

JPA con PostgreSQL

1. Creazione del progetto in IntelliJ

- Apri IntelliJ IDEA
- Clicca su **New Project**
- Scegli **Java**
- Scegli **Maven** come sistema di build
- Scegli la cartella dove salvare il progetto e clicca su **Finish**

2. Struttura delle cartelle

```
biblioteca/  
├── pom.xml  
└── src/  
    ├── main/  
    │   ├── java/  
    │   │   └── com/example/biblioteca/  
    │   │       ├── Main.java  
    │   │       ├── model/  
    │   │       │   ├── Autore.java  
    │   │       │   ├── CasaEditrice.java  
    │   │       │   ├── Categoria.java  
    │   │       │   ├── Pubblicazione.java  
    │   │       │   ├── Libro.java  
    │   │       │   └── Rivista.java  
    │   │       └── dao/  
    │   │           ├── LibroDAO.java  
    │   │           └── RivistaDAO.java
```

```

|
|
|   CategoriaDAO.java
|   CasaEditriceDAO.java
|   AutoreDAO.java
|
|   resources/
|       META-INF/
|           persistence.xml
|
| test/

```

3. Creazione del database in pgAdmin (usando interfaccia grafica)

1. Apri pgAdmin
2. Login con utente postgres
3. Tasto destro su "Databases" > **Create** > **Database**
4. Nome del database: biblioteca
5. Owner: postgres
6. Salva

4. File persistence.xml

Posizionato in src/main/resources/META-INF/persistence.xml (creare la cartella META-INF se non esiste):

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.2"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
  <persistence-unit name="epicode">
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/YYYYYYYYYYYYYYY"/>
      <property name="javax.persistence.jdbc.user" value="postgres"/>

```

```

    <property name="javax.persistence.jdbc.password" value="XXXXXX"/>
    <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
    <property name="hibernate.default_schema" value="public"/>
    <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
    <property name="hibernate.hbm2ddl.auto" value="create"/>
  </properties>
</persistence-unit>
</persistence>

```

- Sostituire XXXXXX con la password del proprio utente postgres
- Sostituire YYYYYYYYYYYY con il nome del database creato (es. biblioteca)

Valori per `hibernate.hbm2ddl.auto` :

- create : crea tutte le tabelle cancellando quelle esistenti
- update : aggiorna le tabelle mantenendo i dati
- validate : controlla che le entità siano coerenti col DB
- none : disattiva il controllo

5. Annotazioni JPA (Jakarta)

```

@Entity // Indica che la classe è un'entità JPA (una tabella nel DB)
@Table(name = "nome_tabella") // Specifica il nome della tabella (minuscolo consigliato)
@Column(name = "colonna", length = 100, unique = true) // Specifica la colonna, dimensione e se è unica
@Id // Identifica la chiave primaria
@GeneratedValue(strategy = GenerationType.IDENTITY) // Autoincremento della chiave primaria
@Enumerated(EnumType.STRING) // Per salvare gli ENUM come stringhe
@ManyToOne // Molti oggetti collegati a uno
@OneToMany(mappedBy = "...") // Uno collegato a molti, lato inverso

```

```
@ManyToMany // Relazione multi-a-molti
@OneToOne // Relazione uno-a-uno
```

6. Dipendenze Maven (pom.xml)

```
<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.2.5.Final</version>
  </dependency>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.7.1</version>
  </dependency>
</dependencies>
```



Autore.java

```
package com.example.biblioteca.model;

import jakarta.persistence.*;
import java.util.Objects;

@Entity
```

```
@Table(name = "autore")
public class Autore {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nome", length = 100, nullable = false)
    private String nome;

    @Column(name = "cognome", length = 100, nullable = false)
    private String cognome;

    // Costruttori
    public Autore() {}

    public Autore(String nome, String cognome) {
        this.nome = nome;
        this.cognome = cognome;
    }

    // Getter e Setter
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    public String getCognome() { return cognome; }
    public void setCognome(String cognome) { this.cognome = cognome; }

    // equals e hashCode
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Autore)) return false;
```

```

        Autore autore = (Autore) o;
        return Objects.equals(id, autore.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }

    // toString
    @Override
    public String toString() {
        return nome + " " + cognome;
    }
}

```



CasaEditrice.java

```

package com.example.biblioteca.model;

import jakarta.persistence.*;
import java.util.Objects;

@Entity
@Table(name = "casa_editrice")
public class CasaEditrice {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nome", length = 100, nullable = false, unique = true)
    private String nome;
}

```

