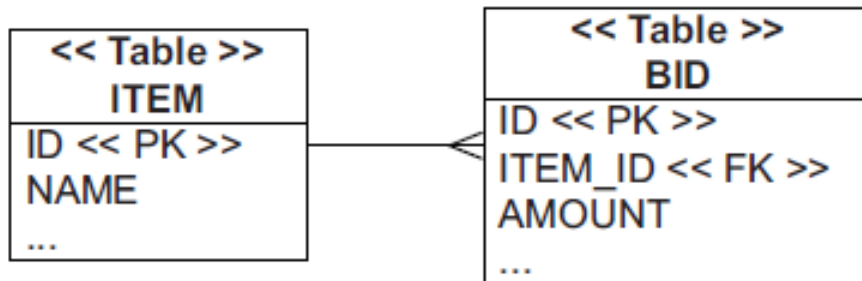


Mappatura di associazioni tra entity (many-to-one)



unidirezionale

`@Entity`

```
public class Bid {
```

```
    @ManyToOne(fetch = FetchType.LAZY)
```

```
    @JoinColumn(name = "ITEM_ID", nullable = false)
```

```
    protected Item item;
```

```
    // ...
```

```
}
```

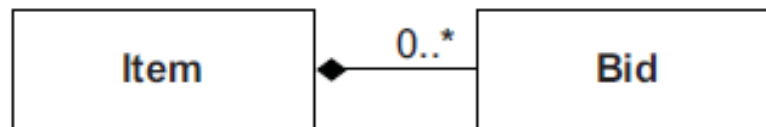
← Defaults to EAGER

Mappatura di associazioni tra entity (many-to-one)

Perchè renderla bidirezionale:

- Chiamando `someItem.getBids()` Hibernate esegue in automatico `SELECT * from BID where ITEM_ID = ?`
- Puoi propagare on cascade i cambiamenti di stato di Item a tutti i Bid referenziati nella collection (verifica sempre se Bid, in questo caso, non debba essere considerato un embeddable type invece di un'entity)

Mappatura di associazioni tra entity (one-to-many)



```
@Entity
public class Item {
    @OneToMany(mappedBy = "item",
                fetch = FetchType.LAZY)
    protected Set<Bid> bids = new HashSet<>();
    // ...
}
```

Required for
bidirectional
association

← Default

Mappatura di associazioni tra entity (cascading states) CascadeType.PERSIST

```
Item someItem = new Item("Some Item");

Bid someBid = new Bid(new BigDecimal("123.00"), someItem);
someItem.getBids().add(someBid);           ← Don't forget!

Item someItem = new Item("Some Item");
em.persist(someItem);

Bid someBid = new Bid(new BigDecimal("123.00"), someItem);
someItem.getBids().add(someBid);           ← Don't forget!
em.persist(someBid);

Bid secondBid = new Bid(new BigDecimal("456.00"), someItem);
someItem.getBids().add(secondBid);
em.persist(secondBid);

tx.commit();                               ← Dirty checking; SQL execution
```

Mappatura di associazioni tra entity (cascading states) CascadeType.PERSIST

```
@Entity
public class Item {

    @OneToMany(mappedBy = "item", cascade = CascadeType.PERSIST)
    protected Set<Bid> bids = new HashSet<>();

    // ...
}
```

```
Item someItem = new Item("Some Item");
em.persist(someItem);
```

Saves the bids automatically
(later, at flush time)

```
Bid someBid = new Bid(new BigDecimal("123.00"), someItem);
someItem.getBids().add(someBid);
```

```
Bid secondBid = new Bid(new BigDecimal("456.00"), someItem);
someItem.getBids().add(secondBid);
```

```
tx.commit();
```

Dirty checking;
SQL execution

Mappatura di associazioni tra entity (cascading states) CascadeType.DELETE

```
Item item = em.find(Item.class, ITEM_ID);  
  
for (Bid bid : item.getBids()) {  
    em.remove(bid);  
}  
  
em.remove(item);
```

← ① Removes bids

← ② Removes owner

Mappatura di associazioni tra entity (cascading states) CascadeType.DELETE

```
@Entity
public class Item {

    @OneToMany(mappedBy = "item",
                cascade = {CascadeType.PERSIST, CascadeType.REMOVE})
    protected Set<Bid> bids = new HashSet<>();

    // ...
}
```

```
Item item = em.find(Item.class, ITEM_ID);
em.remove(item);
```


Deletes bids one by one after loading them

Mappatura di associazioni tra entity (cascading states) orphan removal

```
@Entity
public class Item {

    @OneToMany(mappedBy = "item",
                cascade = CascadeType.PERSIST,
                orphanRemoval = true)
    protected Set<Bid> bids = new HashSet<>();
    // ...
}
```

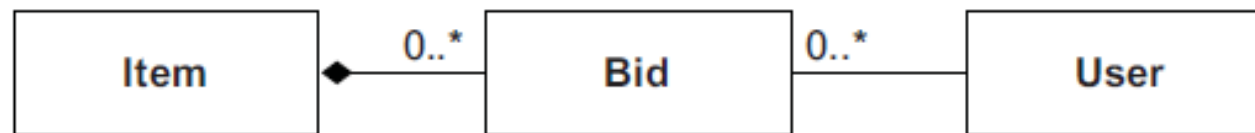
Includes
CascadeType.REMOVE



```
Item item = em.find(Item.class, ITEM_ID);
Bid firstBid = item.getBids().iterator().next();
item.getBids().remove(firstBid);
```

