

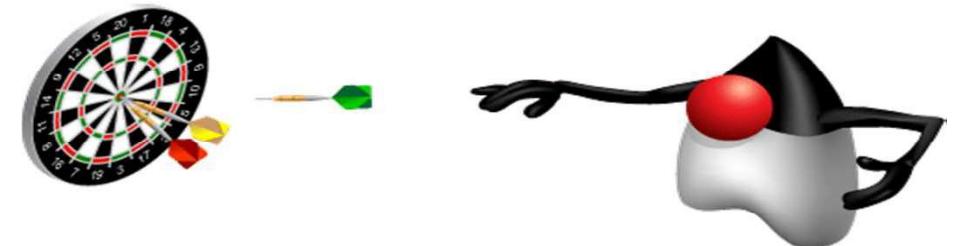


Developing Persistence Layer with JPA Entities

Objectives

After completing this lesson, you should be able to do the following:

- What are JPA Entities?
- Domain Modeling with JPA
- Creating an Entity (a POJO with annotations)
- Specifying Object Relational (OR) Mapping
- Mapping Relationships between Entities
- Inheritance Mapping Strategy (Single Table, Joined Subclass)



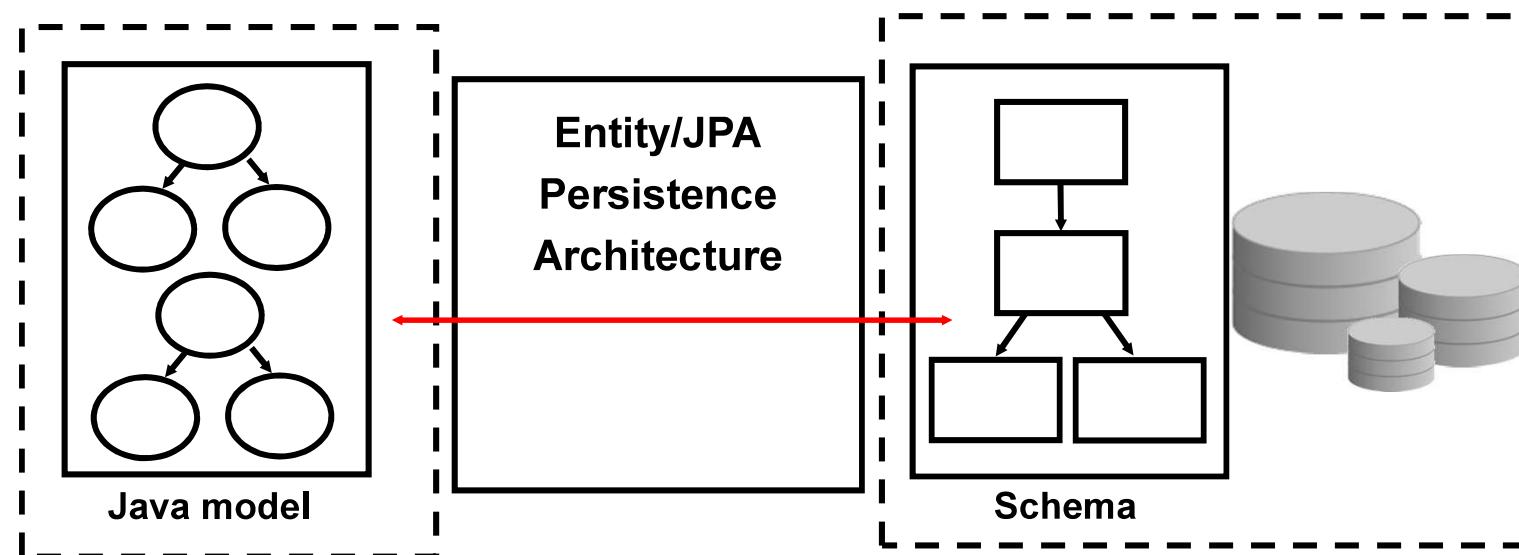
What Is Persistence?

The persistence layer provides mapping between objects and database tables.

- This layer:
 - Enables portability across databases and schemas
 - Supports read, write, and caching capabilities
 - Protects developers from database issues
 - Facilitates change and maintenance
 - Should be used in any application that has an object model

