# Java Persistence API (JPA)

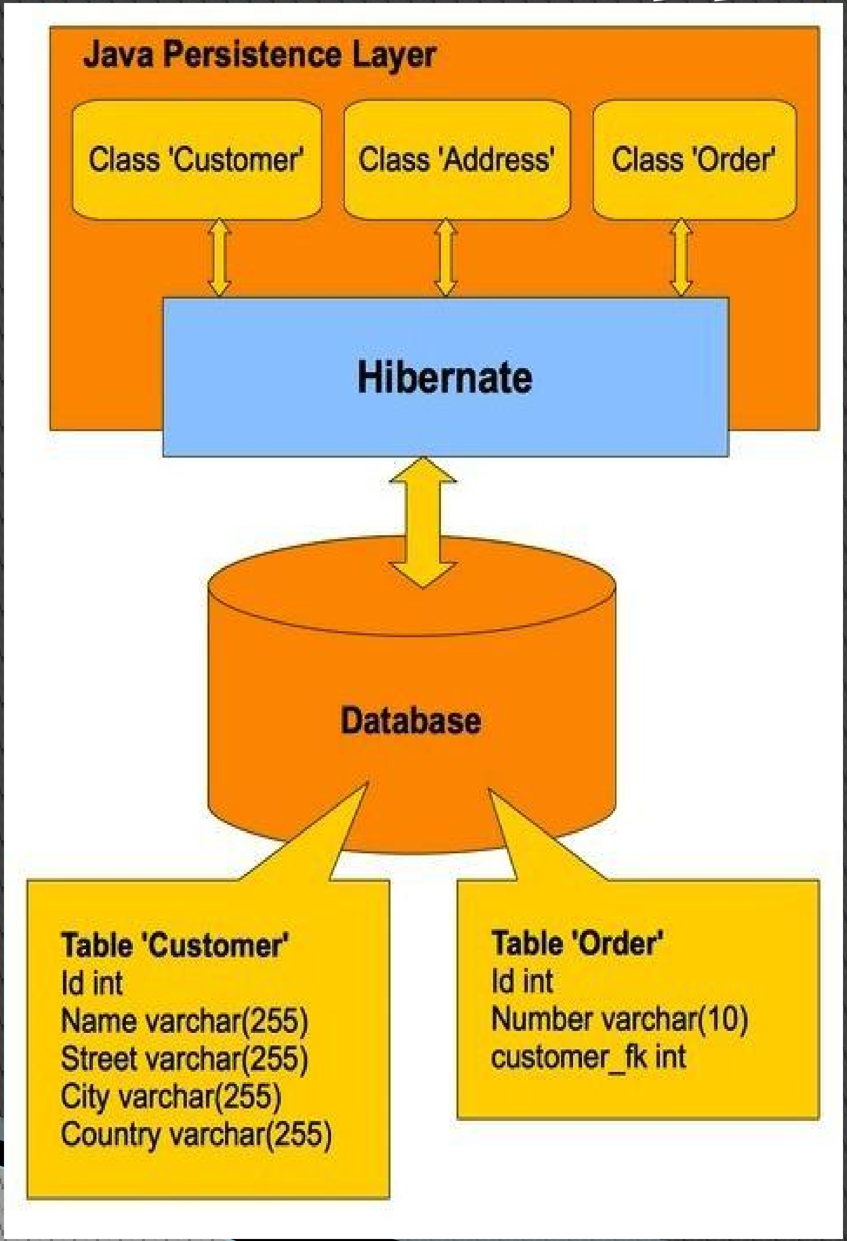
## Topics

- Object-relational mapping
- Java Persistence API
- ORM with annotations
- Persistence Context
- Relationships between entities

## SQL basics

- table, row, column
- primary key, foreign key
- normalized database
- CRUD operations (create, read, update, delete)
- insert into table_name  (id, value) values (111, ‘somevalue’);
- select * from table_name where id = 111;
- update table_name set property = somevalue where id = 111;
- delete from table_name where id = 111;

# Object-relational mapping



## Object-relational mapping

- Table <–> entity class
- Columns of relational table <–> entity attributes, which can be accessed through getters/setters:
  *private String title;*
  *public String getTitle();*
  *public void setTitle(String newTitle);*
- Rows of relational table <–> object instances of entity

## Object method – SQL command mapping

- Entity find (eg by primary key), and load into memory–> SQL SELECT
- Change entity and write it back to DB –> SQL UPDATE
- Create entity –> SQL INSERT
- Remove entity –> SQL DELETE