← One bid removed

Mappatura di associazioni tra entity (cascading states) orphan removal



```
User user = em.find(User.class, USER_ID);
assertEquals(user.getBids().size(), 2);           ← User made two bids

Item item = em.find(Item.class, ITEM_ID);
Bid firstBid = item.getBids().iterator().next();
item.getBids().remove(firstBid);                 ← One bid removed

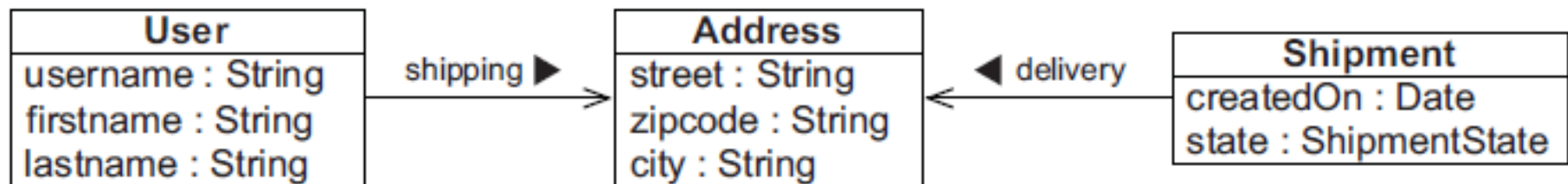
// FAILURE!
// assertEquals(user.getBids().size(), 1);
assertEquals(user.getBids().size(), 2);           ← Still two!
```

Mappatura di associazioni tra entity (cascading states) orphan removal

Considerare sempre la possibilità di un mapping più semplice:

In questo caso si poteva usare una collection di components (Item#bids) mappati con *@ElementCollection*; Bid sarà un *@Embeddable* (non più un Entity) con una property (ad esempio *bidder*), mappata con *@ManyToOne*, che punta a User (una classe Embeddable può avere un'associazione unidirezionale ad un'Entity).

Mappatura di associazioni tra entity (one-to-one)



Mappatura di associazioni tra entity (one-to-one)

1. Chiave primaria condivisa

```
@Entity
public class Address {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    @NotNull
    protected String street;

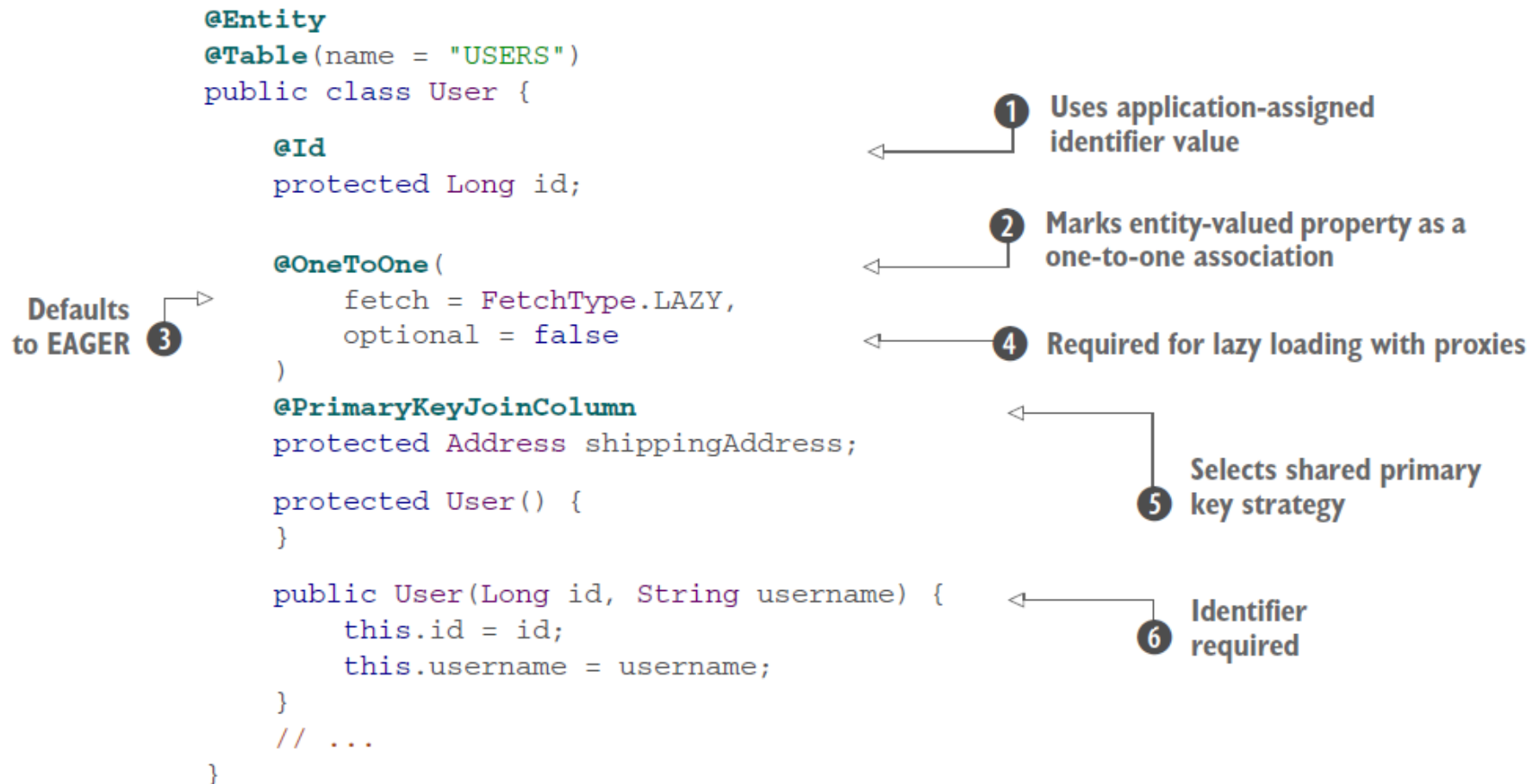
    @NotNull
    protected String zipcode;

    @NotNull
    protected String city;

    // ...
}
```