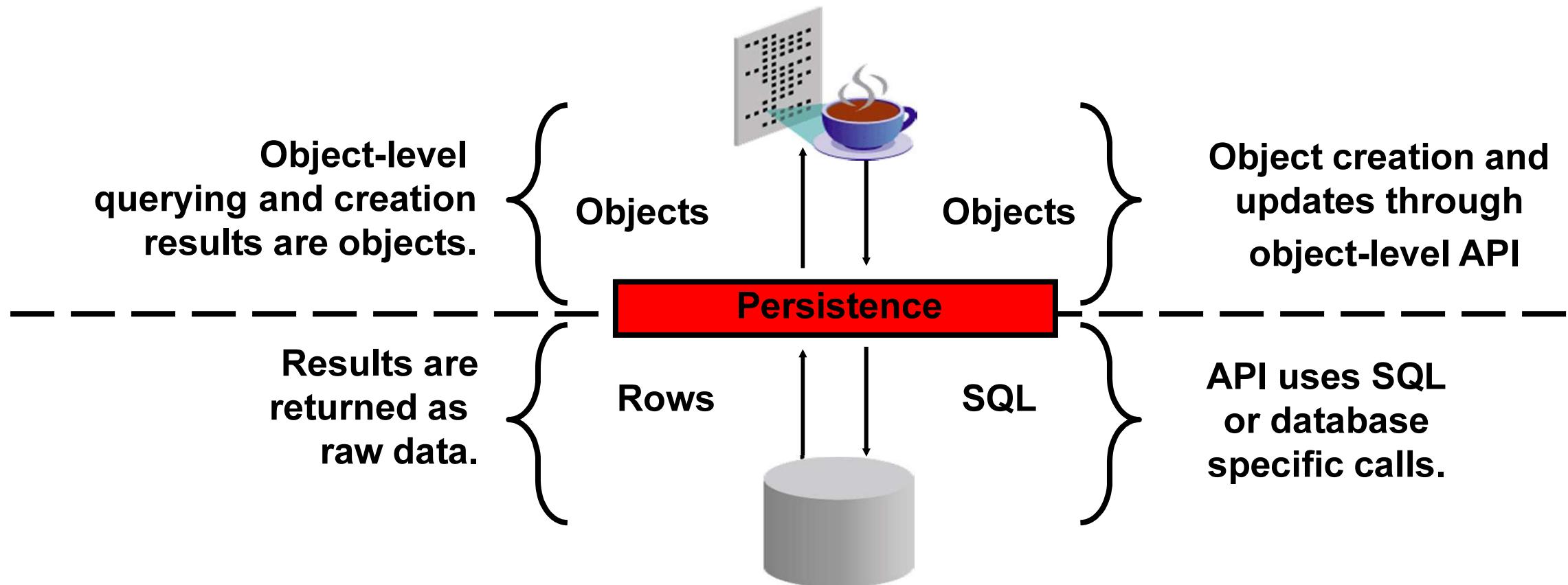
Persistence: Overview

- Mapping relational database objects to Java objects enables easy Java EE application development.
- Frameworks such as EJB 3.0 JPA provide this object-relational mapping.



Persistence Layer

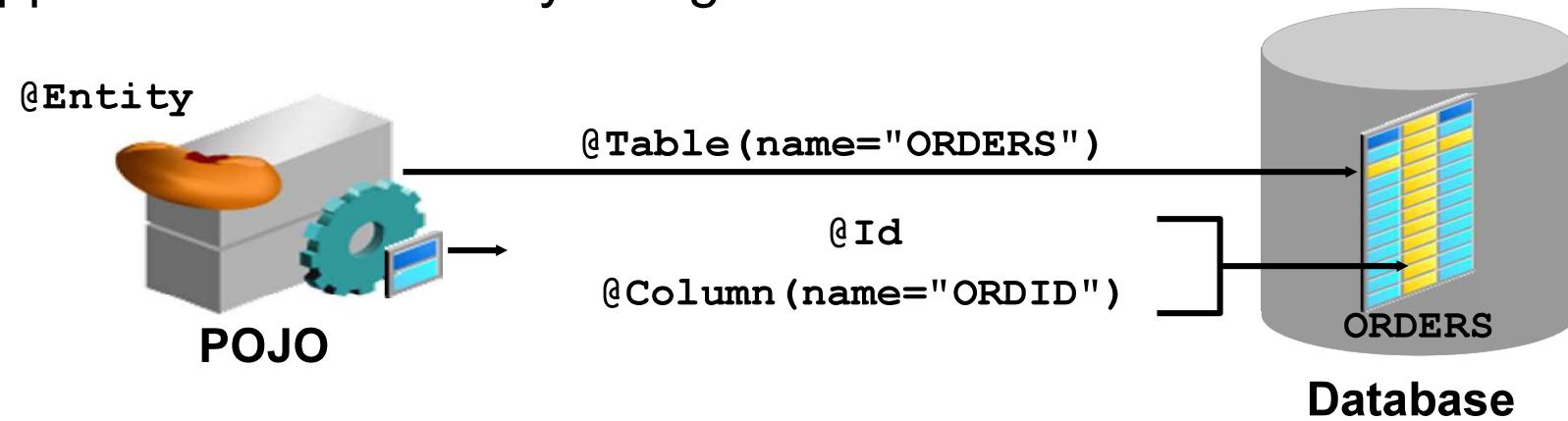
A persistence layer abstracts persistence details from the application layer.



What Are JPA Entities?

A Java Persistence API (JPA) entity is:

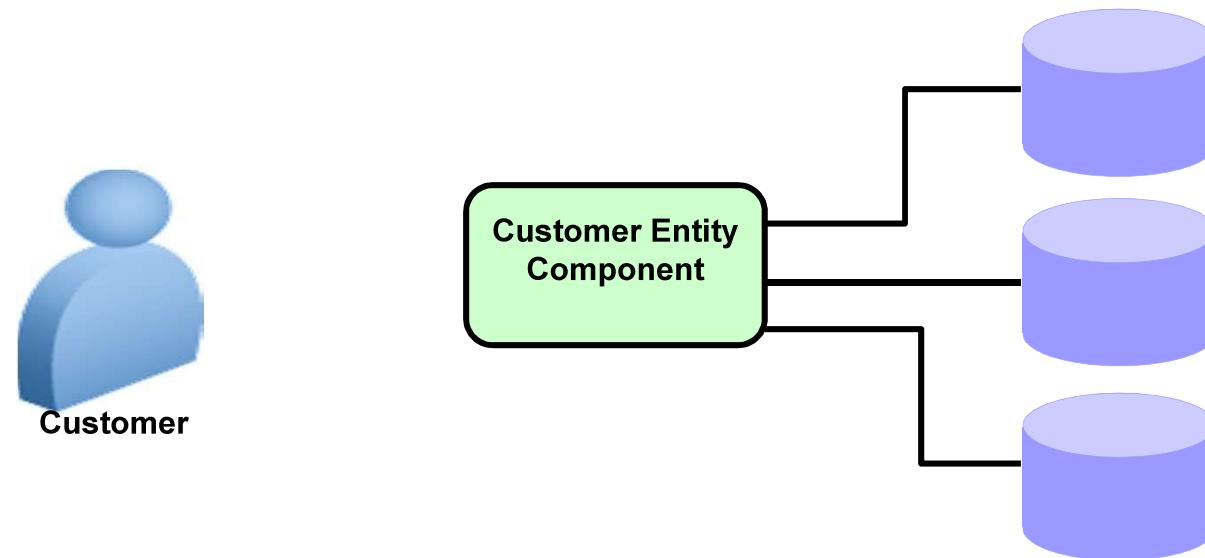
- A lightweight object that manages persistent data
- Defined as a Plain Old Java Object (POJO) marked with the `@Entity` annotation (no interfaces required)
- Not required to implement interfaces
- Mapped to a database by using annotations



