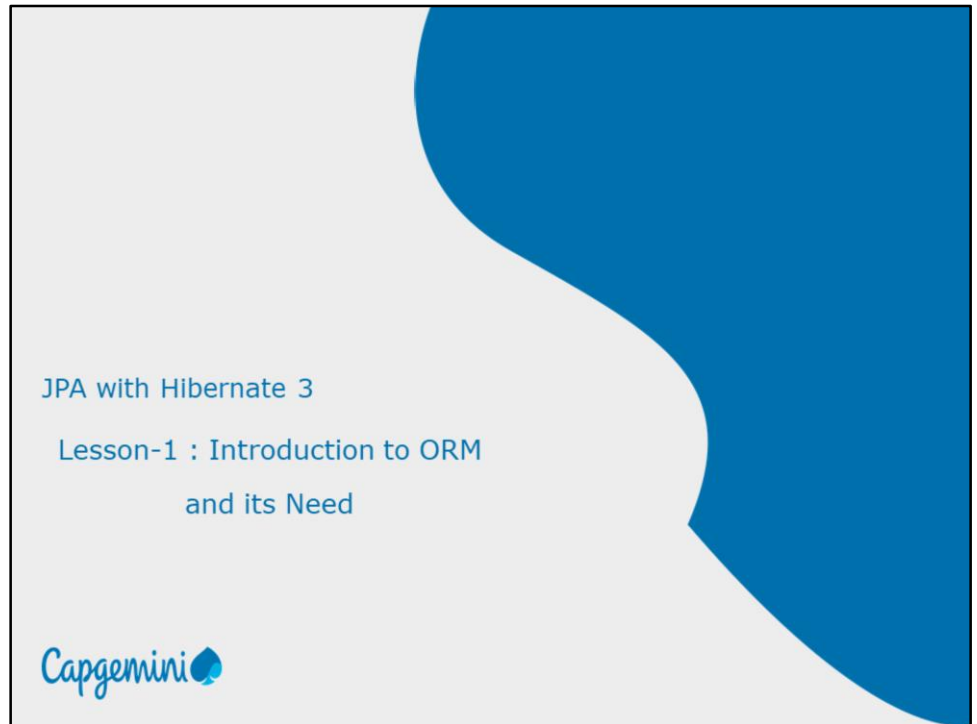


Instructor Notes:

Add instructor notes here.



Instructor Notes:**Lesson Objectives**

After completing this lesson, participants will be able to understand:

- Persistence and its benefits
- Object-relational Impedance Mismatch
- Object/Relational Mapping
- ORM and its need
- JPA and its benefits



Instructor Notes:

1.1: Persistence and its benefits

What is Object Persistence?

Persistence means to make application's data to outlive the applications process.
In Java terms, the objects to live beyond the scope of the JVM so that the same state is available later.

```
class User {  
    String name;  
    int salary;  
}
```

Name	Salary
Amit	20000

Object World

Relational - World

The above diagram depicts mapping of object state into database table columns. To do so, traditionally, we rely on JDBC API, which allows developers to save application data into database, however conversion is required from object format to database table format which un-necessarily increases line of code .

However, there are lot of challenges and mismatch in data processing in these two models. In addition, if database changes, then developer need to make modification in the configuration which is database specific.

So to shorten the development time, and to save application object directly into database, there was need to reinvent the approach of mapping object and relational model.

Instructor Notes:

1.2: Object-Relation Impedance Mismatch

Object-Relation Impedance Mismatch

'Object-relational Impedance Mismatch' means that object models and relational models do not work very well together.

RDBMSs represent data in a tabular format, whereas object-oriented languages, such as Java, represent it as an interconnected graph of objects.

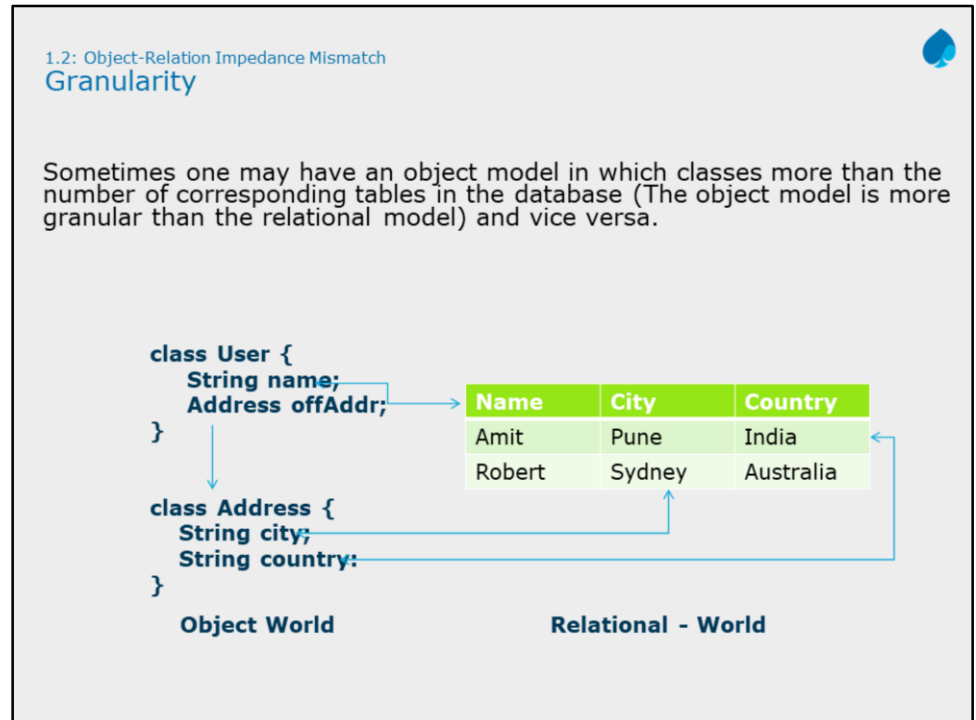
Loading and storing graphs of objects using a tabular relational database exposes us to following mismatch problems...