```
// Costruttori
public CasaEditrice() {}

public CasaEditrice(String nome) {
    this.nome = nome;
}

// Getter e Setter
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome; }

// equals e hashCode
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof CasaEditrice)) return false;
    CasaEditrice that = (CasaEditrice) o;
    return Objects.equals(id, that.id);
}

@Override
public int hashCode() {
    return Objects.hash(id);
}

// toString
@Override
public String toString() {
    return nome;
}
}
```



Categoria.java

```
package com.example.biblioteca.model;

import jakarta.persistence.*;
import java.util.Objects;

@Entity
@Table(name = "categoria")
public class Categoria {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nome", length = 100, nullable = false, unique = true)
    private String nome;

    // Costruttori
    public Categoria() {}

    public Categoria(String nome) {
        this.nome = nome;
    }

    // Getter e Setter
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    // equals e hashCode
    @Override
```



```

public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Categoria)) return false;
    Categoria categoria = (Categoria) o;
    return Objects.equals(id, categoria.id);
}

@Override
public int hashCode() {
    return Objects.hash(id);
}

// toString
@Override
public String toString() {
    return nome;
}
}




```

Tre strategie di ereditarietà JPA per `@Inheritance` e cosa succede a livello di database:



`@Inheritance(strategy = InheritanceType.JOINED)`

Strategia consigliata quando vuoi normalizzazione e integrità referenziale.

-  **Cosa fa:** crea una **tabella per ogni classe**, collegata tramite chiave esterna.
-  **Tabelle create:**
 - `pubblicazione` : contiene i campi comuni (id, titolo, autore, casaEditrice).
 - `libro` : contiene `id` (FK verso `pubblicazione`) + `numero_pagine` .
 - `rivista` : contiene `id` (FK verso `pubblicazione`) + `periodicita` .
-  **Pro:** meno ridondanza.

- ⚠️ **Contro:** le query su `Libro` e `Rivista` richiedono join.

`@Inheritance(strategy = InheritanceType.SINGLE_TABLE)`

Tutto in una singola tabella.

- 🔧 **Cosa fa:** crea una sola tabella per tutte le classi figlie e padre.
- 📁 **Tabelle create:**
 - `pubblicazione` : contiene tutti i campi di `Libro` e `Rivista` + una colonna `DTYPE` automatica.
- 📌 **Pro:** query più veloci, semplice.
- ⚠️ **Contro:** molti campi `NULL` , scarsa normalizzazione.

`@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)`

Una tabella per ogni sottoclasse, senza padre.

- 🔧 **Cosa fa:** crea una tabella per ogni sottoclasse, replicando i campi del padre.
- 📁 **Tabelle create:**
 - `libro` : con tutti i campi di `Pubblicazione` + `numero_pagine` .
 - `rivista` : con tutti i campi di `Pubblicazione` + `periodicita` .
 - ❌ Nessuna tabella `pubblicazione` .
- 📌 **Pro:** semplice da leggere.
- ⚠️ **Contro:** duplicazione dati, impossibile fare query su `Pubblicazione` .

Se vuoi modificare il tipo di strategia, basta cambiare questa annotazione nella superclasse:

```
@Inheritance(strategy = InheritanceType.SINGLE_TABLE) // oppure TABLE_PER_CLASS, JOINED
```

```
package com.example.biblioteca.model;
```

```
import jakarta.persistence.*;
```

```
import java.util.List;
```

```
import java.util.Objects;
```

```
@Entity
```

```
@Inheritance(strategy = InheritanceType.JOINED) // ereditarietà con tabelle separate collegate
```

```
@Table(name = "pubblicazione") // nome tabella in minuscolo
```

```
public abstract class Pubblicazione {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY) // autoincremento
```

```
    private Long id;
```

```
    @Column(name = "titolo", length = 200, nullable = false)
```

```
    private String titolo;
```

```
    @ManyToOne
```

```
    @JoinColumn(name = "autore_id")
```

```
    private Autore autore;
```

```
    @ManyToOne
```

```
    @JoinColumn(name = "casa_editrice_id")
```

```
    private CasaEditrice casaEditrice;
```

```
    @ManyToMany
```

```
    @JoinTable(
```

```
        name = "pubblicazione_categoria",
```

```
        joinColumns = @JoinColumn(name = "pubblicazione_id"),
```

```
        inverseJoinColumns = @JoinColumn(name = "categoria_id")
```

```
    )
```

```

private List<Categoria> categorie;