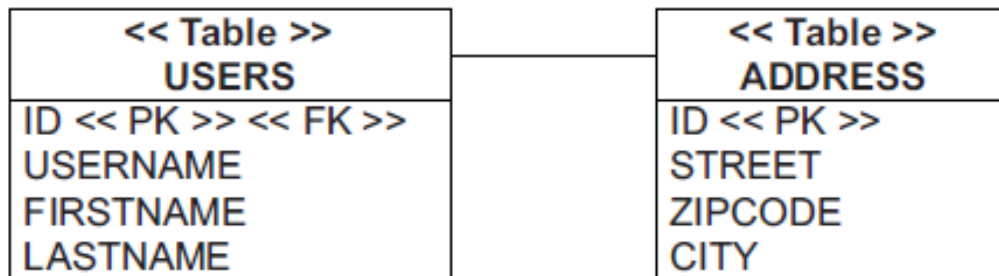
Mappatura di associazioni tra entity (one-to-one)

1. Chiave primaria condivisa



Mappatura di associazioni tra entity (one-to-one)

1. Chiave primaria condivisa



```
Address someAddress =  
    new Address("Some Street 123", "12345", "Some City");
```

```
em.persist(someAddress);
```

```
User someUser =  
    new User(  
        someAddress.getId(),  
        "johndoe"  
    );
```

```
em.persist(someUser);
```

```
someUser.setShippingAddress(someAddress);
```

← Generates
identifier value

← Assigns the same
identifier value

← Optional

Mappatura di associazioni tra entity (one-to-one)

1. Chiave primaria condivisa

- Si salva prima Address e poi si usa il suo id per User: bisogna scegliere una strategia di generazione dell'id che produca l'id prima dell'INSERT
- Il lazy-loading funziona solo se l'associazione è non opzionale (a meno di ricorrere alla bytecode instrumentation)
- È un'associazione unidirezionale

Mappatura di associazioni tra entity (one-to-one)


2. Generazione di chiave primaria esterna

```
@Entity
@Table(name = "USERS")
public class User {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    @OneToOne(
        mappedBy = "user",
        cascade = CascadeType.PERSIST
    )
    protected Address shippingAddress;
    // ...
}
```

Vogliamo un'associazione bidirezionale



Mappatura di associazioni tra entity (one-to-one)

2. Generazione di chiave primaria esterna

```
@Entity
public class Address {

    @Id
    @GeneratedValue(generator = "addressKeyGenerator")
    @org.hibernate.annotations.GenericGenerator(
        name = "addressKeyGenerator",
        strategy = "foreign",
        parameters =
            @org.hibernate.annotations.Parameter(
                name = "property", value = "user"
            )
    )
    protected Long id;

    @OneToOne(optional = false)
    @PrimaryKeyJoinColumn
    protected User user;

    protected Address() {
    }

    public Address(User user) {
        this.user = user;
    }

    public Address(User user, String street, String zipcode, String city) {
        this.user = user;
        this.street = street;
        this.zipcode = zipcode;
        this.city = city;
    }
    // ...
}
```

1 Defines a primary key value generator

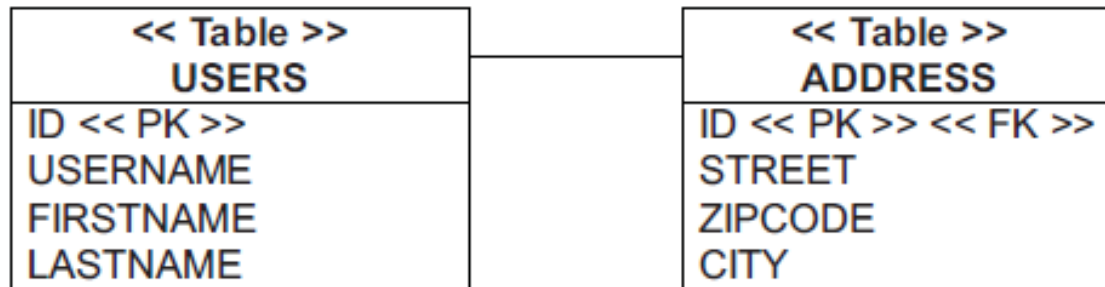
2 Creates foreign key constraint

3 Address must have a reference to a User

4 Public constructors of Address

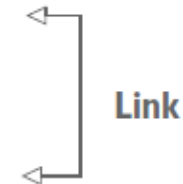
Mappatura di associazioni tra entity (one-to-one)