## Java Persistence API

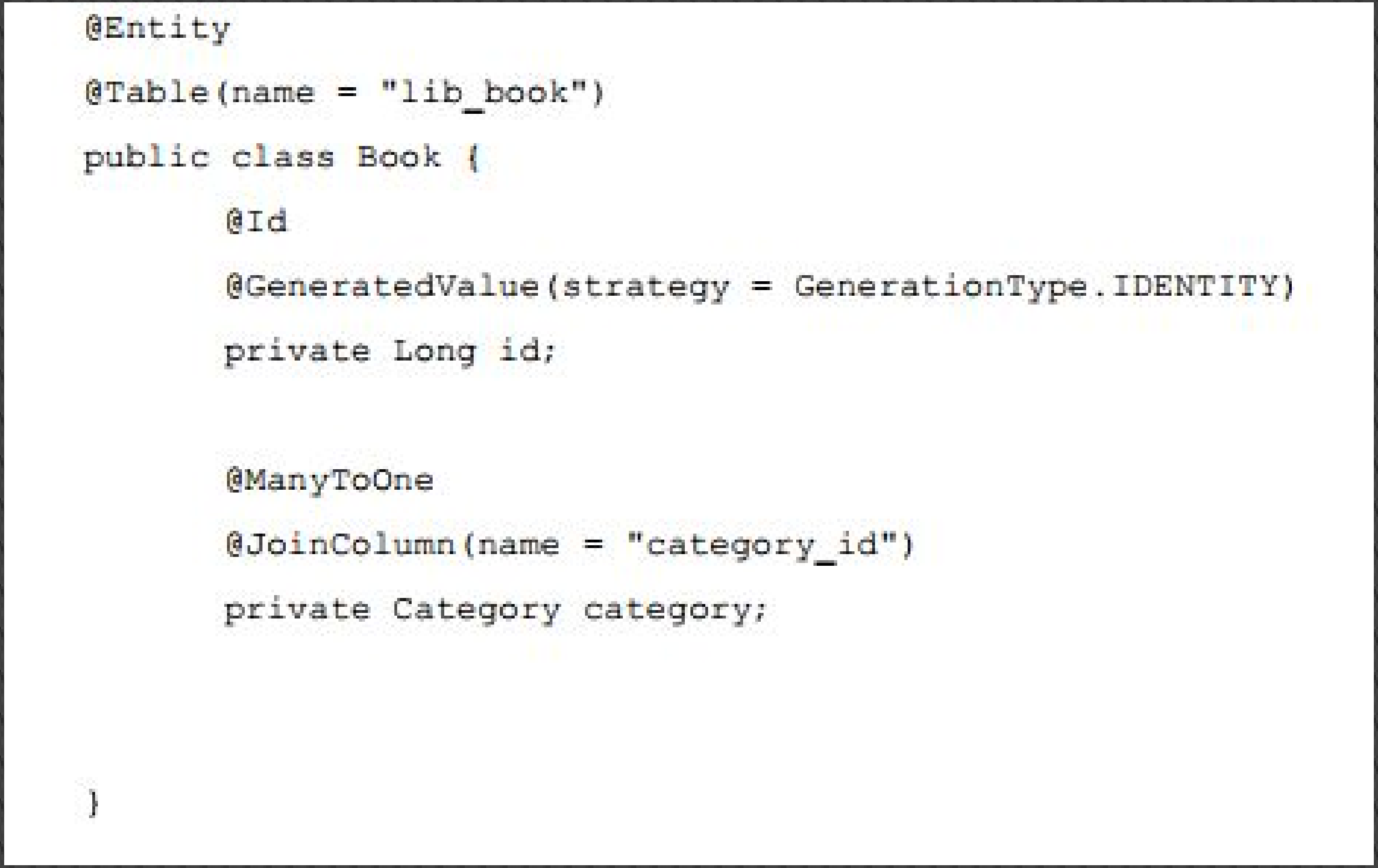
- currently version 2.1
- Persistence Provider: Hibernate, EclipseLink – implementations of the JPA API (we will be using Hibernate)
- to be able to use JPA in Java EE you have to provide a persistence.xml file in the META-INF library: it defines for example the name of the provider and the JNDI name of the database
- javax.persistence package contains the classes we want to use

## Java Persistence API

- persistence.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<persistence version="2.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
   <persistence-unit name="laborPU" transaction-type="JTA">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>java:/laborDS</jta-data-source>
      <properties>
         <property name="hibernate.dialect"
value="org.hibernate.dialect.PostgreSQLDialect"/>
         <property name="hibernate.hbm2ddl.auto" value="none"/>
         <property name="hibernate.show_sql" value="true"/>
         <property name="hibernate.format_sql" value="true"/>
      </properties>
   </persistence-unit>
</persistence>
```

# Object-relational mapping with annotations



