/ JAVA DB SCHOOL JPA

/ Recap

Data persistence

- Non-persistent
  - In memory (heap or stack)
- Persistent
  - Read/write to file
  - Binary serialization
  - XML
  - JSON
  - Database – relational or non-relational

Recap

- Relational database – JDBC
  - Establishing database connection
  - Creating SQL queries
  - Executing queries
  - Operating on query results

- @ActiveRecordEntity – a mechanism for persisting objects in the database

Recap

- The previous model requires a custom logic to manipulate the database objects
- Set of rules known by the developer

- Solution – @ActiveRecord a mechanism for persisting objects in the database
- Downside – custom implementation

/ ORM

Object Relational Mapping (ORM)

- Java – objects
- Database – relational model
- ORM – the process of translating the information between the objectual model and the relational one
- Purpose – data management through Java objects

Objectual model

- Characteristics
  - Identity – objects are distinguishable through their reference
  - Abstractization – the defined objects represent elements from the real world
  - Inheritance – common properties and behaviour can be extracted to a higher type
  - Encapsulation – hiding the internal logic and providing selective access
  - Polimorphism – an element can be represented in multiple ways

Relational model

- Data is represented through n-tuples
- Each n-tuple represents a row in a table
- The attributes that define the entity in an unique way are called primary keys

Objectual model + relational model

- Ideally each model attribute should be mapped should be mapped to a column of the table

- Identity – we will introduce an id that uniquely identifies both an object and a line from the table

ORM instruments (1)

- ORM – automatic persistence of data
- Translation between the objectual model and the relational model
- Software
    - Hibernate
    - Oracle TopLink
    - MyBatis
    - EclipseLink
    - OpenJPA
    - Apache Cayenne

ORM instruments (2)

- Advantages
  - Easy to use and understand
  - Reduces the implementation time
  - Reduces the amount of code and the error rate
  - The application is independent of the database management system
- Disadvantages
  - Lower performance that manually writing the SQL queries regarding big data processing
  - Ramp-up time

/ JPA

Java Persistence API (JPA)

- The high number of ORM instruments lead to the need of a standard
- JPA
  - Collection of classes and methods to persistently store the vast amounts of data into a database
  - It forms a bridge between object models (Java program) and relational models (database program)
  - Based on entities

Entities

- Are POJOs (Plain Old Java Objects)
- Conventions:
  - Default constructor
  - Getters and setters for non-Boolean properties
  - Setter and is methods for Boolean properties

- Can be defined through XML or Annotations

Entities annotation (1)