// Costruttore senza ID
public Pubblicazione(String titolo, Autore autore, CasaEditrice casaEditrice, List<Categoria> categorie) {
    this.titolo = titolo;
    this.autore = autore;
    this.casaEditrice = casaEditrice;
    this.categorie = categorie;
}

public Pubblicazione() {}

// Getter e Setter
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public String getTitolo() { return titolo; }
public void setTitolo(String titolo) { this.titolo = titolo; }

public Autore getAutore() { return autore; }
public void setAutore(Autore autore) { this.autore = autore; }

public CasaEditrice getCasaEditrice() { return casaEditrice; }
public void setCasaEditrice(CasaEditrice casaEditrice) { this.casaEditrice = casaEditrice; }

public List<Categoria> getCategorie() { return categorie; }
public void setCategorie(List<Categoria> categorie) { this.categorie = categorie; }

// equals e hashCode su ID
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Pubblicazione)) return false;
    Pubblicazione that = (Pubblicazione) o;
    return Objects.equals(id, that.id);
}

```

```

@Override
public int hashCode() {
    return Objects.hash(id);
}

// toString
@Override
public String toString() {
    return "Pubblicazione{" +
        "id=" + id +
        ", titolo='" + titolo + '\'' +
        ", autore=" + autore +
        ", casaEditrice=" + casaEditrice +
        ", categorie=" + categorie +
        '}';
}
}

```

Libro.java

```

package com.example.biblioteca.model;

import jakarta.persistence.*;
import java.util.List;

@Entity
@Table(name = "libro")
public class Libro extends Pubblicazione {

    @Column(name = "numero_pagine")
    private int numeroPagine;
}

```

```

// Costruttori
public Libro() {
    super();
}

public Libro(String titolo, Autore autore, CasaEditrice casaEditrice, List<Categoria> categorie, int numeroPagine) {
    super(titolo, autore, casaEditrice, categorie);
    this.numeroPagine = numeroPagine;
}

// Getter e Setter
public int getNumeroPagine() { return numeroPagine; }
public void setNumeroPagine(int numeroPagine) { this.numeroPagine = numeroPagine; }

// equals e hashCode (ereditano id da Pubblicazione)
@Override
public boolean equals(Object o) {
    return super.equals(o);
}

@Override
public int hashCode() {
    return super.hashCode();
}

// toString
@Override
public String toString() {
    return "Libro{" +
        "id=" + getId() +
        ", titolo=" + getTitolo() + '\'' +
        ", autore=" + getAutore() +
        ", casaEditrice=" + getCasaEditrice() +
        ", categorie=" + getCategorie() +
        ", numeroPagine=" + numeroPagine +
        '}';
}

```

```
}  
}
```

✓ Rivista.java

```
package com.example.biblioteca.model;  
  
import jakarta.persistence.*;  
import java.util.List;  
  
@Entity  
@Table(name = "rivista")  
public class Rivista extends Pubblicazione {  
  
    @Column(name = "periodicita", length = 50)  
    private String periodicita;  
  
    // Costruttori  
    public Rivista() {  
        super();  
    }  
  
    public Rivista(String titolo, Autore autore, CasaEditrice casaEditrice, List<Categoria> categorie, String periodicita) {  
        super(titolo, autore, casaEditrice, categorie);  
        this.periodicita = periodicita;  
    }  
  
    // Getter e Setter  
    public String getPeriodicita() { return periodicita; }  
    public void setPeriodicita(String periodicita) { this.periodicita = periodicita; }  
  
    // equals e hashCode (ereditano id da Pubblicazione)  
    @Override
```

```

    public boolean equals(Object o) {
        return super.equals(o);
    }

    @Override
    public int hashCode() {
        return super.hashCode();
    }

    // toString
    @Override
    public String toString() {
        return "Rivista{" +
            "id=" + getId() +
            ", titolo='" + getTitolo() + '\'' +
            ", autore=" + getAutore() +
            ", casaEditrice=" + getCasaEditrice() +
            ", categorie=" + getCategorie() +
            ", periodicit =" + periodicit  + '\'' +
            '}';
    }
}

```



CategoriaDAO.java

