Hibernate - I

ORM

Agenda

1 **Object / Relational Mapping**

Object / Relational Mapping (ORM)

Objectives

In this module you will be able to

- Define Persistence
- Identify the problems with Object/Relational mismatch
- Describe Alternatives for persistence management
- Define Object / Relational Mapping (ORM) in detail

Persistence

Persistence is the availability of the object/data even after the process that created it, is terminated

Persistence can be achieved through:

- Data Base Management System (DBMS)

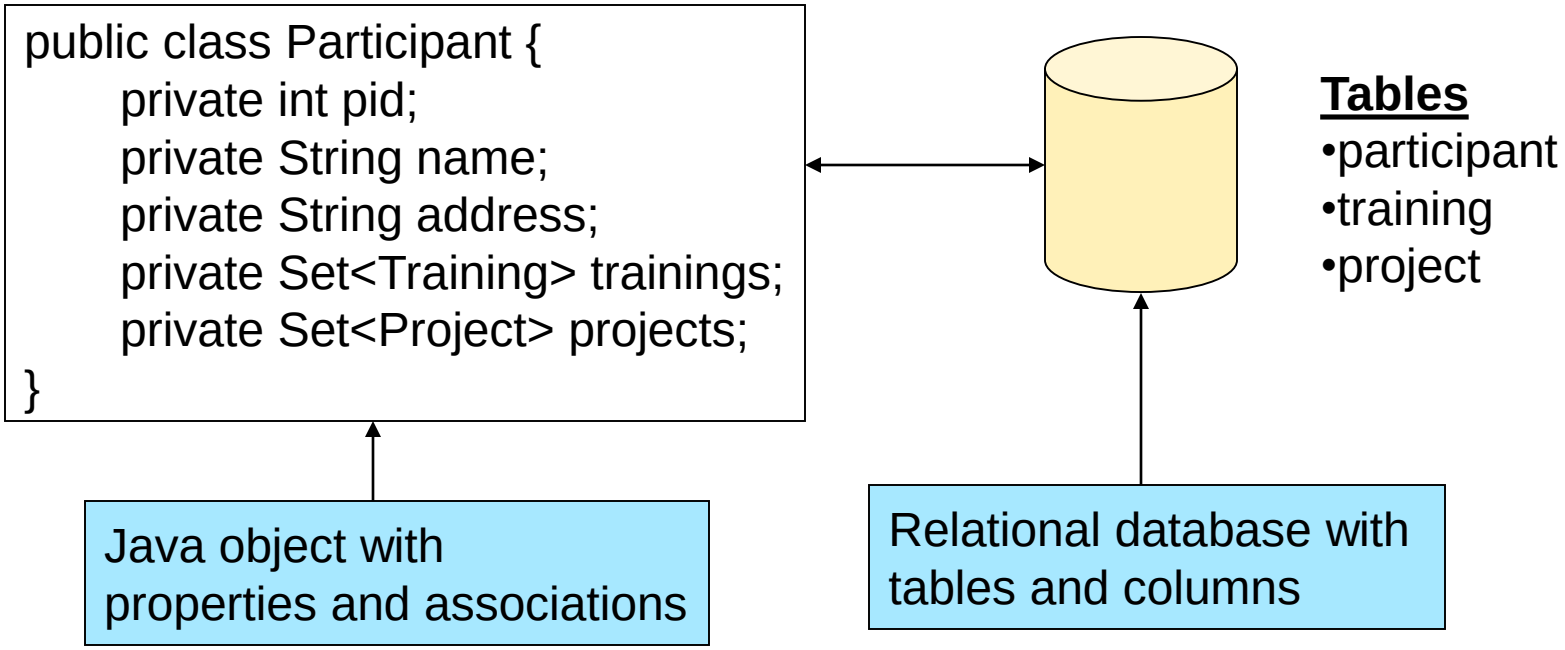
  Data is stored in the database and can be retrieved efficiently using Structured Query Language (SQL). DBMS provides features like Concurrency, Data Sharing, Data Integrity, and Data Security.

- Serialization

  It is a mechanism for storing objects to disk in such a way that they can be retrieved back again later

Object / Relational Mismatch

- Many object-oriented applications may have to implement both object model and relational model of some business entities
- When working with such applications, there is a mismatch between object model and relational database



Object / Relational Mismatch (Contd.).

Problems faced when a relational model is used to store objects:

- Problem of granularity
- Problem of Identity
- Problems related to associations
- Problem of subtypes
- Problem of data navigation

Object / Relational Mismatch (Contd.).

