### 1. Hibernate-Specific Annotations

Hibernate offers additional annotations that go beyond the standard JPA annotations. These can be used to customize entity mappings and improve performance.

#### 1.1 Use Hibernate-Specific Annotations

**1.1.1** @BatchSize

The @BatchSize annotation can be used to control the number of entities fetched in a single batch when using Hibernate's lazy loading.

**Example:**

import org.hibernate.annotations.BatchSize;

@Entity

@Table(name = "employees")

@BatchSize(size = 20)

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id", nullable = false)

private Department department;

// Getters and setters

}

**1.1.2** @Formula

The @Formula annotation allows you to define a computed column that Hibernate calculates on the fly.

**Example:**

import org.hibernate.annotations.Formula;

@Entity

@Table(name = "employees")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id", nullable = false)

private Department department;

@Formula("(select count(\*) from projects p where p.employee\_id = id)")

private Long projectCount;

// Getters and setters

}

**1.1.3** @Type

The @Type annotation is used to specify a custom Hibernate type for a particular field.

**Example:**

import org.hibernate.annotations.Type;

@Entity

@Table(name = "employees")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@Type(type = "org.hibernate.type.NumericBooleanType")

private Boolean active;

// Getters and setters

}

### 2. Configuring Hibernate Dialect and Properties

Configuring the Hibernate dialect and other properties helps optimize performance and ensure compatibility with the database.

#### 2.1 Configure Hibernate Dialect and Properties

**2.1.1** application.properties **Configuration**

Specify the Hibernate dialect and additional properties in your application.properties or application.yml.

**Example in** application.properties**:**

# Hibernate Dialect

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect

# Hibernate Properties

spring.jpa.properties.hibernate.hbm2ddl.auto=update

spring.jpa.properties.hibernate.show\_sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.properties.hibernate.use\_sql\_comments=true

spring.jpa.properties.hibernate.jdbc.batch\_size=30

spring.jpa.properties.hibernate.order\_inserts=true

spring.jpa.properties.hibernate.order\_updates=true

**2.1.2 Key Hibernate Properties:**

* hibernate.dialect: Specifies the SQL dialect for the database.
* hibernate.hbm2ddl.auto: Controls schema generation. Common values are update, create, create-drop, and validate.
* hibernate.show\_sql: Logs SQL statements.
* hibernate.format\_sql: Formats SQL statements for readability.
* hibernate.use\_sql\_comments: Adds comments to SQL statements.
* hibernate.jdbc.batch\_size: Defines the batch size for batch processing.
* hibernate.order\_inserts: Orders SQL insert statements by primary key to improve batch performance.
* hibernate.order\_updates: Orders SQL update statements to improve batch performance.

### 3. Batch Processing

Batch processing with Hibernate improves performance for bulk operations by grouping multiple operations into a single batch.

#### 3.1 Implement Batch Processing

**3.1.1 Configuring Batch Size**

Set the batch size in application.properties (as shown above) to control the number of statements sent in each batch.

**3.1.2 Batch Processing Example**

Use the batch size in your repository or service to perform bulk operations.

**Example Service Method:**

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

@Service

public class EmployeeService {

@PersistenceContext

private EntityManager entityManager;

@Transactional

public void batchInsertEmployees(List<Employee> employees) {

final int batchSize = 30;

for (int i = 0; i < employees.size(); i++) {

entityManager.persist(employees.get(i));

if (i % batchSize == 0 && i > 0) {

// Flush and clear the persistence context to prevent memory issues

entityManager.flush();

entityManager.clear();

}

}

}

}

**3.1.3 Batch Processing in Repository Methods**

You can also perform batch updates using repository methods.

**Example Repository Method:**

import org.springframework.data.jpa.repository.Modifying;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.CrudRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends CrudRepository<Employee, Long> {

@Modifying

@Query("UPDATE Employee e SET e.email = :email WHERE e.id = :id")

void updateEmailById(Long id, String email);

}