What are Cascade Operations in JPA?

Cascade in JPA means that certain operations performed on one entity (the **parent**) will automatically **propagate** to its related entities (the **children**).

This is controlled using the `cascade` attribute in relationship annotations like `@OneToOne`, `@OneToMany`, etc.

✅ Example: Cascade in a Digital Wallet Project

Let's say you have two entities:

```
@Entity
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 = new ArrayList<>();

    // getters, setters
}

@Entity
public class Wallet {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String walletName;
    private BigDecimal balance;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
```

```
    // getters, setters
}
```

📘 Explanation:

Here we used:

`cascade = CascadeType.ALL`

This means that when we perform an operation on `User`, the same operation is cascaded to all its `Wallet` entities.

⚡ CascadeType Options:

Cascade Type	Description	Example
`PERSIST`	When you save (persist) a parent, all its children are also saved	`userRepository.save(user)` will also save all wallets
`MERGE`	When you update a parent, its children are updated too	`entityManager.merge(user)` updates wallets
`REMOVE`	When you delete a parent, its children are deleted	`userRepository.delete(user)` removes wallets
`REFRESH`	Refreshes parent and all child entities from the database	Re-syncs both from DB
`DETACH`	Detaches both parent and children from persistence context	Removes from Hibernate cache
`ALL`	Applies all of the above	Most commonly used

✅ Example:

```
User user = new User();
user.setUsername("Shubham");

Wallet w1 = new Wallet("Wallet 1", new BigDecimal("1000"), user);
Wallet w2 = new Wallet("Wallet 2", new BigDecimal("2000"), user);
```

```
user.getWallets().add(w1);
user.getWallets().add(w2);

userRepository.save(user); // 👈 This automatically saves both wallets because of CascadeType.ALL
```

Without cascade, you'd need to manually save each wallet:

```
walletRepository.save(w1);
walletRepository.save(w2);
```

🧩2️⃣ What is Orphan Removal?

`orphanRemoval = true` ensures that if a **child entity** is removed from the parent's collection, it's also **deleted from the database**.

```
user.getWallets().remove(w1);
userRepository.save(user);
```

👉 The removed wallet `w1` will be **deleted** from the database automatically.

⚙️3️⃣ Fetch Strategies (LAZY vs EAGER)

The `fetch` attribute controls **when related data is loaded** from the database.

🔹 `FetchType.LAZY` (Default for `@OneToMany`)

- Related entities are **not loaded immediately**.
- They are loaded **only when accessed** (on demand).
- Uses **lazy loading** → better performance.

```
@OneToMany(mappedBy = "user", fetch = FetchType.LAZY)
private List<Wallet> wallets;
```

✅ **Example:**

```
User user = userRepository.findById(1L).get();
// Only user data is loaded here

List<Wallet> wallets = user.getWallets(); // Hibernate loads wallets
here (lazy)
```

🔥 **Advantages:**

- Better performance (only loads when needed)
- Reduces unnecessary joins

⚠️ **Drawback:**
If you access a lazy field **outside of a transaction**, you'll get:

```
LazyInitializationException
```

-

🔹 `FetchType.EAGER`

- Related entities are **loaded immediately** along with the parent.
- Performs an **inner join** under the hood.

```
@OneToMany(mappedBy = "user", fetch = FetchType.EAGER)
private List<Wallet> wallets;
```

✅ **Example:**

```
User user = userRepository.findById(1L).get();
// Both user and wallet data are loaded immediately
```

⚠️ **Drawbacks:**

- Slower queries (extra joins)
- Memory overhead if there are many child records

- Circular loading issues (if two entities eagerly load each other)

⚖️ Recommended Best Practices

Relationship	Default Fetch	Recommended
`@ManyToOne`	EAGER	✅ LAZY
`@OneToMany`	LAZY	✅ LAZY
`@OneToOne`	EAGER	✅ LAZY (usually better)
`@ManyToMany`	LAZY	✅ LAZY

✅ Example: Combining Cascade and Fetch

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

This means:

- When a user is saved/deleted → cascade applies to wallets
- Wallets are fetched **only when needed**
- If a wallet is removed from the list → it's also deleted (orphanRemoval)

🧠 Summary

Concept	Description	Example
Cascade	Propagates parent operations to children	`CascadeType.ALL`
Orphan Removal	Deletes child when removed from parent list	`orphanRemoval = true`

FetchType.LAZY	Loads child only when accessed	Default for collections
FetchType.EAGER	Loads child immediately	Should be used sparingly

---