2. Generazione di chiave primaria esterna



```
User someUser = new User("johndoe");
Address someAddress =
    new Address(
        someUser,
        "Some Street 123", "12345", "Some City"
    );
someUser.setShippingAddress(someAddress);
em.persist(someUser);
```

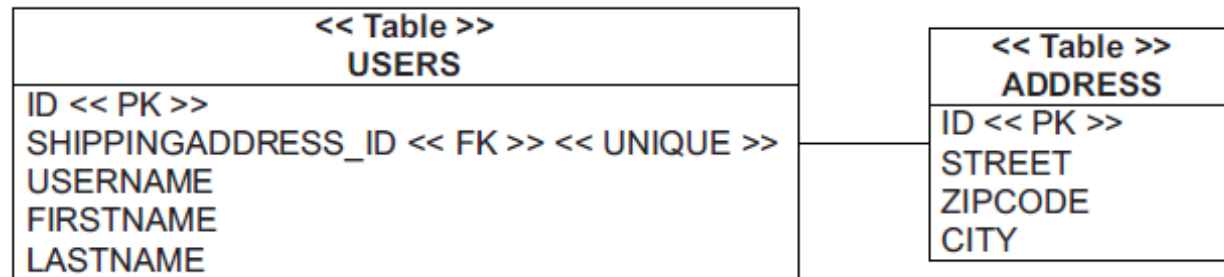
Non occorrono `someAddress.getId()`
o `someUser.getId()`



← Transitive persistence of `shippingAddress`

Mappatura di associazioni tra entity (one-to-one)

3. Colonna di join con chiave esterna



```
@Entity
@Table(name = "USERS")
public class User {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    @OneToOne(
        fetch = FetchType.LAZY,
        optional = false,                ← NOT NULL
        cascade = CascadeType.PERSIST
    )
    @JoinColumn(unique = true)           ← Defaults to SHIPPINGADDRESS_ID
    protected Address shippingAddress;

    // ...
}
```

Mappatura di associazioni tra entity (one-to-one)

3. Colonna di join con chiave esterna

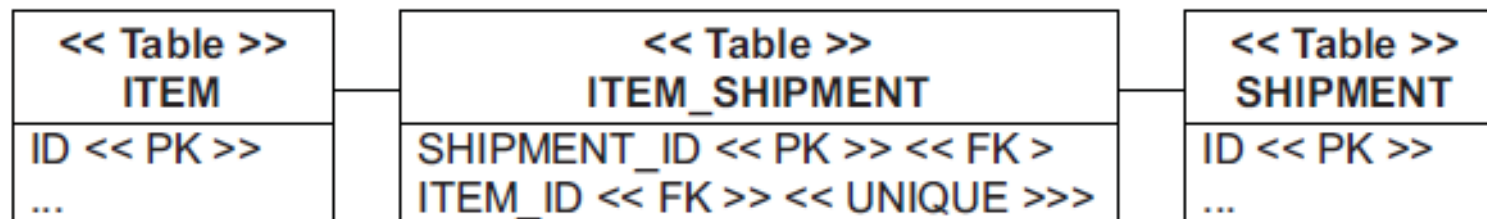
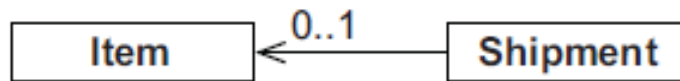
```
User someUser =  
    new User("johndoe");  
  
Address someAddress =  
    new Address("Some Street 123", "12345", "Some City");  
  
someUser.setShippingAddress(someAddress);  
em.persist(someUser);
```

← Link

← Transitive persistence of shippingAddress

Mappatura di associazioni tra entity (one-to-one)

4. Tabella di join



Mappatura di associazioni tra entity (one-to-one)

4. Tabella di join

```
@Entity  
public class Shipment {
```

```
    @OneToOne(fetch = FetchType.LAZY)
```

```
    @JoinTable(  
        name = "ITEM_SHIPMENT",  
        joinColumns =  
            @JoinColumn(name = "SHIPMENT_ID"),  
        inverseJoinColumns =  
            @JoinColumn(name = "ITEM_ID",  
                nullable = false,  
                unique = true)  
    )  
    protected Item auction;  
    public Shipment() {  
    }  
    public Shipment(Item auction) {  
        this.auction = auction;  
    }  
    // ...  
}
```

← Required!

← Defaults to ID

← Defaults to AUCTION_ID

Item-shipment non
ha una classe Java
corrispondente

Mappatura di associazioni tra entity (one-to-one)

4. Tabella di join

```
Shipment someShipment = new Shipment();  
em.persist(someShipment);  
  
Item someItem = new Item("Some Item");  
em.persist(someItem);  
  
Shipment auctionShipment = new Shipment(someItem);  
em.persist(auctionShipment);
```

Mappatura di associazioni tra entity (many-to-one) Bags

Bags più efficienti per realizzare una one-to-many bidirezionale: puoi aggiungere elementi alla Collection in modo lazy, senza interrogare il db, perché non devi controllare se ci sono duplicati (set) o mantenere un indice (list)

```
@Entity
public class Item {

    @OneToMany(mappedBy = "item")
    public Collection<Bid> bids = new ArrayList<>();

    // ...
}

Item someItem = new Item("Some Item");
em.persist(someItem);

Bid someBid = new Bid(new BigDecimal("123.00"), someItem);
someItem.getBids().add(someBid);
someItem.getBids().add(someBid);
em.persist(someBid);
assertEquals(someItem.getBids().size(), 2);
```

← No persistent effect!

