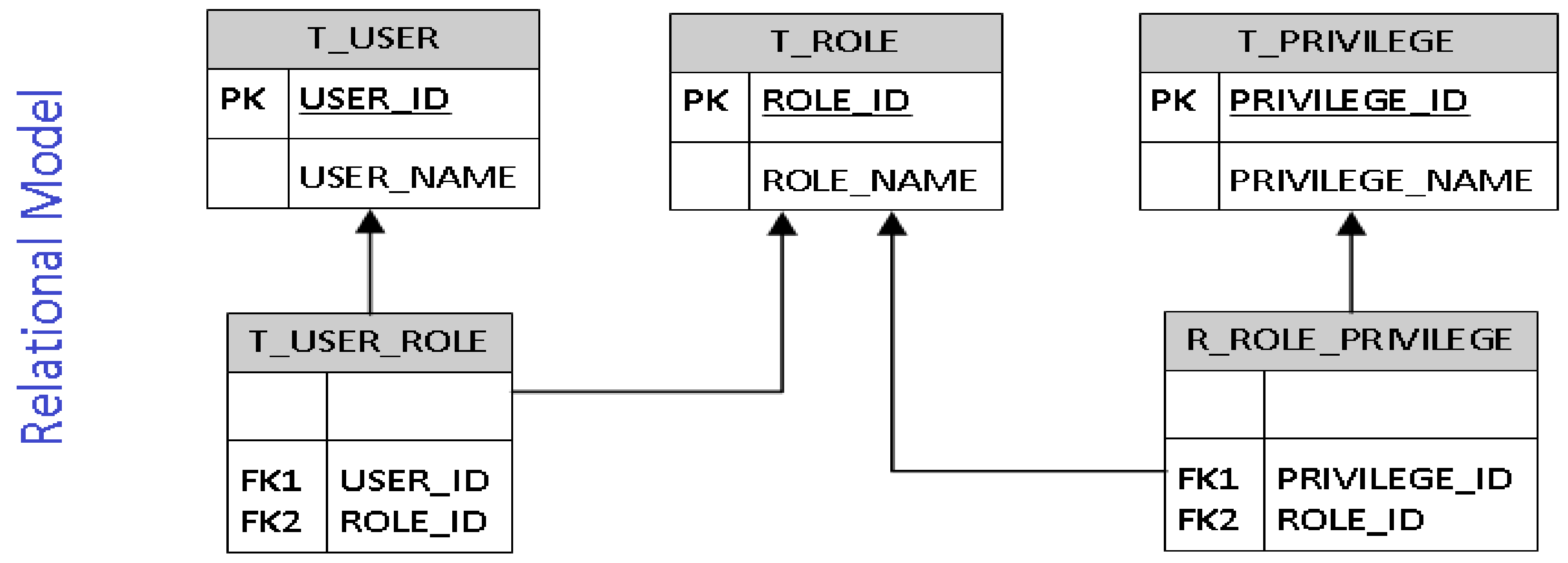
- Granularity
- Inheritance (subtypes)
- Identity
- Associations
- Data navigation

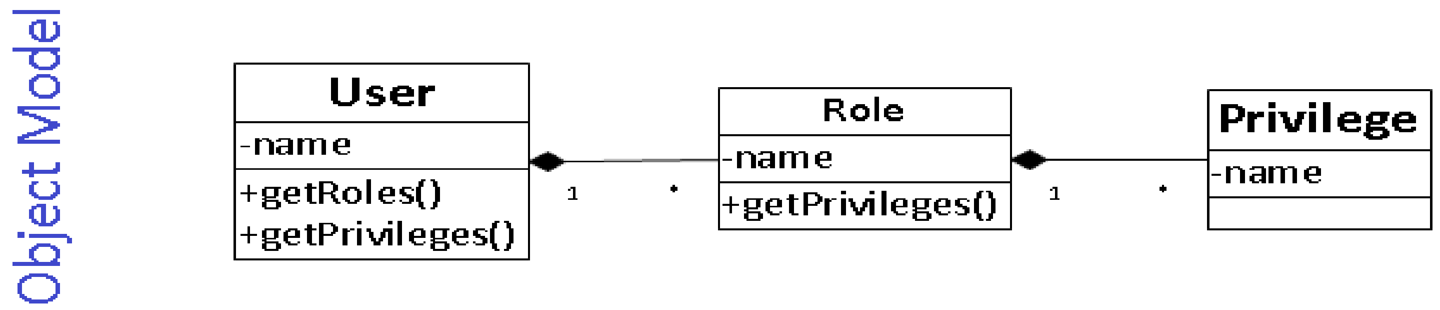
The paradigm mismatch

Mismatches:

- **Granularity**: object model has more granularity than relational model.
- **Inheritance**: not really represented well in database
- **Identity**: like object model, relational model doesn't expose identity while writing equality
- **Associations**: relational models cannot determine multiple relationships while looking into an object domain model.
- **Data navigation**: walk the objects in java, join the tables in a database

The paradigm mismatch





Object with Relational models

What is JPA?