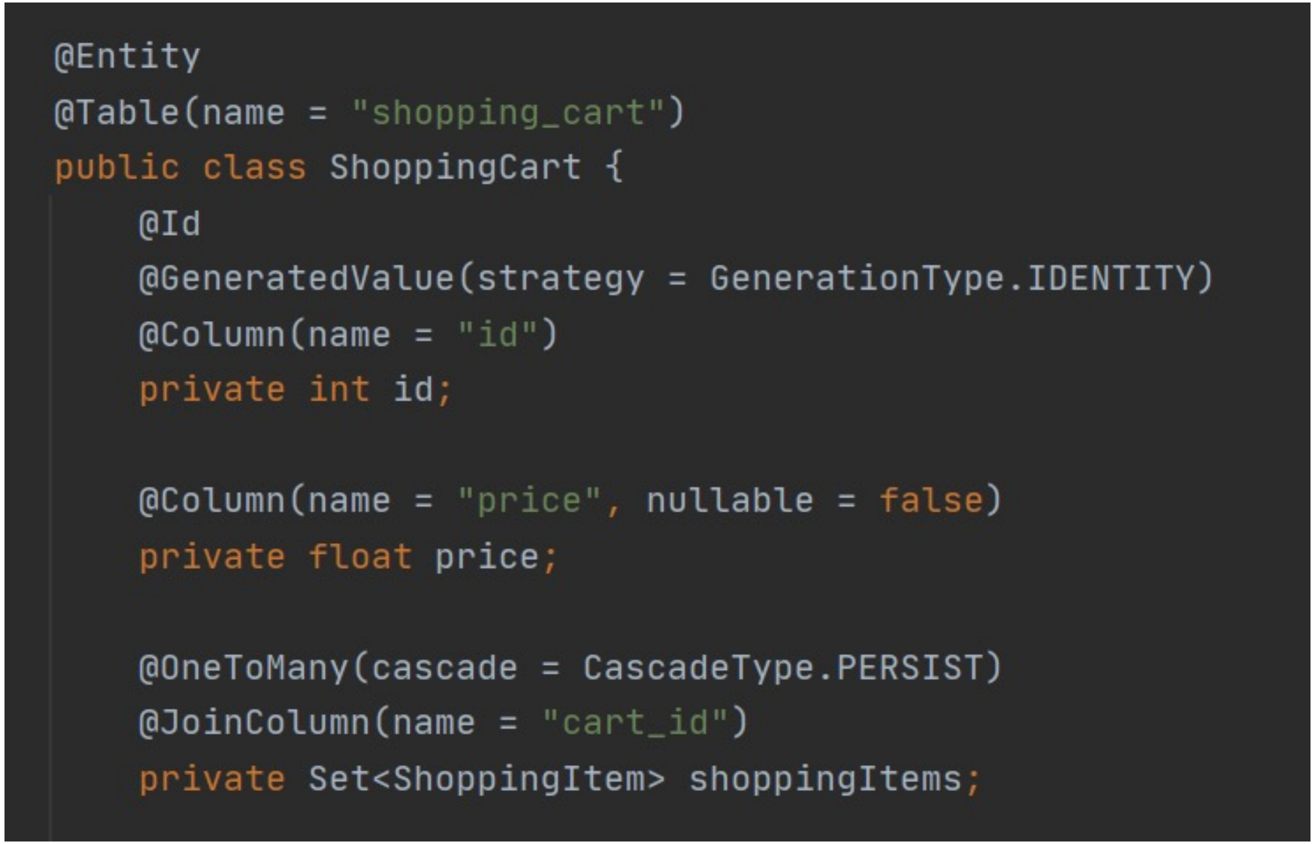
- @Entity - declare the class as entity or a table
- @Table - declare table name
- @Id - specifies the property uses for identity (primary key of a table)
- @GeneratedValue - specifies how the identity attribute can be initialized such as automatic, manual, or value taken from sequence table
- @Transient - specifies the property won’t persist
- @Column - specifies column or attribute for persistence property
- @UniqueConstraint - used to specify a unique constraint on the field

Entities annotation (2)

- @JoinColumn - specify an entity association or entity collection. This is used in many-to-one and one-to-many associations
- @ManyToMany - define a many-to-many relationship between the join tables
- @ManyToOne - define a many-to-one relationship between the join tables
- @OneToMany - define a one-to-many relationship between the join tables
- @OneToOne - define a one-to-one relationship between the join tables