Mappatura di associazioni tra entity (many-to-one) Bags

Se carichi la collection dal db, trovi giustamente un solo oggetto:

```
Item item = em.find(Item.class, ITEM_ID);  
assertEquals(item.getBids().size(), 1);
```

Un add() non forza l'inizializzazione della collection:

```
Item item = em.find(Item.class, ITEM_ID);  
  
Bid bid = new Bid(new BigDecimal("456.00"), item);  
item.getBids().add(bid);  
em.persist(bid);
```

← **No SELECT!**

Mappatura di associazioni tra entity (many-to-one) List

Sicuri di voler usare una List?

```
@Entity
public class Item {

    @OneToMany
    @JoinColumn(
        name = "ITEM_ID",
        nullable = false
    )
    @OrderColumn(
        name = "BID_POSITION",
        nullable = false
    )
    public List<Bid> bids = new ArrayList<>();
    // ...
}
```

← Defaults to BIDS_ORDER

Mappatura di associazioni tra entity (many-to-one)

List

| ID | ITEM_ID | BID_POSITION | AMOUNT |
|----|---------|--------------|--------|
| 1 | 1 | 0 | 99.00 |
| 2 | 1 | 1 | 100.00 |
| 3 | 1 | 2 | 101.00 |
| 4 | 2 | 0 | 4.99 |

```
@Entity
public class Bid {

    @ManyToOne
    @JoinColumn(
        name = "ITEM_ID",
        updatable = false, insertable = false
    )
    @NotNull
    protected Item item;

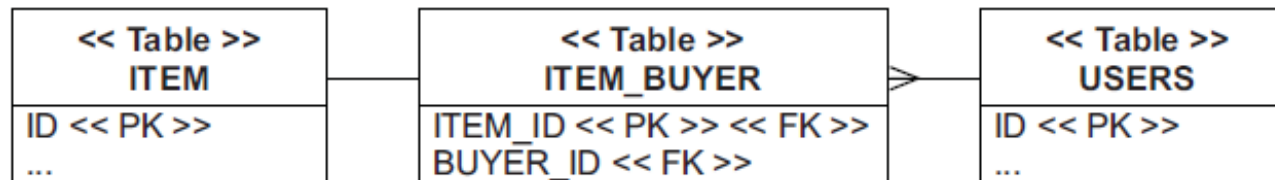
    // ...
}
```

← Disable writing!

← For schema generation

Mappatura di associazioni tra entity (many-to-one) Join Table

Consente di avere un'associazione many-to-one opzionale



Mappatura di associazioni tra entity (many-to-one) Join Table

```
@Entity
@Table(name = "USERS")
public class User {

    @OneToMany(mappedBy = "buyer")
    protected Set<Item> boughtItems = new HashSet<Item>();

    // ...
}
```

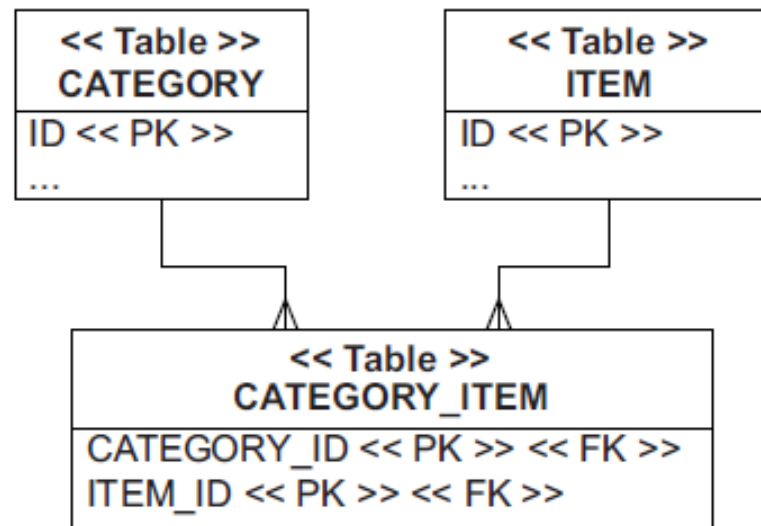
```
@Entity
public class Item {

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinTable(
        name = "ITEM_BUYER",
        joinColumns =
            @JoinColumn(name = "ITEM_ID"),
        inverseJoinColumns =
            @JoinColumn(nullable = false)
    )
    User buyer;
```

← Defaults to ID

← Defaults to BUYER_ID

Mappatura di associazioni tra entity (many-to-many) 1. link table



Mappatura di associazioni tra entity (many-to-many) 1. link table

```
@Entity
public class Category {

    @ManyToMany(cascade = CascadeType.PERSIST)
    @JoinTable(
        name = "CATEGORY_ITEM",
        joinColumns = @JoinColumn(name = "CATEGORY_ID"),
        inverseJoinColumns = @JoinColumn(name = "ITEM_ID")
    )
    protected Set<Item> items = new HashSet<Item>();

    // ...
}

@Entity
public class Item {

    @ManyToMany(mappedBy = "items")
    protected Set<Category> categories = new HashSet<Category>();

    // ...
}
```