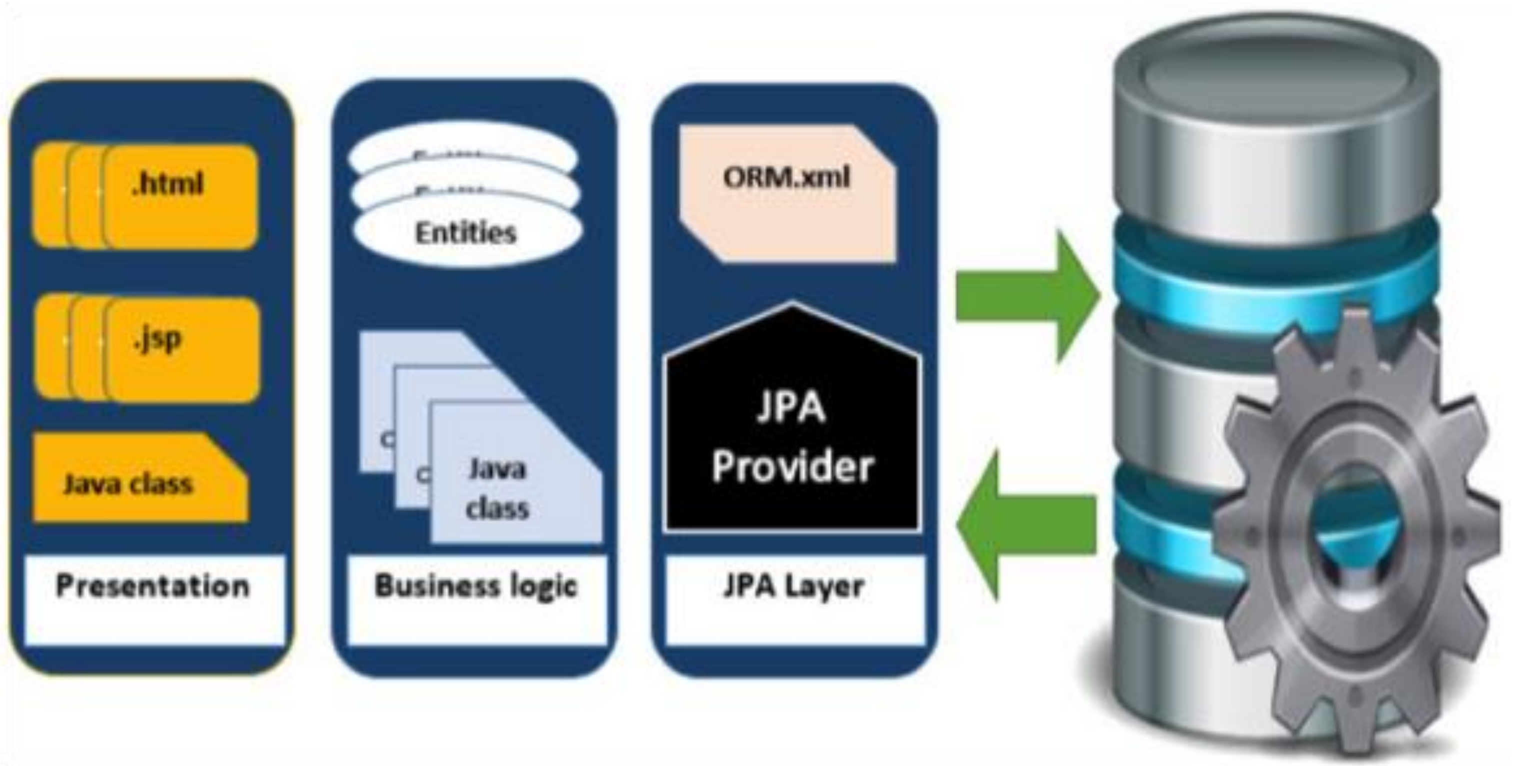
JPA is specification for Object/Relational mapping and Persistence in Java

Hibernate, EclipseLink, OpenJPA and others provide implementations of the JPA specification

Together they provide a framework to assist in mapping the object world to relational world a.k.a. ORM (Object Relational Mapping) tools

Uses annotations to map POJO and fields to database tables and columns

Where to use JPA?



JPA reduces the burden of writing codes for relational object management

Potential Benefits of JPA

Potential Benefits of JPA:

- Write less code
- Provides a consistent model for database interaction
- Performance
- Vendor independence - but only if you avoid the vendor specific features (database or JPA provider)
- Shields you from having to know SQL

Potential Drawbacks

Potential Drawbacks:

Complexity - JPA adds a layer of abstraction:

- harder to learn
- harder to debug
- performance issues creep in more often

Flexibility - lack of:

- harder to leverage database specific features
- Spring JDBC is closer to the metal

JPA History

Earlier versions of EJB, defined persistence layer combined with business logic layer using interface `javax.ejb.EntityBean`

- EJB 3.0 separates persistence layer and specifies JPA 1.0 (specifications released in JAVA EE5 on May 2006 using JSR 220)
- JPA 2.0 was released with the specifications of JAVA EE6 on December 2009 using JSR 317
- JPA 2.1 was released with the specification of JAVA EE7 on April 2013 using JSR 338

LUXOFT TRAINING

Thank you for your attention!

Questions?