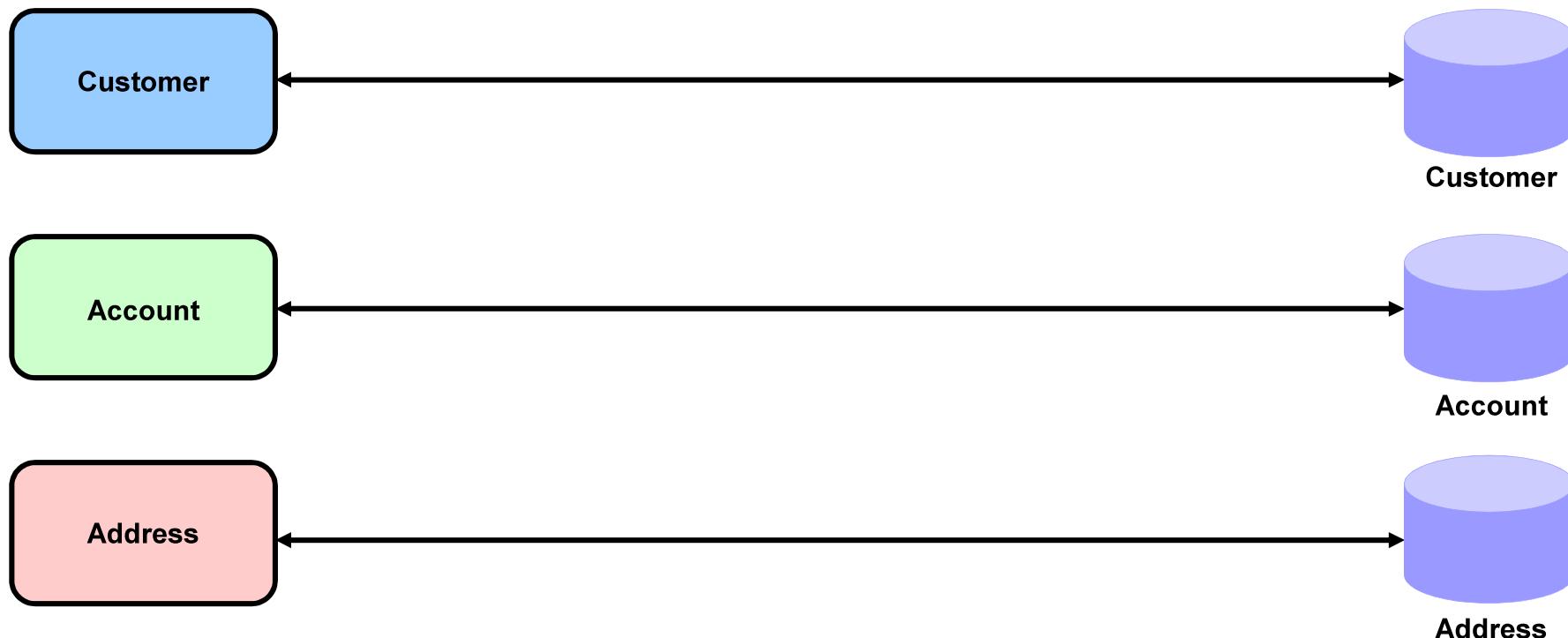
Object Relational Mapping

Object-relational mapping (ORM) software:

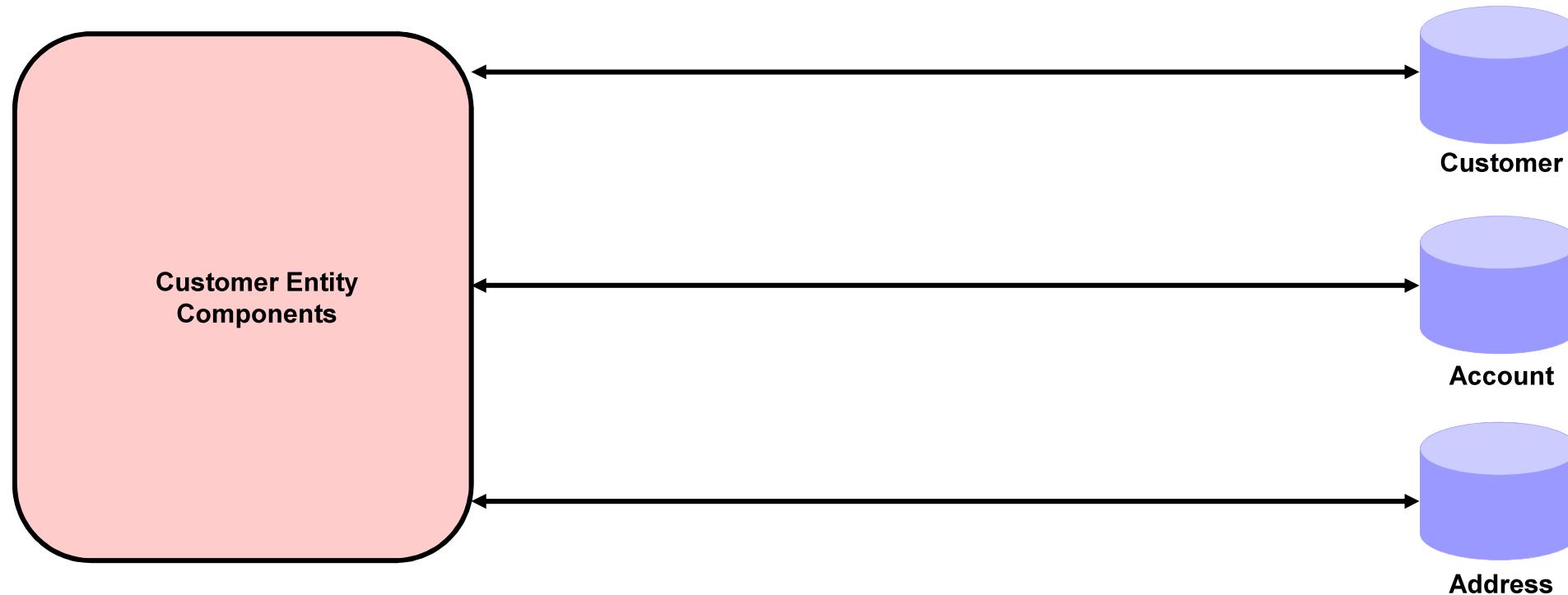
- Provides an object-oriented view of the database
- Examples include EclipseLink and Hibernate