Mappatura di associazioni tra entity (many-to-many) 1. link table

```
Category someCategory = new Category("Some Category");
Category otherCategory = new Category("Other Category");

Item someItem = new Item("Some Item");
Item otherItem = new Item("Other Item");

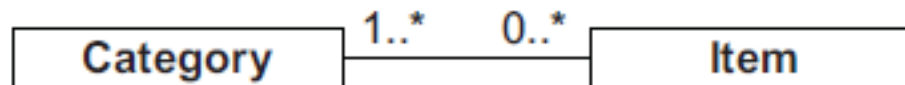
someCategory.getItems().add(someItem);
someItem.getCategories().add(someCategory);

someCategory.getItems().add(otherItem);
otherItem.getCategories().add(someCategory);

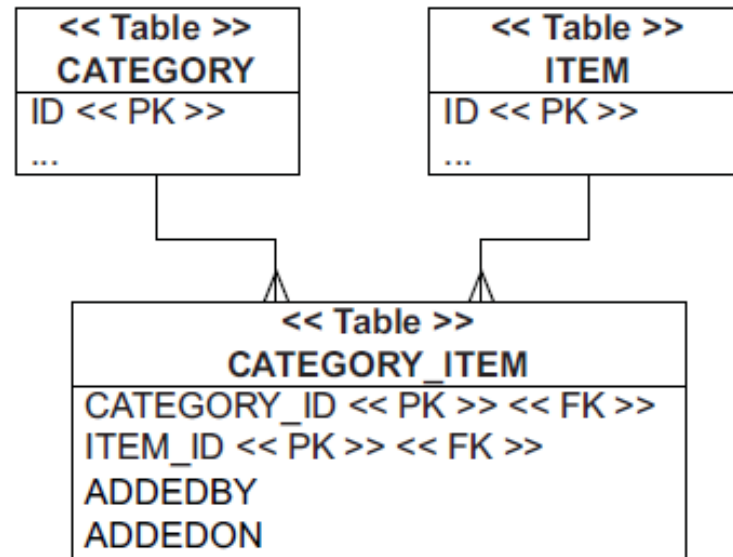
otherCategory.getItems().add(someItem);
someItem.getCategories().add(otherCategory);

em.persist(someCategory);
em.persist(otherCategory);
```


Mappatura di associazioni tra entity (many-to-many) 2.entity come link table



Many-to-many come 2
many-to-one



Mappatura di associazioni tra entity (many-to-many) 2.entity come link table

```
@Entity
@Table(name = "CATEGORY_ITEM")
@org.hibernate.annotations.Immutable
public class CategorizedItem {

    @Embeddable
    public static class Id implements Serializable {

        @Column(name = "CATEGORY_ID")
        protected Long categoryId;

        @Column(name = "ITEM_ID")
        protected Long itemId;

        public Id() {
        }

        public Id(Long categoryId, Long itemId) {
            this.categoryId = categoryId;
            this.itemId = itemId;
        }
    }
}
```

← 1 Declares class immutable

← 2 Encapsulates composite key

Mappatura di associazioni tra entity (many-to-many) 2.entity come link table

```
public boolean equals(Object o) {  
    if (o != null && o instanceof Id) {  
        Id that = (Id) o;  
        return this.categoryId.equals(that.categoryId)  
            && this.itemId.equals(that.itemId);  
    }  
    return false;  
}  
  
public int hashCode() {  
    return categoryId.hashCode() + itemId.hashCode();  
}  
}
```

```
@EmbeddedId  
protected Id id = new Id();  
  
@Column(updatable = false)  
@NotNull  
protected String addedBy;
```

← 3 Maps identifier property
and composite key
columns

← 4 Maps username

Mappatura di associazioni tra entity (many-to-many) 2.entity come link table

```
@Column(updatable = false)
@NotNull
protected Date addedOn = new Date();

@ManyToOne
@JoinColumn(
    name = "CATEGORY_ID",
    insertable = false, updatable = false)
protected Category category;

@ManyToOne
@JoinColumn(
    name = "ITEM_ID",
    insertable = false, updatable = false)
protected Item item;

public CategorizedItem(
    String addedByUsername,
    Category category,
    Item item) {

    this.addedBy = addedByUsername;
    this.category = category;
    this.item = item;

    this.id.categoryId = category.getId();
    this.id.itemId = item.getId();

    category.getCategorizedItems().add(this);
    item.getCategorizedItems().add(this);
}

// ...
}
```

← 5 Maps timestamp

← 6 Maps category

← 7 Maps item

← 8 Constructs CategorizedItem

Sets fields

Sets identifier values

← Guarantees referential integrity if made bidirectional

Mappatura di associazioni tra entity (many-to-many) 2.entity come link table

Per avere navigazione bidirezionale:

@Entity

```
public class Category {  
  
    @OneToMany(mappedBy = "category")  
    protected Set<CategorizedItem> categorizedItems = new HashSet<>();  
  
    // ...  
}
```

@Entity

```
public class Item {  
  
    @OneToMany(mappedBy = "item")  
    protected Set<CategorizedItem> categorizedItems = new HashSet<>();  
  
    // ...  
}
```

