Java Backend Development Live-85

lecture-13

Agenda

- JPA Transactions
- EntityManager
- Entity Lifecycle
- Hibernate Cache
- Minor Project-1
  - Requirement finalization
  - Low Level Design

JPA Mapping annotation

@OneToOne: Branch <--> Location

```
@OneToOne(  cascade = CascadeType.ALL)
  @JoinColumn(name = "locationId")
  private Location location;
```

@JoinColumn & **@OneToOne** should be mappedBy attribute when foreign key is held by one of the entities.

@ManyToOne: Employee <--> Branch

```
@ManyToOne
  @JoinColumn(name = "branchId")
  private Branch branch;
```

Cascading: When we perform some action on the target entity, the same action will be applied to the associated entity.

@OneToMany: Branch <--> Employee

```
@OneToMany(mappedBy = "branch", fetch = FetchType.EAGER)
 private Set<Employee> employees;
```

FetchType: EAGER, LAZY

LAZY = fetch when needed in a Transaction

EAGER = fetch immediately

JPA Transactions

Spring 3.1 introduces the **@EnableTransactionManagement** annotation that we can use in a **@Configuration** class to enable transactional support

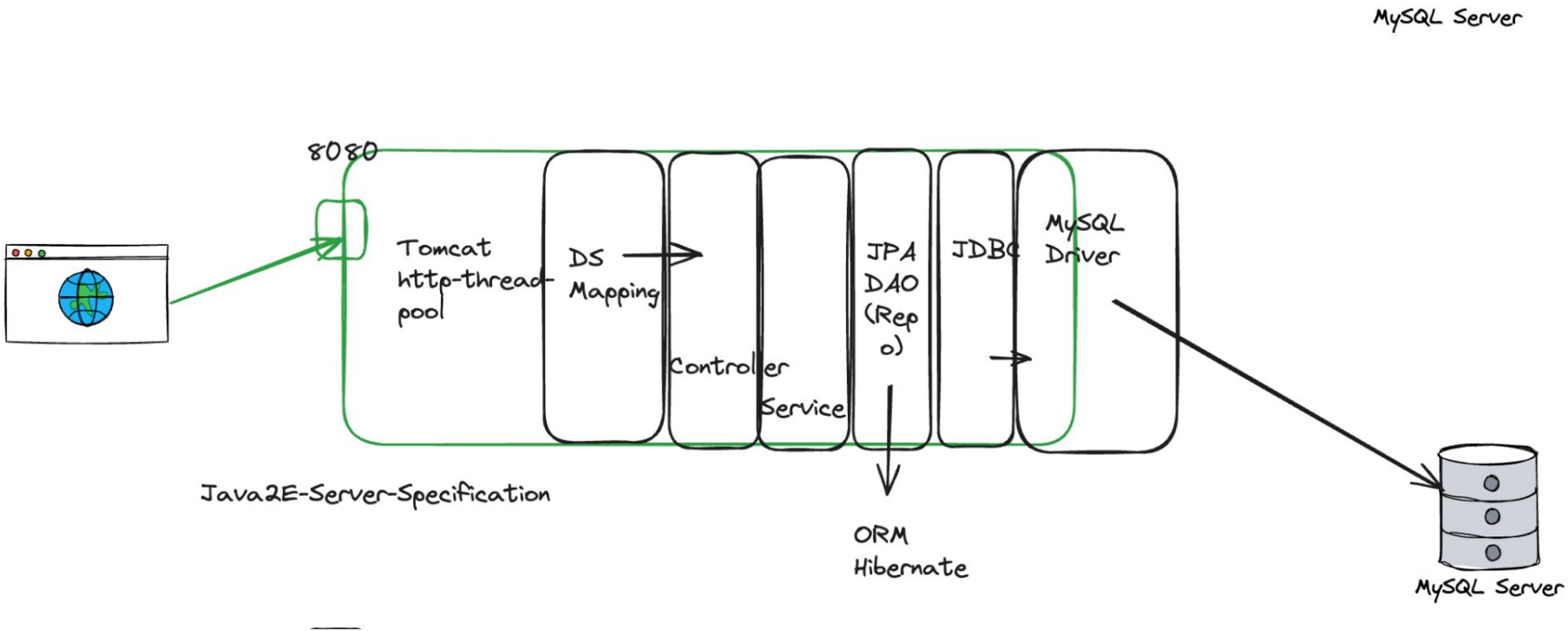
@Transactional either at the class or method level