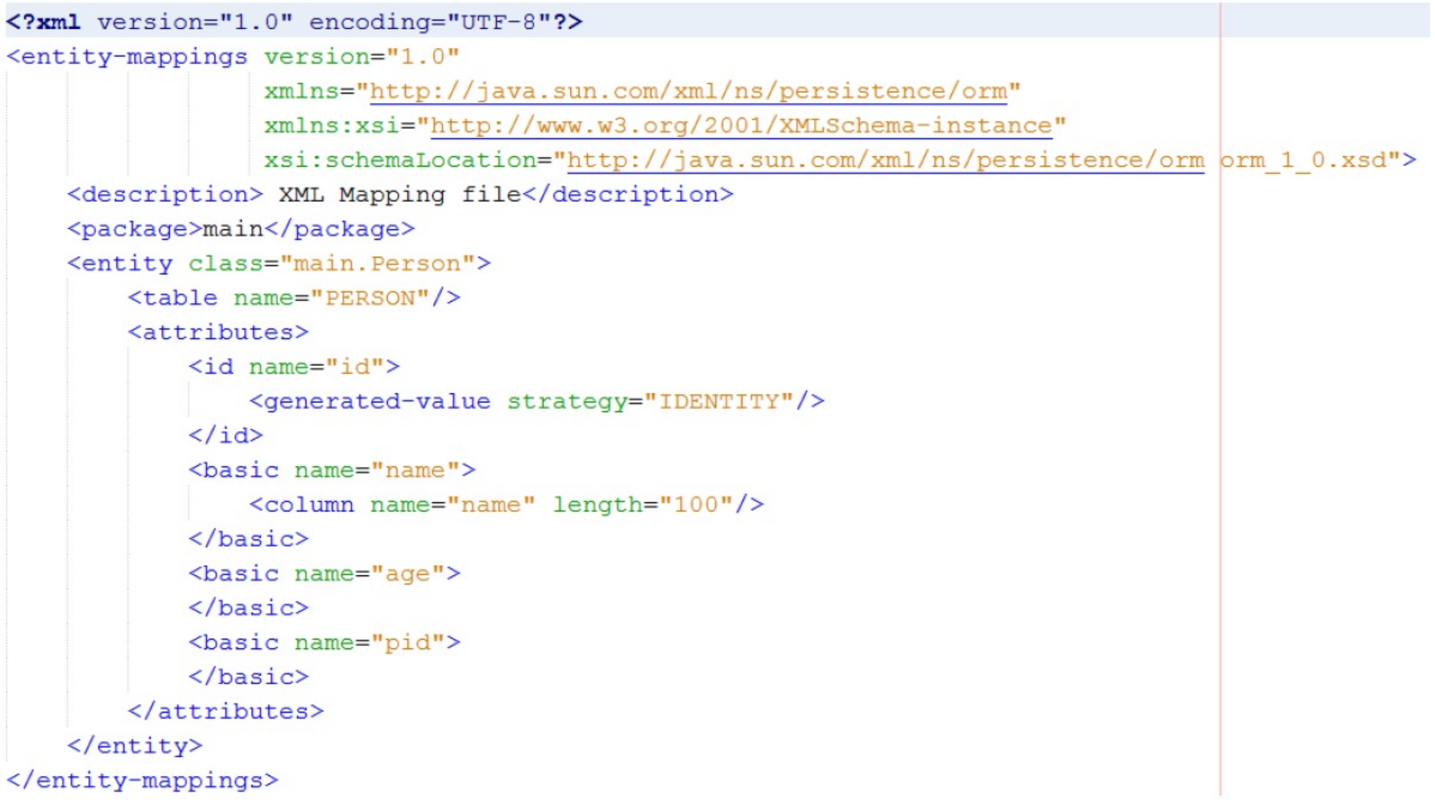
Entities annotation (3)

- Cascading – the action on the target entity will be applied to the associated entity
- javax.persistence.CascadeType
  - ALL
  - PERSIST
  - MERGE
  - REMOVE
  - REFRESH
  - DETACH



```java
@Entity
@Table(name = "shopping_cart")
public class ShoppingCart {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "price", nullable = false)
    private float price;

    @OneToMany(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "cart_id")
    private Set<ShoppingItem> shoppingItems;
```

Entities XML



```xml
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings version="1.0"
                 xmlns="http://java.sun.com/xml/ns/persistence/orm"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm orm_1_0.xsd">
    <description> XML Mapping file</description>
    <package>main</package>
    <entity class="main.Person">
        <table name="PERSON"/>
        <attributes>
            <id name="id">
                <generated-value strategy="IDENTITY"/>
            </id>
            <basic name="name">
                <column name="name" length="100"/>
            </basic>
            <basic name="age">
            </basic>
            <basic name="pid">
            </basic>
        </attributes>
    </entity>
</entity-mappings>
```

/ Hibernate

Hibernate (1)

- Hibernate is a Java framework that simplifies the development of Java application to interact with the database
- It is an open source, lightweight, ORM tool
- Implements the specifications of JPA for data persistence