Mappatura di associazioni tra entity (many-to-many) 2.entity come link table

```
Category someCategory = new Category("Some Category");
Category otherCategory = new Category("Other Category");
em.persist(someCategory);
em.persist(otherCategory);

Item someItem = new Item("Some Item");
Item otherItem = new Item("Other Item");
em.persist(someItem);
em.persist(otherItem);

CategorizedItem linkOne = new CategorizedItem(
    "johndoe", someCategory, someItem
);

CategorizedItem linkTwo = new CategorizedItem(
    "johndoe", someCategory, otherItem
);

CategorizedItem linkThree = new CategorizedItem(
    "johndoe", otherCategory, someItem
);

em.persist(linkOne);
em.persist(linkTwo);
em.persist(linkThree);
```

Mappatura di chiavi primarie naturali

chiave primaria semplice

```
User user = new User("johndoe");  
em.persist(user);
```

```
@Entity  
@Table(name = "USERS")  
public class User {  
  
    @Id  
    protected String username;  
  
    protected User() {  
    }  
  
    public User(String username) {  
        this.username = username;  
    }  
  
    // ...  
}
```

Mappatura di chiavi primarie naturali

chiave primaria composta

La tabella USERS ha una chiave composta dalle colonne USER-NAME e DEPARTMENTNR

@Embeddable

```
public class UserId implements Serializable {
```

```
    protected String username;
```

```
    protected String departmentNr;
```

```
    protected UserId() {  
    }
```

```
    public UserId(String username, String departmentNr) {  
        this.username = username;  
        this.departmentNr = departmentNr;  
    }
```

@Override

```
    public boolean equals(Object o) {  
        if (this == o) return true;
```

← **1** @Embeddable, Serializable class

← **2** Automatically NOT NULL

← **3** Protected constructor

← **4** Public constructor

← **5** Overrides equals() and hashCode()

Mappatura di chiavi primarie naturali

chiave primaria composta

```
    if (o == null || getClass() != o.getClass()) return false;
    UserId userId = (UserId) o;
    if (!departmentNr.equals(userId.departmentNr)) return false;
    if (!username.equals(userId.username)) return false;
    return true;
}

@Override
public int hashCode() {
    int result = username.hashCode();
    result = 31 * result + departmentNr.hashCode();
    return result;
}

// ...
}
```

5 Overrides equals() and hashCode()

Mappatura di chiavi primarie naturali

chiave primaria composta

Per utilizzare la chiave composta si usa poi @EmbeddedId

```
@Entity
@Table(name = "USERS")
public class User {

    @EmbeddedId
    protected UserId id;

    public User(UserId id) {
        this.id = id;
    }

    // ...
}
```

← Optional: @AttributeOverrides

Mappatura di chiavi primarie naturali

chiave primaria composta

Users adesso ha una chiave composta:

| << Table >> USERS | |
|-----------------------|--|
| USERNAME << PK >> | |
| DEPARTMENTNR << PK >> | |
| ... | |

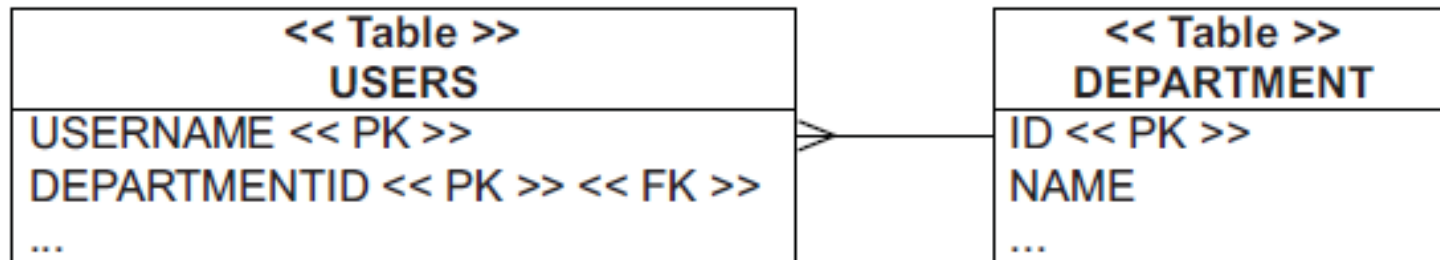
Per salvare istanza:

```
UserId id = new UserId("johndoe", "123");  
User user = new User(id);  
em.persist(user);
```

Per caricarla:

```
UserId id = new UserId("johndoe", "123");  
User user = em.find(User.class, id);  
assertEquals(user.getId().getDepartmentNr(), "123");
```

Mappatura di chiavi primarie naturali chiave esterna in chiave primaria composta



```
@Embeddable
public class UserId implements Serializable {

    protected String username;

    protected Long departmentId;

    // ...

}
```

Mappatura di chiavi primarie naturali chiave esterna in chiave primaria composta

1. Utilizzo di @MapsId (serve ad ignorare il valore di UserId#departmentId quando si salva un'istanza di User... deve usare l'id di Department assegnato a User#department)

```
@Entity
@Table(name = "USERS")
public class User {

    @EmbeddedId
    protected UserId id;

    @ManyToOne
    @MapsId("departmentId")
    protected Department department;

    public User(UserId id) {
        this.id = id;
    }

    // ...
}
```

Mappatura di chiavi primarie naturali chiave esterna in chiave primaria composta

```
Department department = new Department("Sales");  
em.persist(department);
```

```
UserId id = new UserId("johndoe", null);
```

← Null?

```
User user = new User(id);
```

```
user.setDepartment(department);
```

