RelationShip Mapping

@OneToOne — *One Entity ↔ One Entity*

Meaning

Each record in one table is associated with **exactly one record** in another table.

Example:
Each `User` has exactly **one Profile**.

💻 Code Example

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

    private String address;
    private String phone;
}

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

    private String username;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "profile_id") // foreign key in users table
    private UserProfile profile;
}
```

Database Structure

users	user_profiles
id	id
username	address
profile_id →	phone

Notes

- `@JoinColumn` defines which column holds the **foreign key**.
- `cascade = CascadeType.ALL` ensures profile is saved/deleted with user.
- You can make it **bidirectional** by adding `@OneToOne(mappedBy = "profile")` in `UserProfile`.

2️⃣ @OneToMany — *One Entity ↔ Many Entities*

Meaning

One record in the parent table can have **multiple child records** in another table.

Example:
A `User` can have multiple `Wallets`.

💻 Code Example

```
@Entity
@Table(name = "wallets")
public class Wallet {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private double balance;

    @ManyToOne
    @JoinColumn(name = "user_id") // foreign key in wallets table
    private User user;
}
```

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

    private String username;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<Wallet> wallets;
}
```

📊 Database Structure

users	wallets
id	id
username	balance
	user_id →

Notes

- `@OneToMany(mappedBy = "user")` — "mappedBy" means `Wallet` owns the relationship.
- `@ManyToOne` on the child (`Wallet`) defines the **foreign key**.
- `orphanRemoval = true` deletes wallets if the user is removed.

3️⃣ `@ManyToOne` — *Many Entities ↔ One Entity*

Meaning

Many records in one table are associated with **a single record** in another table.
This is simply the **inverse of `@OneToMany`**.

Example:
Many `Transactions` belong to one `Wallet`.

Code Example

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

    private double amount;
    private String type;

    @ManyToOne
    @JoinColumn(name = "wallet_id") // foreign key in transactions
table
    private Wallet wallet;
}
```

Database Structure

wallets	transactions
id	id
balance	amount
	wallet_id →

Notes

- This annotation is used on the **child entity** (the “many” side).
- It’s the **owning side** of the relationship.
- Always use `@JoinColumn` to define the foreign key column.

---

4 `@ManyToMany` — *Many Entities* ↔ *Many Entities*

Meaning

Multiple records in one table relate to multiple records in another table.
Handled via a **join (bridge) table**.

Example:
A `User` can have multiple `Roles`, and a `Role` can belong to multiple `Users`.

Code Example

```
@Entity
@Table(name = "roles")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String roleName;

    @ManyToMany(mappedBy = "roles")
    private Set<User> users = new HashSet<>();
}

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

    private String username;

    @ManyToMany
    @JoinTable(
        name = "user_roles", // join table name
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Set<Role> roles = new HashSet<>();
}
```

Database Structure

users	roles	user_roles (join table)
id	id	user_id
username	role_name	role_id

Notes

- `@JoinTable` defines the **bridge table** (`user_roles`).
- `joinColumns` = column for this entity (`user_id`).
- `inverseJoinColumns` = column for the other entity (`role_id`).
- Use `Set` instead of `List` to avoid duplicates.

Quick Comparison Table

Relationship	Description	Example	Foreign Key Location
`@OneToOne`	One ↔ One	User → Profile	One side
`@OneToMany`	One ↔ Many	User → Wallets	Many side (Wallet)
`@ManyToOne`	Many ↔ One	Transaction → Wallet	Many side (Transaction)
`@ManyToMany`	Many ↔ Many	User ↔ Role	Join table (user_roles)

Tips for Real Projects (like your Digital Wallet)

- Always identify **owning side** (the one with `@JoinColumn`).
- Use `cascade = CascadeType.ALL` if you want changes in parent to reflect on children.

- Use `FetchType.LAZY` for performance — load related entities **only when needed**.
- To prevent infinite JSON loops in REST APIs, use:
    - `@JsonManagedReference` on the parent side
    - `@JsonBackReference` on the child side