Hibernate (2)

- Advantages
  - Open Source and Lightweight
  - Fast – it uses 2 levels of cache
  - Database independent query – uses Hibernate Query Language, the object-oriented version of SQL
  - Automatic table creation
  - Simplifies complex joins

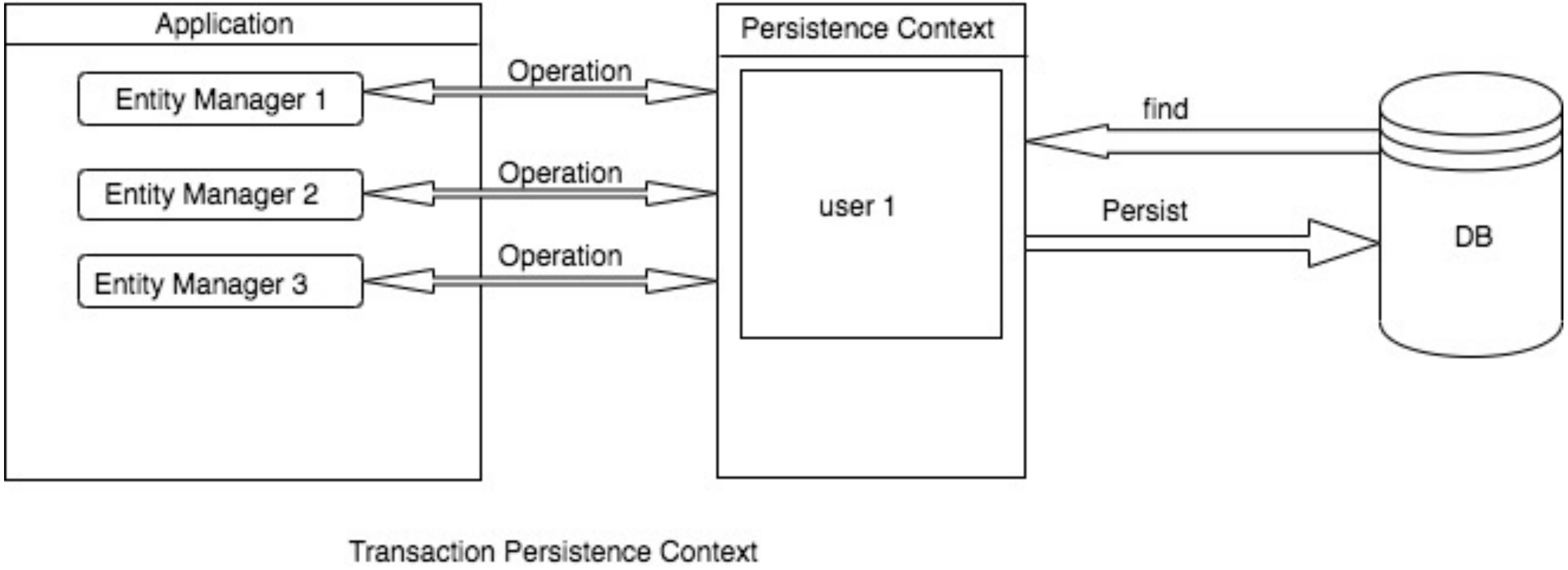
Persistence Context

- A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance

- Within the persistence context, the entity instances and their lifecycle are managed

- The EntityManager API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities

- Persistence contexts are available in two types:
    - Transaction-scoped persistence context
    - Extended-scoped persistence context