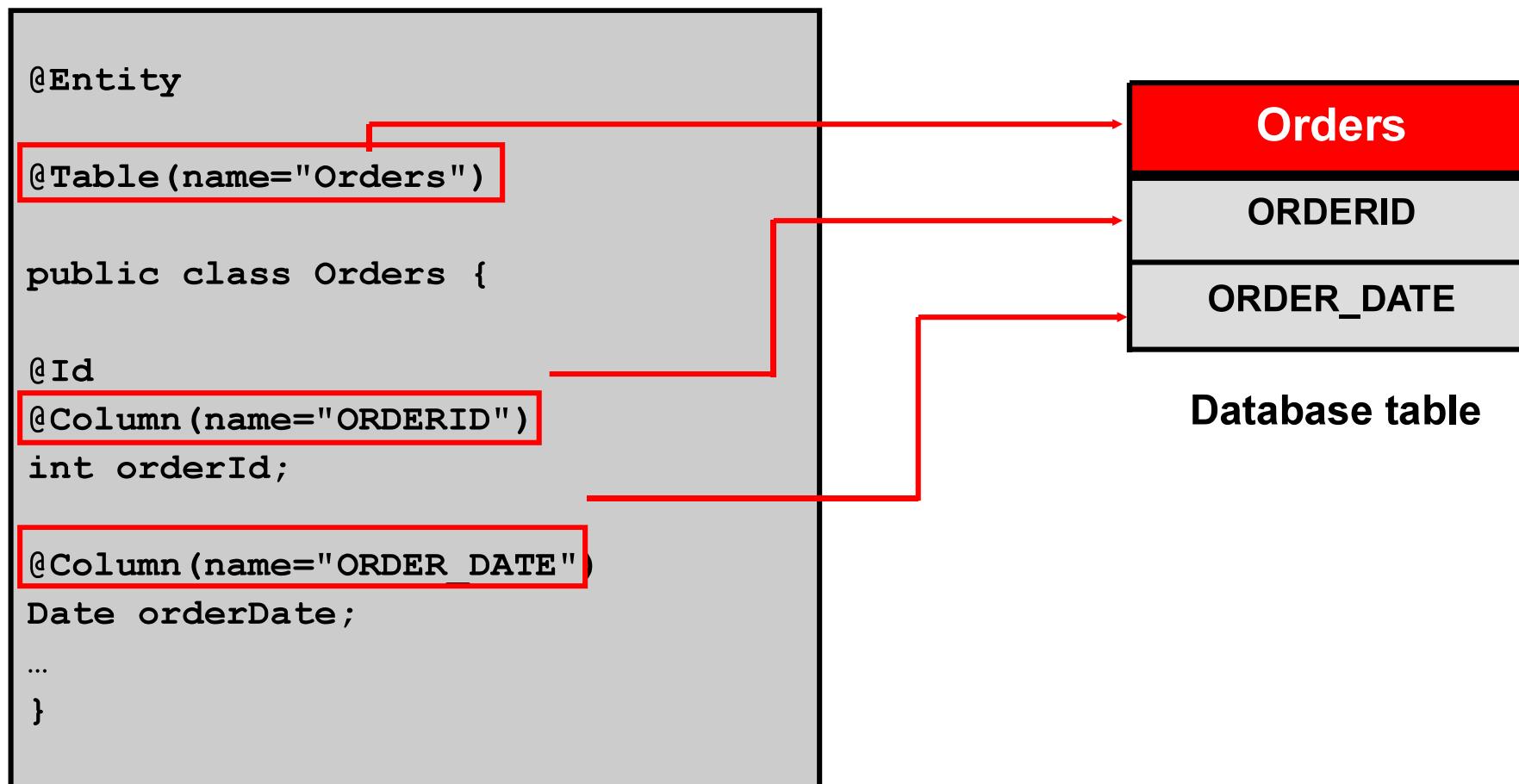
Normalized Data Mapping



Use of an Entity Component Across a Set of Database



JPA entity

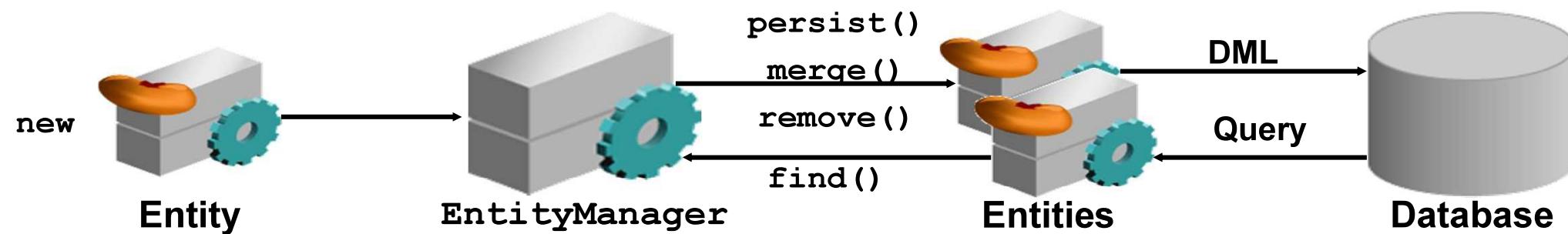


Domain Modeling with Entities

- Entities support standard object-oriented domain modeling techniques:
 - Inheritance
 - Encapsulation
 - Polymorphic relationships
- Entities can be created with the `new` operator.

Managing Persistence of Entities

- The life cycle of an entity is managed by using the EntityManager interface, which is part of the JPA.
- An entity can be created by using:
 - The new operator (creates detached instance)
 - The EntityManager Query API (synchronized with the database)
- An entity is inserted, updated, or deleted from a database through the EntityManager API.



Declaring an Entity

- Declare a new Java class with a no-arg constructor.
- Annotate it with `@Entity`.
- Add fields corresponding to each database column:
 - Add setter and getter methods.
 - Use the `@Id` annotation on the primary key getter method.

```
@Entity           // annotation
public class Customer implements java.io.Serializable {
    private int customerID;
    private String name;