```
@Entity
@Table(name = "lib_book")
public class Book {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @ManyToOne
        @JoinColumn(name = "category_id")
        private Category category;



}
```

## Object-relational mapping with annotations

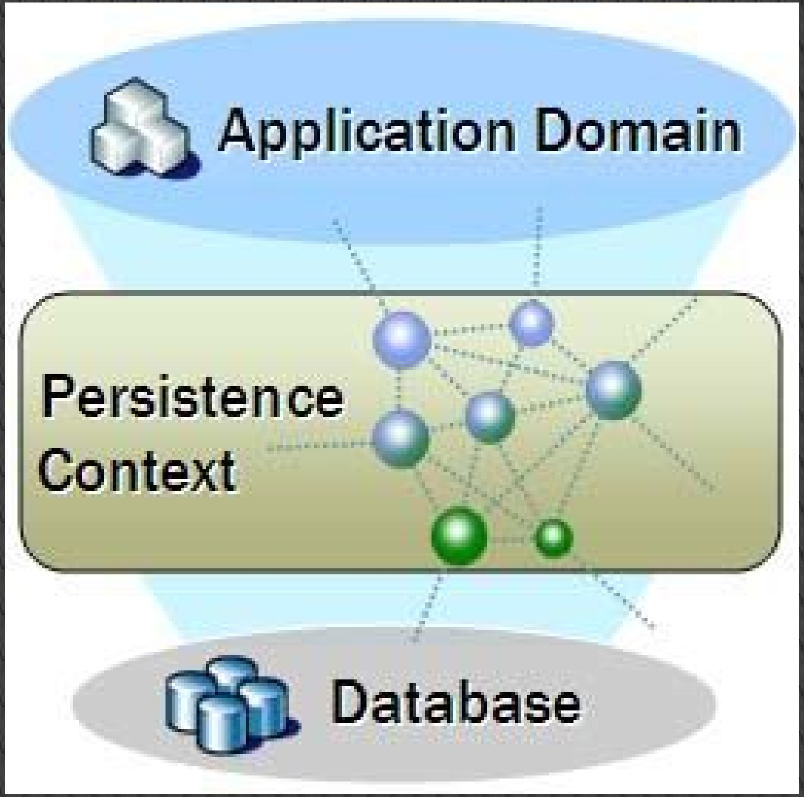
- POJO (Plain Old Java Object) + @Entity annotation
- Table – entity relationship
  @Table(name="myTable", schema="test")

- Column – attribute relationship: @Column(name="myColumn")
- Primary key definition: @Id
- Relationships between entities:
  @OneToOne(mappedBy="person")
  @OneToMany(mappedBy="person")
  @ManyToOne
  @JoinColumn(name="relation")

## Persistence Context

```
@PersistenceContext
private EntityManager em;
```

- A set of entities held in memory and managed by the persistence provider. For any persistent entity there is a unique entity instance.
- Entities, entity lifecycle is managed through the persistence context. the PC is the connection between the database and the Java EE world.



## Entity Manager

- Interface to manage entities and their lifecycle
- 3 types of methods:
  - entity lifecycle management methods
  - database synchronization
  - finding entities

## Entity Manager usage

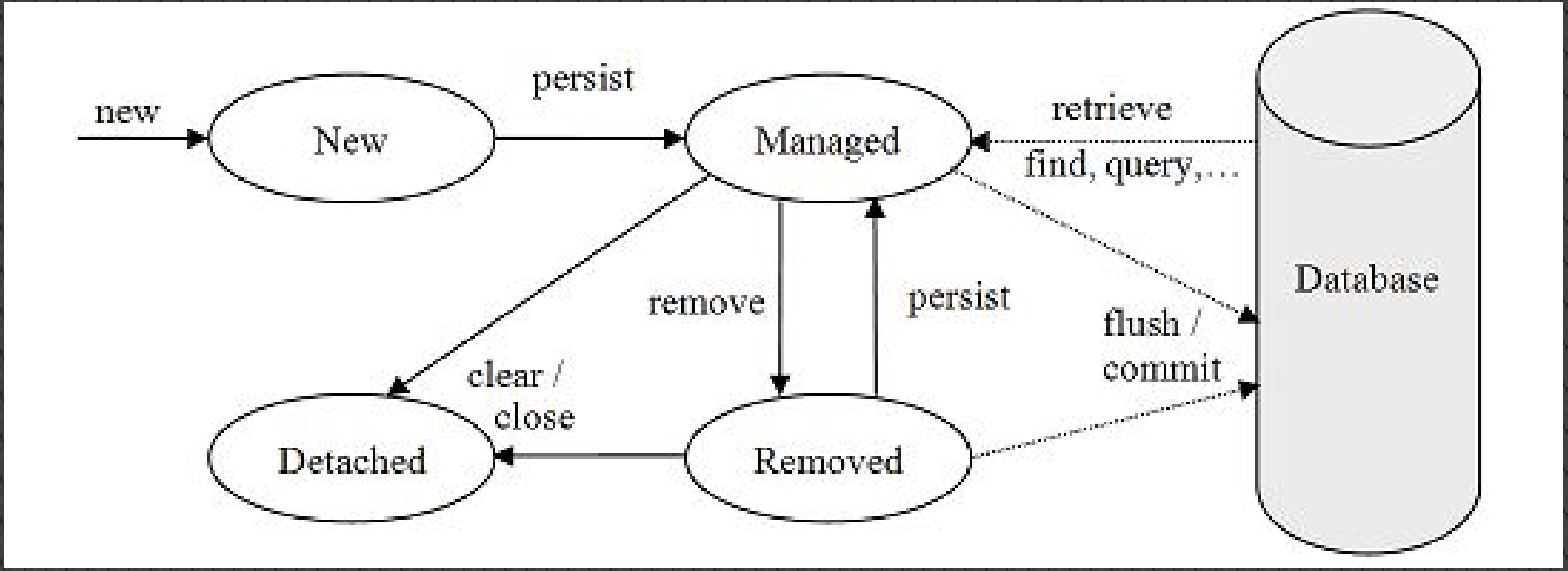
- Usage through injection:

```
@PersistenceContext(unitName = “laborDS”)
private EntityManager em;
```

- @PersistenceContext parameters:

– unitName: if there are more units in the persistence.xml

## Entity lifecycle



## Entity states

- **new**: new Entity() – exists only in memory but not in DB
- **managed**: exists in DB and associated with a persistence context –> entityManager.flush() will write entity from memory to DB. Flush is called automatically.
- **detached**: exists in DB but not is persistence context (memory)
- **removed**: still in persistence context but is flagged for deletion from DB

# Entity lifecycle

- Making  a new entity managed:
- **persist():** if primary key already exists in context, error
- **merge(): if** primary key already exists in context, SQL UPDATE, if not, INSERT
- **merge() returns with a managed entity**
- Entity remove from persistence context:
  - Clearing persistence context: em.clear();
  - Closing persistence context: em.close();
  - For one entity: em.detach(entity);

## Database synchronization

- 2 EntityManager methods are responsible for synchronization:
  - flush(): will write all the changes in PC to DB
  - refresh(entity): reads the entity from DB with changes
- We usually do not call these methods in EJB context because they are automatically called by container.

## Entitások közötti kapcsolatok

- Cardinality types:
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
- Directions:
  - one-directional
  - bi-directional (entities on both ends will have getter/setter methods)
- the developer has to maintain consistency between the two ends
- ONLY one OWNER for each relationship

## Relationship between entities

- Customer orders

```
 public class Customer{

     @OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")
     private Set<Order> orders = new HashSet<>();
}
```

- Orders

```
 public class Order{

  @ManyToOne(fetch = FetchType.LAZY)
  @JoinColumn(name = "CUSTOMER_ID")
  private Customer customer;

}
```

# Entitások közötti kapcsolatok

- Megrendeléshez kapcsolódó termékek – adatbázisban kapcsolótábla

```
public class Order{

  @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
  @JoinTable(name = "order2product", joinColumns = {
  @JoinColumn(name = "ORDER_ID", nullable = false) },
  inverseJoinColumns = { @JoinColumn(name = "PRODUCT_ID",
  nullable = false) })
  private Set<Product> products = new HashSet<>();

}
```

## Fetch

- All 4 relationship attributes can take a fetch attribute @OneToMany(fetch=FetchType.LAZY)
- DEFINES IF AFTER LOADING AN ENTITY ALSO THE ASSOCIATED RELATIONSHIPS ARE LOADED
- LAZY : will not load, only if we reference it, does not take space in memory, faster –> if we need it it takes an extra query
- EAGER (default, exceptions: OneToMany, ManyToMany): associated relationships are loaded
- Best practice:
- leave it lazy by default and use queries with fetch join
- SELECT c form Customer c LEFT JOIN FETCH c.orders

## Cascade

- All 4 relationship attributes can take a cascade attribute
- @OneToMany(cascade={CascadeType.PERSIST, CascadeType.MERGE})
- Possible values: PERSIST, MERGE, REMOVE, REFRESH, ALL
- Defines which entity manager methods will be called for associated entities
- Default: no cascade