- Granularity
- Inheritance (subtypes)
- Identity

The above slide demonstrates on the challenges of mapping objects and database relations.

Instructor Notes:

Discuss the scenarios
real world scenarios to
discuss the Granularity



Above diagram shows how data of two entities are persisted into one relation (table)

Conversion of the java datatypes to underlying database types is done by ORM automatically.

Instructor Notes:

Add instructor notes here.

1.2: Object-Relation Impedance Mismatch
Subtypes (inheritance)

Inheritance is a natural paradigm in object-oriented programming languages. However, RDBMSs do not define anything similar on the whole.

The diagram illustrates the difference in how inheritance is handled between object-oriented programming and relational databases. On the left, labeled 'Object World', a box labeled 'User' is connected by a vertical line to a horizontal line, which then branches to two boxes labeled 'Customer' and 'Employee', representing inheritance. On the right, labeled 'Relational - World', the same structure is shown, but the 'User' box is not connected to the 'Customer' and 'Employee' boxes. Instead, there is a red circle with a diagonal slash (a prohibition sign) over the area where the connection would be, indicating that inheritance is not supported in the relational world.

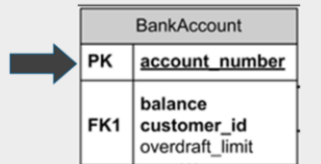
Instructor Notes:

Add instructor notes here.

1.2: Object-Relation Impedance Mismatch Identity



An RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity ($a == b$) and object equality ($a.equals(b)$).



Relational - World

$a == b$
 $a.equals(b)$

Object World

Instructor Notes:

Add instructor notes here.

1.2: Object-Relation Impedance Mismatch

Association

Linking between tables in database is done in different way as compared to linking between classes in an application.

Product

ProdID
Name
Price
SupplierID

Supplier

SupplierID
Name
Address
Phone

Relational - World

```
class Product{  
    -----  
    Supplier[] supplierList;  
}  
  
class Supplier {  
    -----  
}
```

Object World

Instructor Notes:

Add instructor notes here.

1.3: Object-Relation Mapping
Introducing ORM



To remove the impedance mismatch problem, map the data representations in an object model (Java Types) to relational model (SQL Types). This process is called as ORM.

Storing object-oriented entities in a relational database is often not a simple task and requires a great deal of repetitive code along with conversion between data types.

Object-relational mapper, or O/RM, were created to solve this problem. An O/RM persists entities in and retrieves entities from relational databases without the programmer having to write SQL statements and translate entity properties to statement parameters and result set columns to entity properties.

Instructor Notes:

Compare the given code snippet with JDBC insert task. Do mention about much of the boilerplate code now goes behind the screen like:

1. Obtaining connection
2. Creating statement
3. Converting Object types to RDBMS types
4. Closing opened resources

1.3: Object-Relation Mapping An ORM Solution



It consists of:

- An API, to perform CRUD operations on objects of persistent classes
- A language to specify queries that refer to classes and properties of classes
- A facility, to specify mapping metadata
- A technique, for the ORM implementation to interact with transactional objects to perform dirty checking. Lazy association, fetching, and other optimization functions.

```
public void addProduct(Product product)
{

    this.getCurrentTransaction().persist(product);
}
```

Dirty Checking:

A dirty checking feature avoids unnecessary database write actions by performing SQL updates only on the modified fields of persistent objects. For example, If you modify salary of employee on object model, only salary field will be updated instead of updating entire employee object.

Lazy association fetching:

Lazy fetching decides whether to load child objects while loading the Parent Object. For example, Consider department entity consist of many employees, and someone query to fetch department details, ORM fetches only department details and defers loading employees. This will be done, when one request details of employees working in that department.