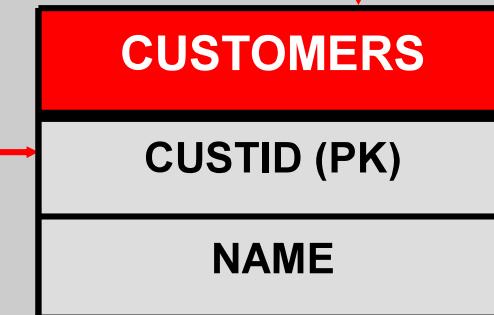
    public Customer() { ... } // no-arg constructor
    @Id             // annotation
    public int getCustomerID() { ... }
    public void setCustomerID(int id) { ... }
    public String getName() { ... }
    public void setName(String n) { ... }
}
```

Mapping Entities

Mapping of an entity to a database table is performed:

- By default
- Explicitly using annotations or in an XML deployment descriptor

```
@Entity  
@Table(name="CUSTOMERS")  
public class Customer implements java.io.Serializable {  
  
    @Id  
    @Column(name="CUSTID")  
    private int customerID;  
    private String name;  
    ...  
    public int getCustomerID() { ... }  
    public void setCustomerID(int id) { ... }  
    public String getName() { ... }  
    public void setName(String n) { ... }  
}
```



Quiz

An entity is a lightweight persistence domain object that represents:

1. A relational database
2. A table in a relational database
3. Entity beans in EJB 2.x specification
4. Persistence data in a file

Specifying Entity Identity

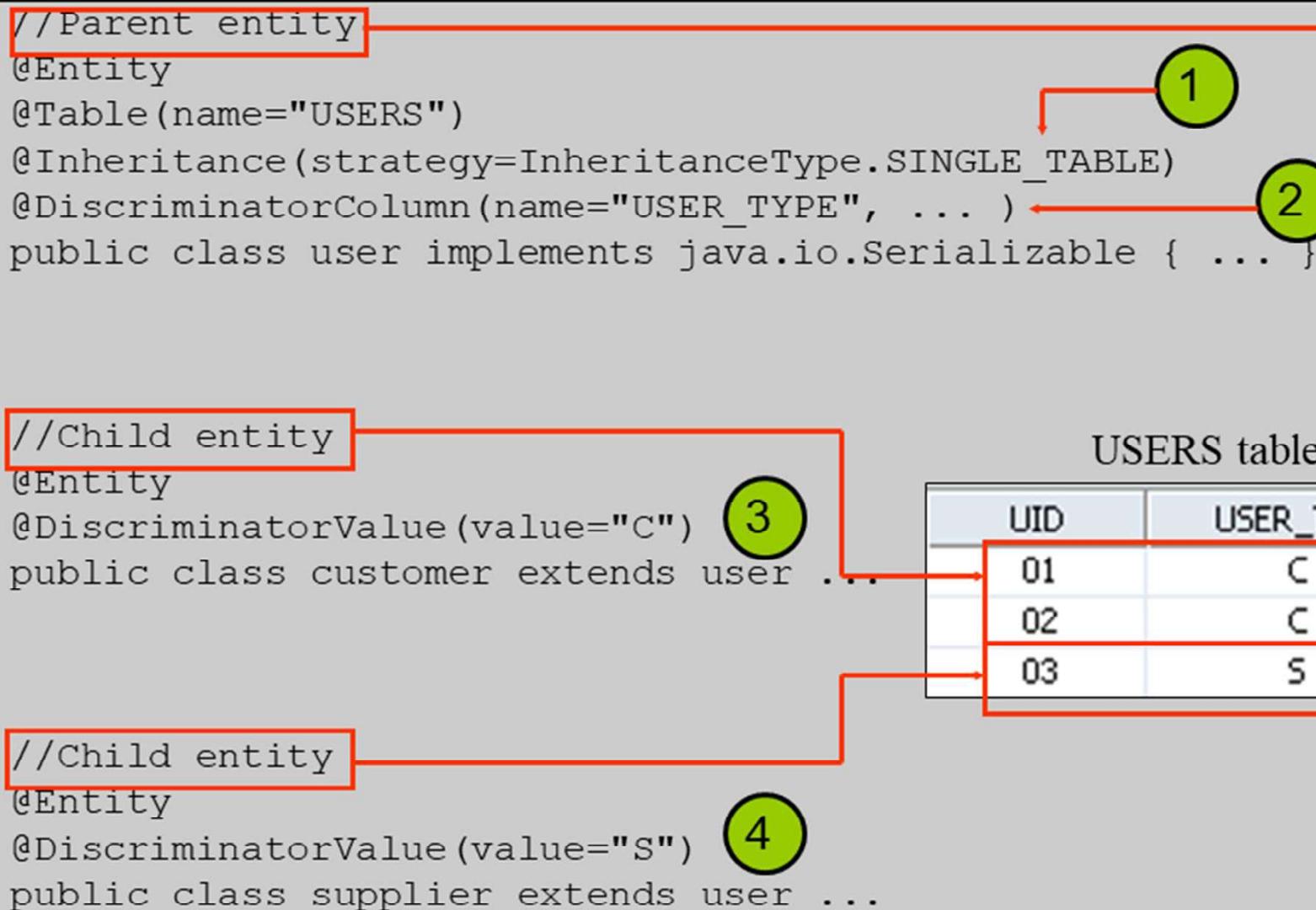
The identity of an entity can be specified by using:

- The `@Id` annotation
- The `@IdClass` annotation

Mapping Inheritance

- Entities can implement inheritance relationships.
- You can use three inheritance mapping strategies to map entity inheritance to database tables:
 - Single-table strategy
 - Joined-tables strategy
 - Table-per-class strategy
- Use the `@Inheritance` annotation.

Single-Table Strategy

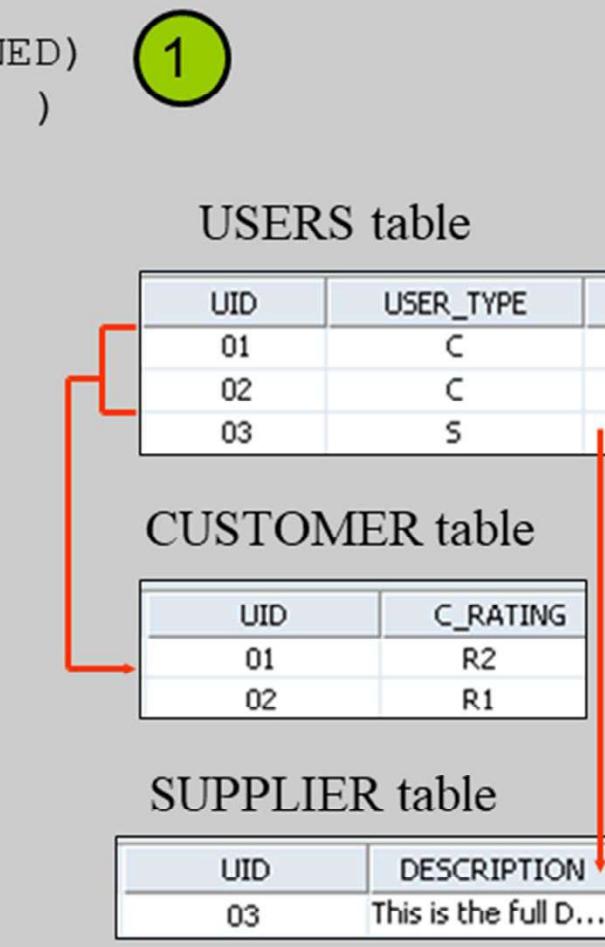


Joined-Tables Strategy

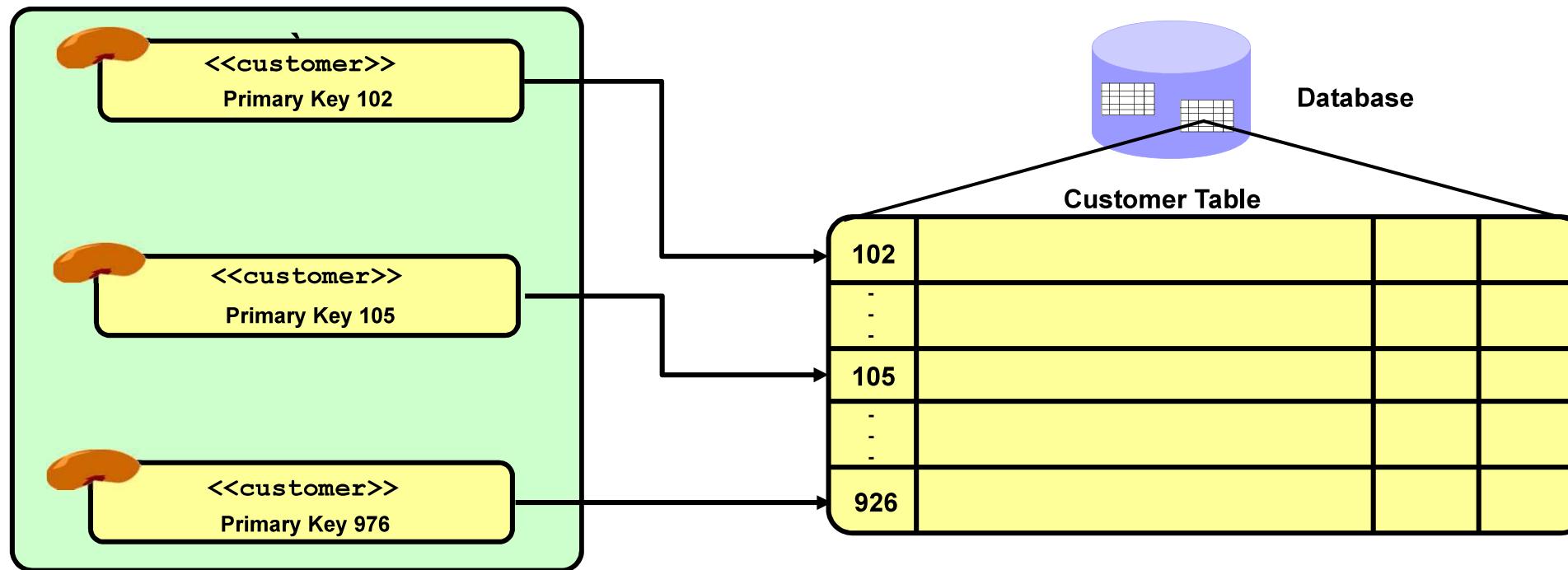
```
//Parent entity
@Entity
@Table(name="USERS")
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="USER_TYPE", ... )
public class user ...

//Child entity
@Entity
@Table(name="CUSTOMER")
@DiscriminatorValue(value="C")
@PrimaryKeyJoinColumn(name="UID")
public class customer extends user ...

//Child entity
@Entity
@Table(name="SUPPLIER")
@DiscriminatorValue(value="S")
@PrimaryKeyJoinColumn(name="UID")
public class supplier extends user ...
```