By default, rollback happens for runtime, unchecked exceptions only. The checked exception does not trigger a rollback of the transaction. We can configure this behavior with the **rollbackFor** and annotation parameters.

At a high level, Spring creates **proxies** for all the classes annotated with @Transactional, either on the class or on any of the methods. The proxy allows the framework to inject transactional logic before and after the running method, mainly for starting and committing the transaction.

In the programmatic approach, we rollback the transactions using **TransactionAspectSupport**,

Request Flow

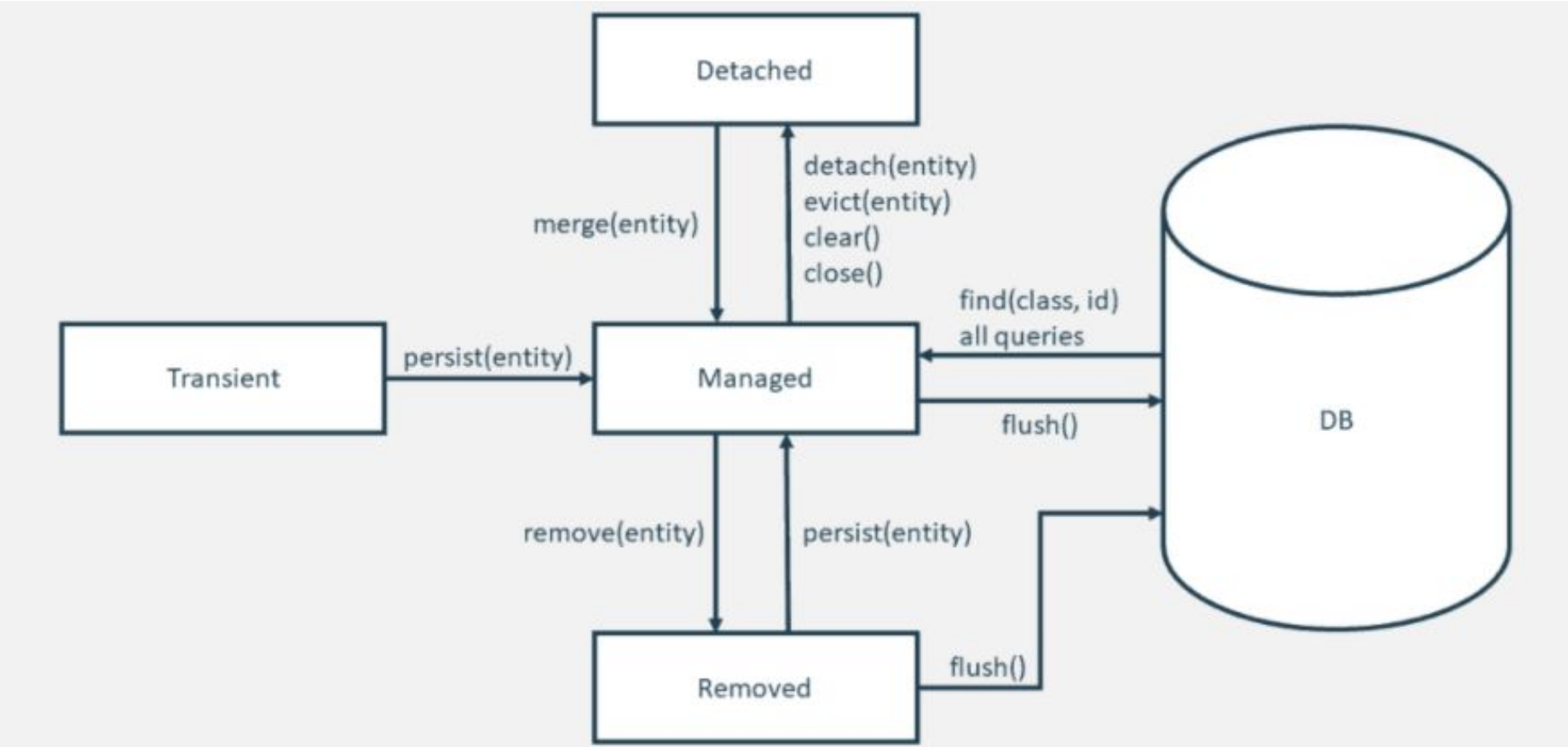


EntityManager

The EntityManager API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities.

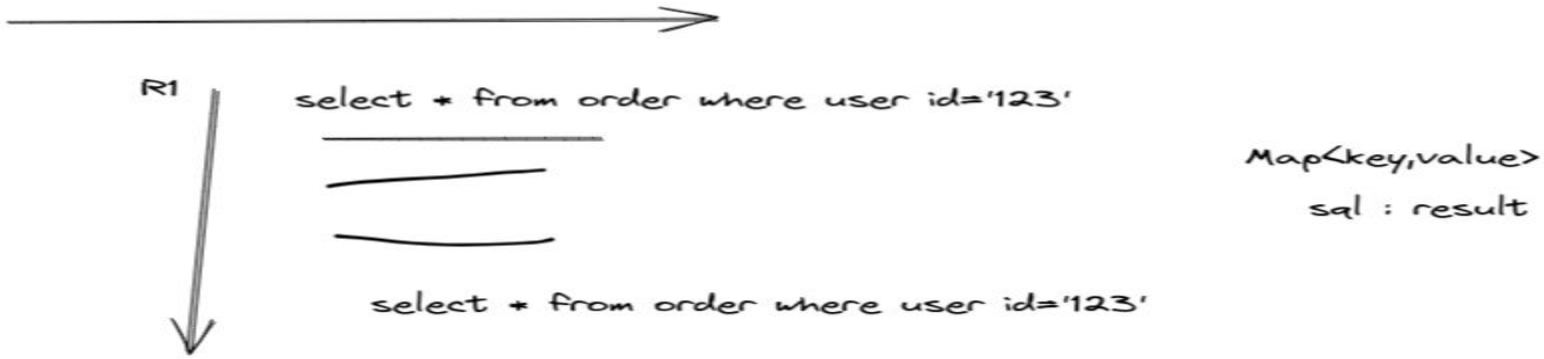
A **persistence context** is a set of entity instances in which for any persistent entity identity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle are managed.