```

package com.example.biblioteca.dao;

import com.example.biblioteca.model.Categoria;

import jakarta.persistence.EntityManager;

public class CategoriaDAO {

```



```
private final EntityManager em;

public CategoriaDAO(EntityManager em) {
    this.em = em;
}

public void save(Categoria categoria) {
    em.getTransaction().begin();
    em.persist(categoria);
    em.getTransaction().commit();
}

public Categoria getById(Long id) {
    return em.find(Categoria.class, id);
}

public void delete(Categoria categoria) {
    em.getTransaction().begin();
    em.remove(em.contains(categoria) ? categoria : em.merge(categoria));
    em.getTransaction().commit();
}
}
```



AutoreDAO.java

```
package com.example.biblioteca.dao;

import com.example.biblioteca.model.Autore;

import jakarta.persistence.EntityManager;

public class AutoreDAO {
```

```

private final EntityManager em;

public AutoreDAO(EntityManager em) {
    this.em = em;
}

public void save(Autore autore) {
    em.getTransaction().begin();
    em.persist(autore);
    em.getTransaction().commit();
}

public Autore getById(Long id) {
    return em.find(Autore.class, id);
}

// em.remove(obj) può essere chiamato solo su entità gestite (managed).
// Se passi un'entità detached (ad esempio creata o caricata
// da un altro EntityManager o dopo la chiusura della sessione),
// remove() lancia un'eccezione.
// em.contains(obj) verifica se l'oggetto è gestito.
// Se non lo è, lo rendiamo gestito con em.merge(obj), poi lo rimuoviamo.

public void delete(Autore autore) {
    em.getTransaction().begin();
    em.remove(em.contains(autore) ? autore : em.merge(autore));
    em.getTransaction().commit();
}
}

```



CasaEditriceDAO.java

```
package com.example.biblioteca.dao;

import com.example.biblioteca.model.CasaEditrice;

import jakarta.persistence.EntityManager;

public class CasaEditriceDAO {

    private final EntityManager em;

    public CasaEditriceDAO(EntityManager em) {
        this.em = em;
    }

    public void save(CasaEditrice casaEditrice) {
        em.getTransaction().begin();
        em.persist(casaEditrice);
        em.getTransaction().commit();
    }

    public CasaEditrice getById(Long id) {
        return em.find(CasaEditrice.class, id);
    }

    public void delete(CasaEditrice casaEditrice) {
        em.getTransaction().begin();
        em.remove(em.contains(casaEditrice) ? casaEditrice : em.merge(casaEditrice));
        em.getTransaction().commit();
    }
}
```

Perfetto! Ecco i DAO avanzati per `Libro` e `Rivista`, con metodi CRUD e query named.

✓ NamedQuery in Libro.java

Aggiungiamo queste in cima alla classe Libro :

```
@NamedQueries({
    @NamedQuery(name = "Libro.getByAutore", query = "SELECT l FROM Libro l WHERE l.autore = :autore"),
    @NamedQuery(name = "Libro.getByCasaEditrice", query = "SELECT l FROM Libro l WHERE l.casaEditrice = :casaEditrice")
})
```

```
@NamedQueries({ ... })
@Entity
@Table(name = "libro")
public class Libro extends Pubblicazione { ... }
```

✓ LibroDAO.java

```
package com.example.biblioteca.dao;

import com.example.biblioteca.model.Libro;
import com.example.biblioteca.model.Autore;
import com.example.biblioteca.model.CasaEditrice;

import jakarta.persistence.EntityManager;
import java.util.List;

public class LibroDAO {

    private final EntityManager em;

    public LibroDAO(EntityManager em) {
```

```

        this.em = em;
    }

    public void save(Libro libro) {
        em.getTransaction().begin();
        em.persist(libro);
        em.getTransaction().commit();
    }

    public Libro getById(Long id) {
        return em.find(Libro.class, id);
    }

    public void delete(Libro libro) {
        em.getTransaction().begin();
        em.remove(em.contains(libro) ? libro : em.merge(libro));
        em.getTransaction().commit();
    }

    public List<Libro> getByAutore(Autore autore) {
        return em.createNamedQuery("Libro.getByAutore", Libro.class)
            .setParameter("autore", autore)
            .getResultList();
    }

    public List<Libro> getByCasaEditrice(CasaEditrice casaEditrice) {
        return em.createNamedQuery("Libro.getByCasaEditrice", Libro.class)
            .setParameter("casaEditrice", casaEditrice)
            .getResultList();
    }
}

```



RivistaDAO.java