Entity Component Primary Key Association



Generating Primary Key Values

Use the `@GeneratedValue` annotation.

```
@Entity  
@Table(name="CUSTOMERS")  
public class Customer implements java.io.Serializable {  
    @Id  
    @SequenceGenerator(name = "CUSTOMER_SEQ_GEN",  
        sequenceName = "CUSTOMER_SEQ", initialValue = 1,  
        allocationSize = 1)  
    @GeneratedValue(strategy = GenerationType.SEQUENCE,  
        generator = "CUSTOMER_SEQ_GEN")  
    @Column(name = "CUSTID", nullable = false)  
    private Long customerId;  
    ...  
    public int getCustomerID() { ... }  
    ...  
}
```

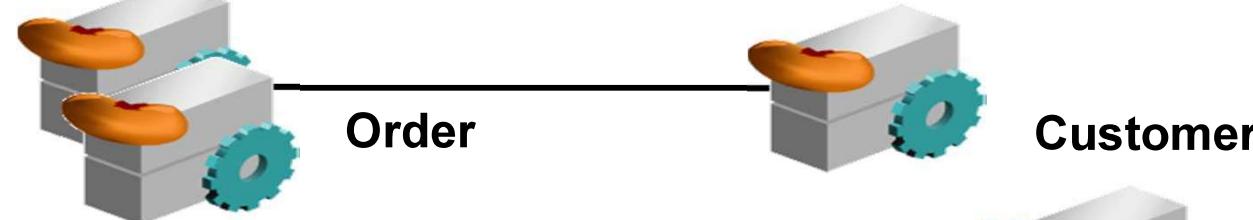
Mapping Relationships Between Entities

Annotations for entity relationships:

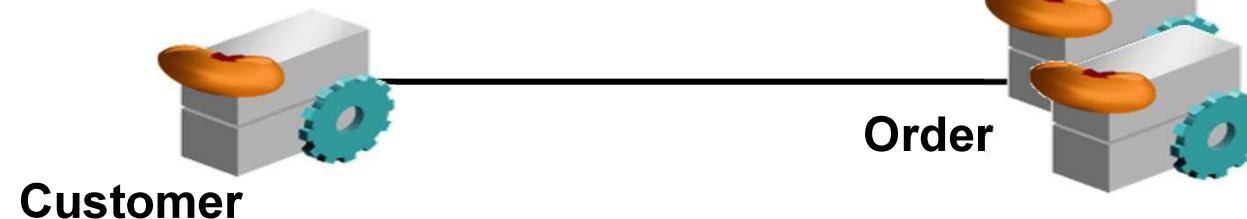
- @OneToOne



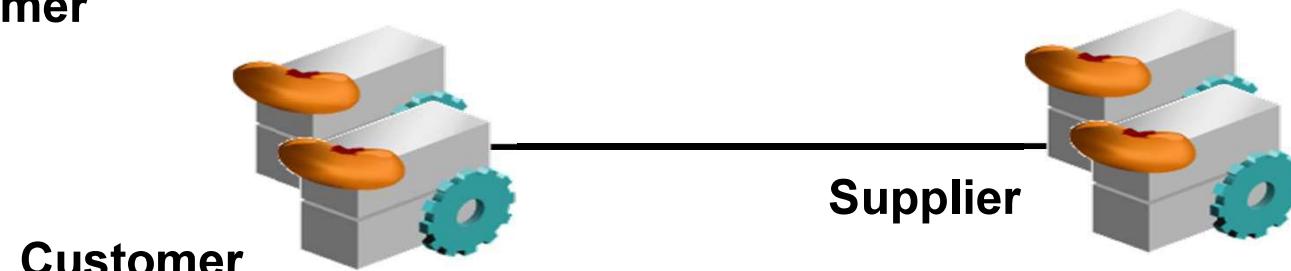
- @ManyToOne



- @OneToMany

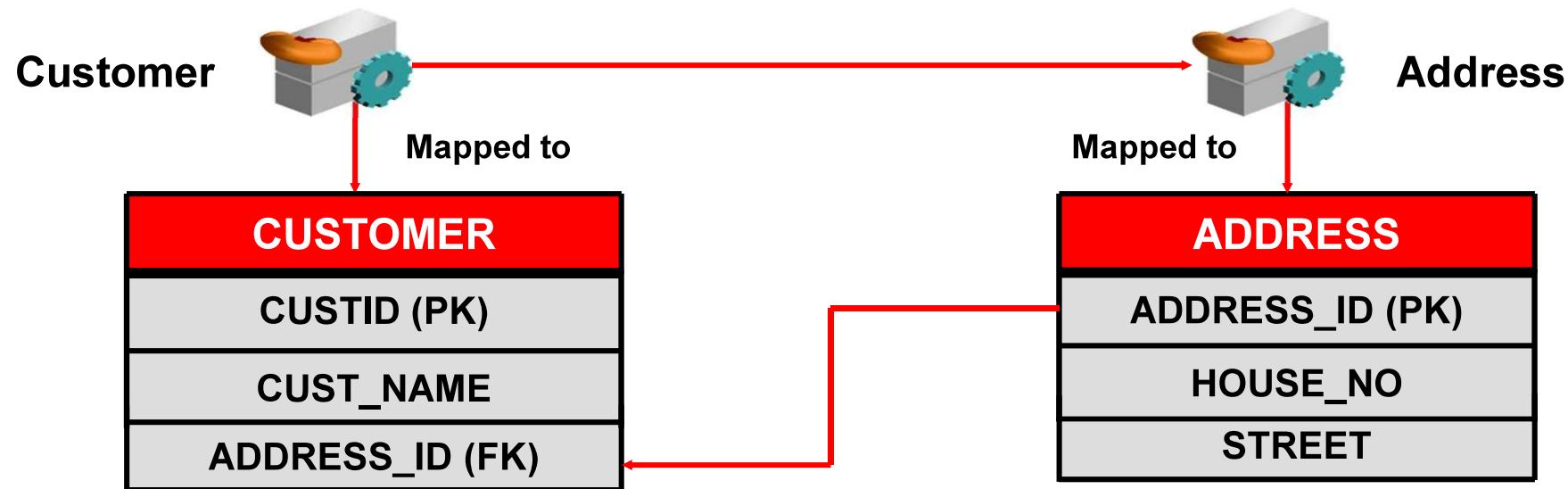


- @ManyToMany



Implementing One-to-One Relationships

- You can map one-to-one relationships by using the `@OneToOne` annotation.
- Depending on the foreign key location, the relationship can be implemented by using:
 - The `@JoinColumn` annotation
 - The `@PrimaryKeyJoinColumn` annotation



Implementing One-to-One Relationships

Example: Mapping a one-to one relationship between the Customer class and the Address class by using the @JoinColumn annotation

```
// In the Customer class:  
@Table(name="CUSTOMER")  
...  
@OneToOne  
@JoinColumn(name="ADDRESS_ID",  
           referencedColumnName="ADDRESS_ID")  
protected Address address;  
...  
  
// In the Address class:  
@Table(name="ADDRESS")  
...  
@Column(name="ADDRESS_ID")  
...
```

Implementing Many-to-One Relationships

- Mapping a many-to-one relationship:
 - Using the `@ManyToOne` annotation
 - Defines a single-valued association
- Example: Mapping an `Orders` class to a `Customer`

```
// In the Order class
@Entity
@Table(name="ORDER")
public class Order ... {
    ...
    @ManyToOne
    @JoinColumn(name="ORDERS_CUSTID_REF",
                referencedColumnName="CUSTID_PK", updatable=false)
    protected Customer customer;
    ...
}
```

Implementing One-to-Many Relationships

Mapping a one-to-many relationship by using the @OneToMany annotation.

```
//In the Customer class:  
@Table(name="CUSTOMER")  
...  
@OneToMany(mappedBy="customer")  
protected Set<Order> order;  
...  
  
// In the Order class:  
@Table(name="ORDER")  
...  
@ManyToOne  
@JoinColumn(name="ORDERS_CUSTID_REF",  
    referencedColumnName="CUSTID_PK", updatable=false)  
protected Customer customer;  
...
```

Implementing Many-to-Many Relationships

Mapping a many-to-many relationship by using the `@ManyToMany` annotation.

```
// In the Customer class:  
...  
@ManyToMany(cascade=PERSIST)  
@JoinTable(name="CUST_SUP",  
    joinColumns=  
        @JoinColumn(name="CUST_ID", referencedColumnName="CID"),  
    inverseJoinColumns=  
        @JoinColumn(name="SUP_ID", referencedColumnName="SID"))  
protected Set<Supplier> suppliers;  
...  
  
// In the Supplier class:  
...  
@ManyToMany(cascade=PERSIST, mappedBy="suppliers")  
protected Set<Customer> customers;  
...
```

Managing Entities

- Entities are managed by using the EntityManager API.
- EntityManager performs the following tasks for the entities:
 - Implements the object-relational mapping between Java objects and database
 - Performs the CRUD operations for the entities
 - Manages the life cycle of the entities
 - Manages transactions

Summary

In this lesson, you should have learned how to:

- What are JPA Entities?
- Domain Modeling with JPA
- Creating an Entity (a POJO with annotations)
- Specifying Object Relational (OR) Mapping
- Mapping Relationships between Entities
- Inheritance Mapping Strategy (Single Table, Joined Subclass)



Practice: Overview

These practice covers the following topics:

- Creating a simple entity bean by coding the bean
- Using the JDeveloper wizards to create a set of entity beans
- Creating and managing a session bean that provides client access to the entity beans
- Creating a test client to invoke the session bean