Entity Lifecycle



Hibernate Cache

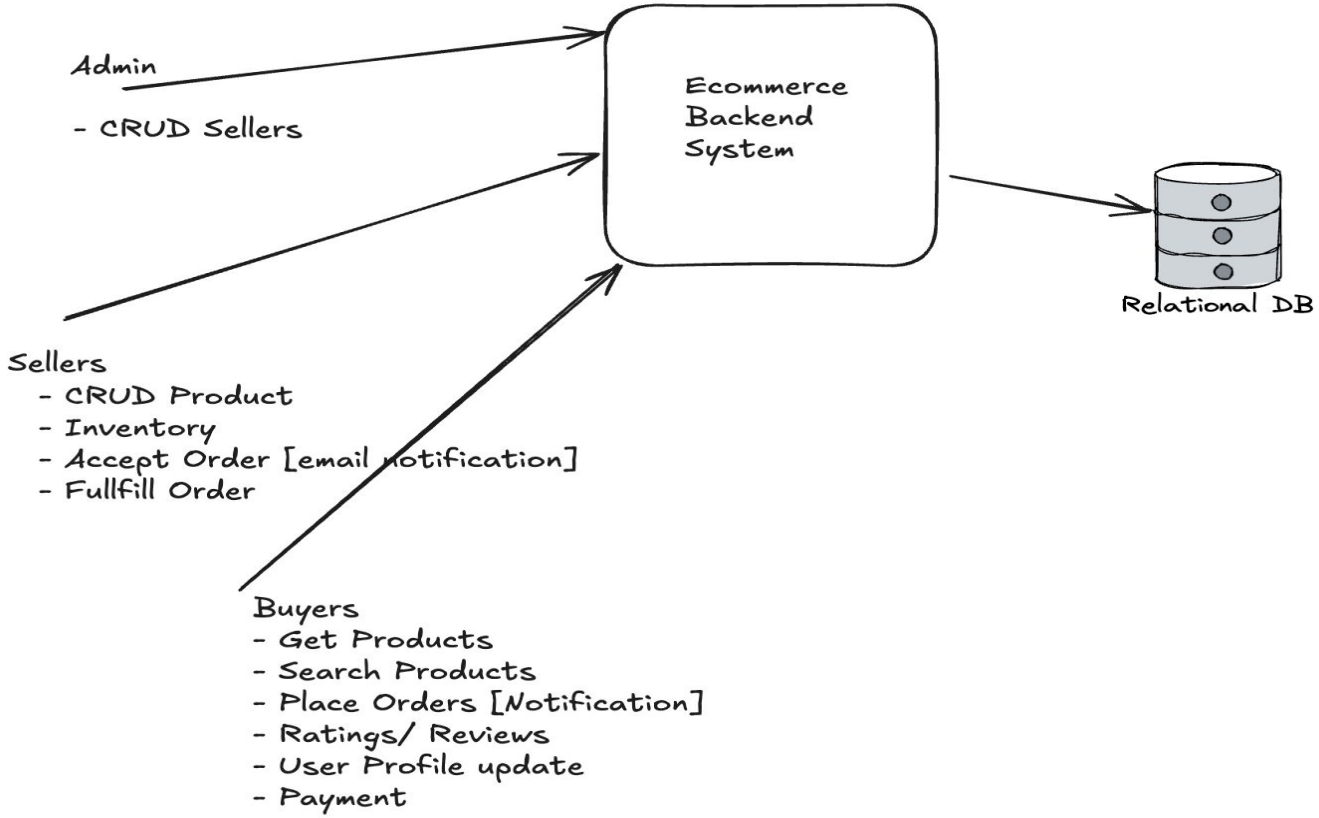
Purpose of Caching: Improve application performance by reducing the number of database hits.



Type of Cache in Hibernate

- **First-Level Cache (Session Cache):**
  - Associated with the Hibernate session.
  - Enabled by default and mandatory.
  - Caches objects within the same session.
- **Second-Level Cache (Session Factory Cache):**
  - Shared across sessions.
  - Must be explicitly configured.
  - Useful for caching frequently accessed data.
- **Query Cache:**
  - Caches the result of queries.
  - Works with the second-level cache.

Ecommerce Backend System



Functional Requirements

Admin

- CRUD Sellers

Sellers

- CRUD Products
- Accept Order
- Fulfil Order

Buyers

- Search Products
- Add product to Order
- Place Order
- Order History
- Profiles
- Payments

Tables

User (id,name, email, password, role[ADMIN, SELLER, CUSTOMER], created_at, updated_at, company_id)

Company (id, name, number, is_active, user_id, created_at, updated_at)

Product(id, name, description, price, stock, company_id, is_active, category_id, created_at, updated_at)

Order (id, user_id, total_amount, status(DRAFT,PLACED,ACCEPTED,SHIPPED,OFD), created_at, updated_at)

Order_Item (id, order_id, product_id, quantity, price)

Category(id, name, description)

API Request flow