Reference: https://www.baeldung.com/jpa-hibernate-persistence-context

Transaction-scoped persistence context

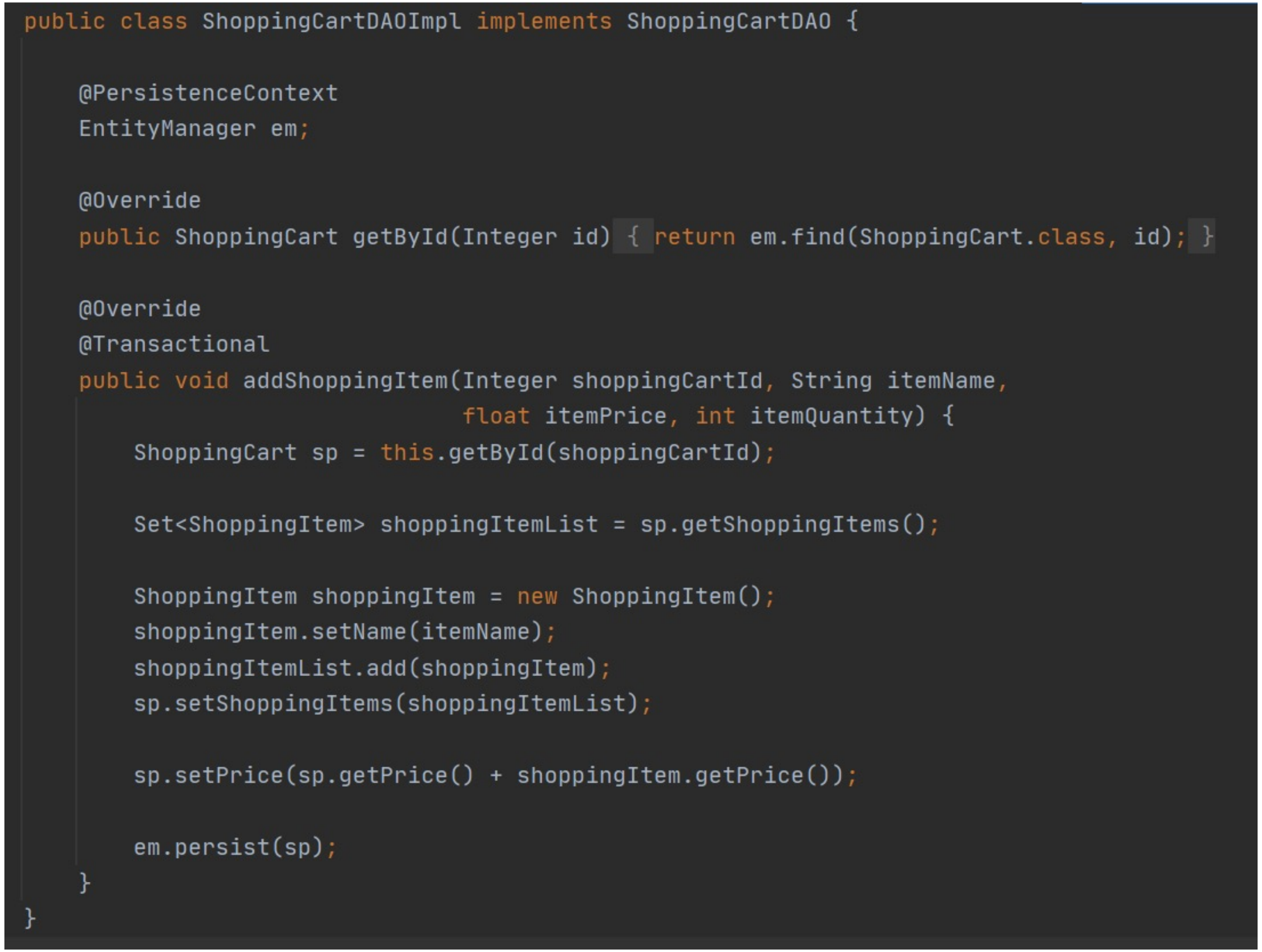
- Bound to a transaction
- As soon as the transaction finishes, the entities present in the persistence context will be flushed into persistent storage
- Is the default persistence context type

Persistence context annotations

- @EnableTransactionManagement
  - Enables Spring's annotation-driven transaction management capability
  - Is added on the main class, that also has @SpringBootApplication

- @Transactional
  - Annotation that is added to all methods that change the database structure (insert, update, delete)