Instructor Notes:

Add instructor notes here.

1.4: ORM and its need

Why ORM?



It “shields” developers from “messy” SQL

ORM tools allows developers to focus on the business logic of the application rather than repetitive CRUD (Create Read Update Delete) logic.

Some of the benefits of ORM are:

- Productivity
- Maintainability
- Performance
- Vendor independence

Instructor Notes:

Add instructor notes here.

1.5: JPA and its benefits

Introduction to Java Persistence API



Developed as part of Java Specification Request (JSR) 317

- Original goal to simplify EJB CMP entity beans

Simplifies the development of Java EE and Java SE applications using data persistence

JPA standardized the ORM persistence technology for Java developers.

Usable both within Java SE environments as well as Java EE

- POJO based
- Works with XML descriptors and annotations

JPA is just an specification from Sun, which is released under JEE 5 specification. JPA standardized the ORM persistence technology for Java developers. JPA is not a product and can't be used as it is for persistence. It needs an ORM implementation to work and persist the Java Objects. ORM frameworks that can be used with JPA are Hibernate, Toplink, Open JPA etc.

The Java Persistence API (JPA) is one approach to ORM. Via JPA the developer can map, store, update and retrieve data from relational databases to Java Objects and vice versa, JPA permits the developer to work directly with objects rather than with SQL statements. JPA is a specification and several implementations are available

Instructor Notes:

JPA not only provides database-independence but it also gives us a flexibility to have ORM implementer independence.

1.5: JPA and its benefits
What is JPA?

JPA is not a product and can't be used as it is for persistence. JPA needs an ORM implementation to work and persist the Java Objects. ORM frameworks that can be used with JPA are Hibernate, TopLink, Open JPA etc.

```
graph LR; A[Java Application] --> B[JPA]; B --> C[Hibernate]; C --> D[(DATABASE)]
```

The diagram illustrates the persistence flow: a Java Application (dark blue chevron) connects to JPA (purple chevron), which connects to Hibernate (red chevron), which finally connects to a DATABASE (green cylinder). The labels 'Java Application', 'JPA', and 'Hibernate' are inside their respective chevrons, while 'DATABASE' is inside the cylinder.

JPA is not the first attempt to create an ORM solution in Java. Before JPA, there were Java Data Objects (JDO) and Enterprise JavaBeans (EJB). JDO used to be popular, but seems to have run out of steam.

EJB, up to version 2.1, was overly complex and hard to use, harder than losing weight. EJB 3.0 simplifies things a lot and even uses JPA as its persistence mechanism. In short, JPA has started as part of EJB 3.0. However, since people want to use JPA without an EJB container, JPA has become an independent specification.

JPA is merely a specification, i.e. a document. In order for it to be useful, it needs a reference implementation, which is a Java API that implements the specification. There are numerous software packages that are JPA reference implementations. Hibernate, EclipseLink, and Apache OpenJPA are some of them.

Instructor Notes:

Add instructor notes here.

1.5: JPA and its benefits

Advantages of JPA



Simplified Persistence technology
ORM frameworks independence
▪ Any ORM framework can be used
Data can be saved in ORM way
Supported by industry leaders

Below listed are few advantages of JPA:

1. You don't need to create tables. In some cases, you don't even need to create a database. If any of your entity classes changes, the modern JPA provider can be configured to adapt the tables.
2. You don't need to write SQL statements, even though sometimes you may have to work with JPQL, the Java Persistence Query Language.
3. Changing databases, say from Oracle to MySQL, is a breeze

There are disadvantages too, but most of them are negligible.

1. JPA adds to the application's memory usage. Negligible in most cases.
2. JPA adds an extra layer to the application, making the system a bit slower than if it accesses the database through JDBC directly. However, the performance penalty is small that it is considered negligible.

Instructor Notes:

Add instructor notes here.

1.5: JPA and its benefits

Why JPA?



Impedance mismatch

- Object-oriented vs. relational

Java developers are not database developers

- Reduce the need for developers to know and fully understand database design, SQL, performance tuning
- Increase portability across database vendors and/or ORM frameworks

Increase performance by deferring to experts

- Potential decrease in database calls
- More efficient SQL statements

Instructor Notes:

Add instructor notes here.

Summary

In this lesson, you have learned about:

- Object relational mapping
- Object relational impedance mismatch
- ORM and its need
- JPA and its benefits



Instructor Notes:

Answers:

- 1.True
- 2.Option 1 and 2

Review Question

JPA provides database and ORM implementer independence.

- True/False

Which of the following are ORM implementers?

- Option 1: Hibernate
- Option 2: OpenJPA
- Option 3: JPA
- Option 4: TopLink

Which of the following condition/s indicates granularity impedance mismatch?

- Option 1: Having more classes and less tables
- Option 2: Having more tables and less classes
- Option 3: Having one table per class