- **Problem of granularity**

Suppose you have to design and implement an online e-commerce application. In this application, you would need a class to represent information about a user of the system, and another class to represent information about the user's billing details. The SQL schema for USER and BILLING_DETAILS are given below.

```
create table USER1 (
USERNAME1 VARCHAR(15) NOT NULL PRIMARY KEY,
NAME1 VARCHAR(50) NOT NULL,
ADDRESS1 VARCHAR(100)
)
create table BILLING_DETAILS_1 (
ACCOUNT_NUMBER1 VARCHAR(10) NOT NULL PRIMARY Key,
ACCOUNT_NAME1 VARCHAR(50) NOT NULL,
ACCOUNT_TYPE1 VARCHAR(2) NOT NULL,
USERNAME1 VARCHAR(15) FOREIGN KEY REFERENCES USER1
)
```

Object / Relational Mismatch (Contd.).

- The most obvious problem with the above implementation is that Address is modeled as a simple String value. In most systems, it's necessary to store street, city, state, country, and ZIP code information separately. Of course, we could add these properties directly to the User class, but since it's highly likely that other classes in the system will also carry address information, it makes more sense to create a separate Address class. Basically, we have the choice of adding either several columns or a single column (of a new SQL data type). This is a problem of *granularity* (Bauer and King, 2006).

- **Problem of Identity**

Object identity is not same as database identity. Object identity can be established by checking for the references of 2 objects or using equals method for checking for equality. Database identity is established by having a primary key which identifies each record distinctly. But there will be problems when we have tables without primary key as databases support tables without primary key.

Object / Relational Mismatch (Contd.).

- **Problems related to associations**

In an object model it is easy to represent any kind of association using object references. One-to-one, one-to-many, many-to-one , many-to-many which can be unidirectional or Bi-directional. Relational model does not support all of these relationship. We have to establish these kind of relationship by associating a foreign key along with the primary key of another table.

- **Problem of subtypes**

It is difficult to represent Inheritance in relational model. There is no direct support for Inheritance in an RDBMS. There is no concept of base types and subtypes. As a result of not supporting inheritance RDBMS cannot support polymorphism also.

- **Problems of data navigation**

In an object model we try to access information through method calls which will select data piece wise, but this is not the right way to access data from database as there will be more trips to the database for a single query. In relational model data can be accessed in large chunks.

Approaches to ORM

- Java Data Base Connectivity (JDBC)
- Serialization
- Enterprise Java Beans (EJB)
- Object Oriented Data Base Management System (OODBMS)

Approaches to ORM

- We saw the mismatch between the Object model and the Relational model. Object relational mapping is a technique of mapping the data representation from an object model to a relational data model. There are alternatives for performing object relational mapping. Let us check them and see whether they are going to help us in efficiently persisting Objects.
- **Java Data Base Connectivity (JDBC):** Object/Relational mapping can be experienced through JDBC. The problem with approach is that it is tedious and requires lot of code. Also difficult to represent associations between objects.
- **Serialization.** Using Java serialization we can write application state to a file.
- **Enterprise Java Beans (EJB):** EJBs especially Container Manager Persistent (CMP) Entity beans help us to achieve some transparency with respect to database. But even EJB falls short of a better solution as the implementation of CMP is vendor specific & it does not allow us to perform some complex queries. EJBs are also heavy as they need to be run in a managed environment.
- **Object Oriented Data Base Management System (OODBMS):**There are some databases which support Object modeling, but they have not become that popular to catch the attention of the developers as the relational model. So for the developer, due to the existence of a large applications which use Relational model and the non popularity of the Object oriented databases, even OODBMS is not a viable solution currently.

The preferred solution for ORM

Use an Object-Relational Mapping System such as Hibernate that provides:

- An API for performing basic CRUD operations on objects of persistent classes
- A language or API for specifying queries that refer to classes and properties of classes
- A facility for specifying mapping metadata

Hibernate is a framework that provides tools for object relational mapping (ORM). Java Applications can use **Hibernate framework** as the persistence layer for storing and retrieving plain old Java objects (POJOs) to/from a relational database

Note:CRUD stands for Create Read Update Delete.

Advantages of ORM

- **Productivity** - As hibernate hides a lot of details of database activities it also helps him/her by allowing to concentrate on business concerns which speeds up the development. (Bauer and King, 2006).
- **Maintainability -** ORM reduces the number of lines of code. Lesser the Lines Of Code, easier it will be to maintain.
- **Performance** - Different Databases has to be tuned differently; it is difficult to hard code this for a variety of databases. ORM takes care of these concerns and the developer need not have to worry much.
- **Vendor Independence** - ORM supports portability. It works for different kinds of Databases.

Summary

In this module you were able to

- Define Persistence
- Identify the problems with Object/Relational mismatch
- Describe Alternatives for persistence management
- Define Object / Relational Mapping (ORM) in detail

Thank You