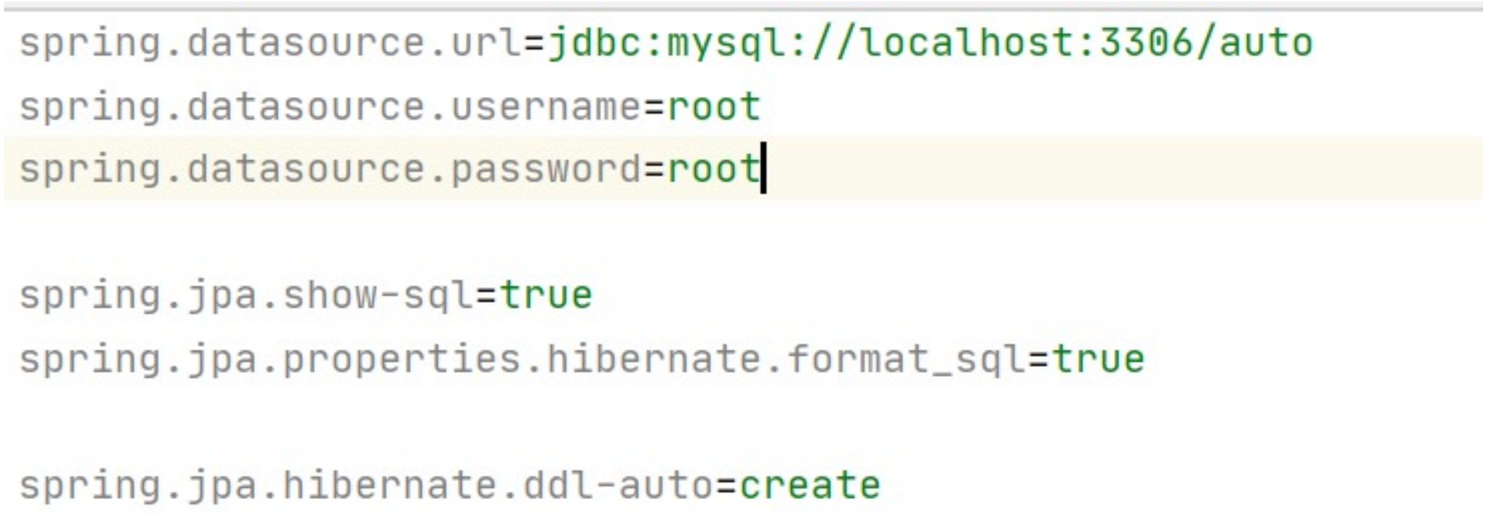
- @PersistenceContext
  - To inject the EntityManager



```java
public class ShoppingCartDAOImpl implements ShoppingCartDAO {

    @PersistenceContext
    EntityManager em;

    @Override
    public ShoppingCart getById(Integer id) { return em.find(ShoppingCart.class, id); }

    @Override
    @Transactional
    public void addShoppingItem(Integer shoppingCartId, String itemName,
                                float itemPrice, int itemQuantity) {
        ShoppingCart sp = this.getById(shoppingCartId);

        Set<ShoppingItem> shoppingItemList = sp.getShoppingItems();

        ShoppingItem shoppingItem = new ShoppingItem();
        shoppingItem.setName(itemName);
        shoppingItemList.add(shoppingItem);
        sp.setShoppingItems(shoppingItemList);

        sp.setPrice(sp.getPrice() + shoppingItem.getPrice());

        em.persist(sp);
    }
}
```

/ application.properties

application.properties



```
spring.datasource.url=jdbc:mysql://localhost:3306/auto
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

spring.jpa.hibernate.ddl-auto=create
```