```
package com.example.biblioteca.dao;

import com.example.biblioteca.model.Rivista;

import jakarta.persistence.EntityManager;

public class RivistaDAO {

    private final EntityManager em;

    public RivistaDAO(EntityManager em) {
        this.em = em;
    }

    public void save(Rivista rivista) {
        em.getTransaction().begin();
        em.persist(rivista);
        em.getTransaction().commit();
    }

    public Rivista getById(Long id) {
        return em.find(Rivista.class, id);
    }

    public void delete(Rivista rivista) {
        em.getTransaction().begin();
        em.remove(em.contains(rivista) ? rivista : em.merge(rivista));
        em.getTransaction().commit();
    }
}
```



Main.java

```
package com.example.biblioteca;

import com.example.biblioteca.dao.*;
import com.example.biblioteca.model.*;

import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("epicode");
        EntityManager em = emf.createEntityManager();

        CategoriaDAO categoriaDAO = new CategoriaDAO(em);
        AutoreDAO autoreDAO = new AutoreDAO(em);
        CasaEditriceDAO casaEditriceDAO = new CasaEditriceDAO(em);
        LibroDAO libroDAO = new LibroDAO(em);
        RivistaDAO rivistaDAO = new RivistaDAO(em);

        // Categorie
        Categoria narrativa = new Categoria("Narrativa");
        Categoria scienza = new Categoria("Scienza");
        Categoria storia = new Categoria("Storia");
        Categoria tecnologia = new Categoria("Tecnologia");
        Categoria fumetti = new Categoria("Fumetti");

        List<Categoria> categorie = Arrays.asList(narrativa, scienza, storia, tecnologia, fumetti);
        categorie.forEach(categoriaDAO::save);

        // Autori
        Autore autore1 = new Autore("Mario", "Rossi");
```

```
Autore autore2 = new Autore("Laura", "Verdi");
Autore autore3 = new Autore("Giorgio", "Bianchi");
Autore autore4 = new Autore("Sara", "Neri");
Autore autore5 = new Autore("Luigi", "Blu");

List<Autore> autori = Arrays.asList-autore1, autore2, autore3, autore4, autore5);
autori.forEach-autoreDAO::save);

// Case editrici
CasaEditrice mondadori = new CasaEditrice("Mondadori");
CasaEditrice feltrinelli = new CasaEditrice("Feltrinelli");
CasaEditrice einaudi = new CasaEditrice("Einaudi");
CasaEditrice laterza = new CasaEditrice("Laterza");
CasaEditrice panini = new CasaEditrice("Panini");

List<CasaEditrice> editori = Arrays.asList(mondadori, feltrinelli, einaudi, laterza, panini);
editori.forEach(casaEditriceDAO::save);

// Libri
Libro libro1 = new Libro("Il grande viaggio", autore1, mondadori, List.of(narrativa, storia), 320);
Libro libro2 = new Libro("La fisica facile", autore2, feltrinelli, List.of(scienza), 210);
Libro libro3 = new Libro("Codice e cuore", autore3, einaudi, List.of(tecnologia), 180);
Libro libro4 = new Libro("Viaggio nella storia", autore4, laterza, List.of(storia), 270);
Libro libro5 = new Libro("Fumetti per tutti", autore5, panini, List.of(fumetti), 140);

List<Libro> libri = List.of(libro1, libro2, libro3, libro4, libro5);
libri.forEach(libroDAO::save);

// Rivista
Rivista rivista1 = new Rivista("Scienza Oggi", autore2, feltrinelli, List.of(scienza, tecnologia), "Mensile");
rivistaDAO.save(rivista1);

// GetById
Categoria cat = categoriaDAO.getById(narrativa.getId());
System.out.println("Categoria: " + cat);
```



```
Autore aut = autoreDAO.getById(autore1.getId());
System.out.println("Autore: " + aut);

CasaEditrice ce = casaEditriceDAO.getById(mondadori.getId());
System.out.println("Casa Editrice: " + ce);

Libro l = libroDAO.getById(libro1.getId());
System.out.println("Libro: " + l);

Rivista r = rivistaDAO.getById(rivista1.getId());
System.out.println("Rivista: " + r);

// Query: libri per autore
System.out.println("\nLibri scritti da \"" + autore1.getNome() + "\":\");
libroDAO.getByAutore(autore1).forEach(System.out::println);

// Delete esempio
libroDAO.delete(libro5);
System.out.println("\nLibro cancellato: \"" + libro5.getTitolo());

// Chiudi EM e Factory
em.close(); // chiude la sessione con il database
emf.close(); // chiude la factory: importante per rilasciare le risorse del connection pool
}
}
```