← Required

```
em.persist(user);
```

```
UserId id = new UserId("johndoe", DEPARTMENT_ID);
```

```
User user = em.find(User.class, id);
```

```
assertEquals(user.getDepartment().getName(), "Sales");
```

Mappatura di chiavi primarie naturali chiave esterna in chiave primaria composta

In alternativa:

```
@Entity
@Table(name = "USERS")
public class User {

    @EmbeddedId
    protected UserId id;

    @ManyToOne
    @JoinColumn(
        name = "DEPARTMENTID",
        insertable = false, updatable = false
    )
    protected Department department;

    public User(UserId id) {
        this.id = id;
    }




    // ...
}
```

Defaults to DEPARTMENT_ID

Make it read-only

Non posso più usare `someUser.setDepartment()`!
DEPARTMENTID viene settato da `UserId#departmentId`

Mappatura di chiavi primarie naturali chiave esterna in chiave primaria composta

```
Department department = new Department("Sales");  
em.persist(department);  Assigns primary key value  
  
UserId id = new UserId("johndoe", department.getId());  Required  
  
User user = new User(id);  
em.persist(user);  
  
assertNull(user.getDepartment());  Careful!
```

Department non è stato settato!

Mappatura di chiavi primarie naturali chiave esterna in chiave primaria composta

Department popolato al load dell'istanza:

```
UserId id = new UserId("johndoe", DEPARTMENT_ID);  
User user = em.find(User.class, id);  
assertEquals(user.getDepartment().getName(), "Sales");
```

Mappatura di chiavi primarie naturali chiave esterna in chiave primaria composta

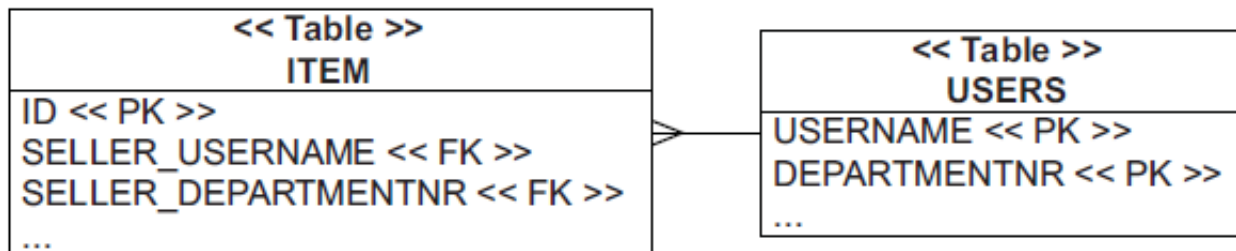
Si può forzare come instanziare lo User nel costruttore:

```
@Entity
@Table(name = "USERS")
public class User {

    public User(String username, Department department) {
        if (department.getId() == null)
            throw new IllegalStateException(
                "Department is transient: " + department
            );
        this.id = new UserId(username, department.getId());
        this.department = department;
    }

    // ...
}
```

Mappatura di chiavi primarie naturali chiave esterna in chiave primaria composta

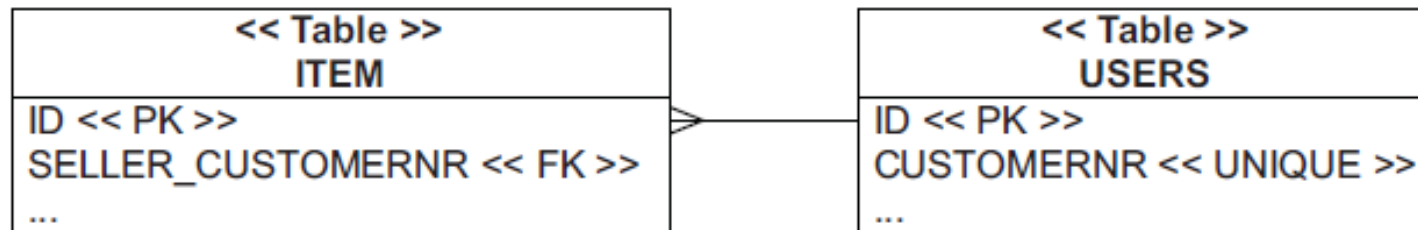


```
@Entity
public class Item {

    @NotNull
    @ManyToOne
    @JoinColumns({
        @JoinColumn(name = "SELLER_USERNAME",
                    referencedColumnName = "USERNAME"),
        @JoinColumn(name = "SELLER_DEPARTMENTNR",
                    referencedColumnName = "DEPARTMENTNR")
    })
    protected User seller;

    // ...
}
```

Mappatura di chiavi primarie naturali chiave esterna che non riferenzia chiave primaria



```
@Entity
@Table(name = "USERS")
public class User implements Serializable {

    @Id
    @GeneratedValue(generator = Constants.ID_GENERATOR)
    protected Long id;

    @NotNull
    @Column(unique = true)
    protected String customerNr;

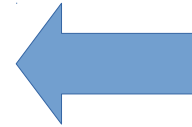
    // ...
}
```

Mappatura di chiavi primarie naturali chiave esterna che non riferenzia chiave primaria

```
@Entity
public class Item {

    @NotNull
    @ManyToOne
    @JoinColumn(
        name = "SELLER_CUSTOMERNR",
        referencedColumnName = "CUSTOMERNR"
    )
    protected User seller;

    // ...
}
```



Di default punta alla
chiave primaria
della tabella target