ddl-auto

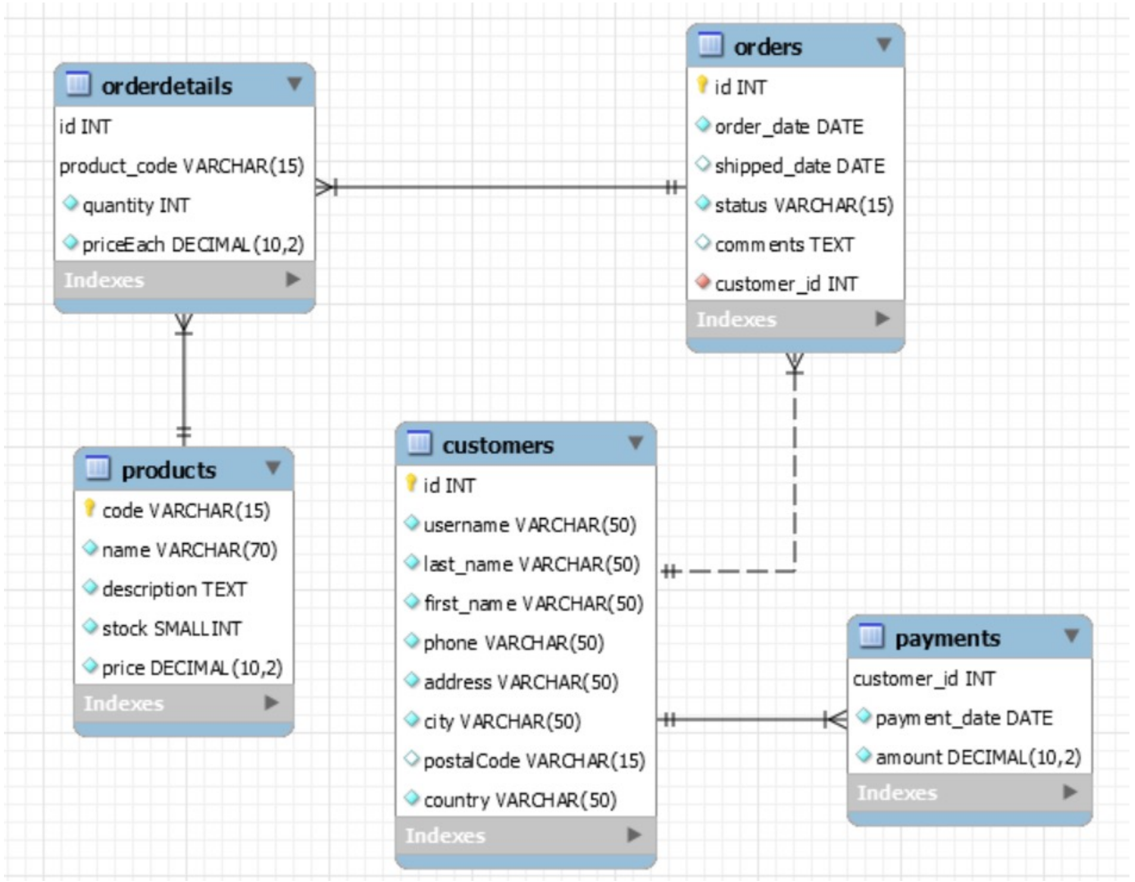
- Validate – validates schema, does not modify the database
- Update – modifies the database schema
- Create – deletes initial data and recreates schema
- None – does not make changes to the database

/ Practice, practice, practice

Practice

- Create the new project schema from @Entity classes
- Attention – The entities should respect the objectual model characteristics

- Map the previous implementation to the new Hibernate approach



orderdetails
id INT
product_code VARCHAR(15)
quantity INT
priceEach DECIMAL(10,2)
Indexes
orders
id INT
order_date DATE
shipped_date DATE
status VARCHAR(15)
comments TEXT
customer_id INT
Indexes
products
code VARCHAR(15)
name VARCHAR(70)
description TEXT
stock SMALLINT
price DECIMAL(10,2)
Indexes
customers
id INT
username VARCHAR(50)
last_name VARCHAR(50)
first_name VARCHAR(50)
phone VARCHAR(50)
address VARCHAR(50)
city VARCHAR(50)
postalCode VARCHAR(15)
country VARCHAR(50)
Indexes
payments
customer_id INT
payment_date DATE
amount DECIMAL(10,2)
Indexes

/ Q&A

MOBILE / ACADEMY