Hibernate ORM Mapping

* A Java class with fields is mapped to a database table through a Hibernate Framework
* Hibernate is a type of ORM
* Queries generated by hibernate get executed using JDBC
* JPA is a specification that defines how Java Objects should be mapped to relational database tables and how their relationships should be handled.
* Spring Data JPA automatically creates DAO (data access objects). We can perform CURD operations without writing queries through Spring Data JPA

Hibernate

* Hibernate is a powerful, high-performance Object-Relational Mapping (ORM) framework that is widely used with Java. It provides a framework for mapping an object-oriented domain model to a relation data base.
* Hibernate is one of the implementations of the Java Persistence API (JPA), which is a standard specification for ORM in Java.

JPA (Java Persistence API)

* JPA is a specification for object-relational mapping (ORM) in Java.
* It defines a set of interface and annotations for mapping Java objects to database tables and vice versa.
* JPA itself is just a set of guidelines and does not provide any implementation. The implementation of JPA is provided by ORM frameworks such a Hibernate, EclipseLink, and OpenJPA.

Summary

* JPA Provides a standard for ORM in Java applications, ensuring that developers can switch between different JPA providers without changing their code. Hibernate is a JPA Provider
* Hibernate is a specific implementation of JPA and a powerful ORM framework on its own. It offers additional features and optimizations beyond the JPA specification, making it a popular choice for ORM in Java Application.

Common Hibernate Configurations

* Spring.jpa.hibernate.ddl-auto=update/create/validate/create-drop/none
  + The setting spring.jpa.hibernate.ddl-auto in Spring Boot tells Hibernate how to **handle schema (table) generation and validation** when the application starts.

| **Value** | **Description** | **What it Does** |
| --- | --- | --- |
| none | 🚫 Do nothing | Hibernate **won’t touch** the schema — you manage everything manually |
| validate | 🔍 Validate schema only | Checks if the tables/columns **match** your entities. If not, it **throws an error** |
| update | 🛠️ Update schema | Hibernate will **alter the schema** to match entities (add missing tables/columns, not remove) |
| create | ✨ Create schema on every start | Drops and recreates tables **every time** the app starts |
| create-drop | 💣 Create schema at start, **drop it** at shutdown | Good for testing; creates fresh schema on start, drops it on close |

**When to Use What?**

| **Environment** | **Recommended Option** | **Why?** |
| --- | --- | --- |
| **Production** | none or validate | Never let Hibernate change your prod DB automatically |
| **Development** | update or create | Fast and convenient to auto-sync with your entity changes |
| **Testing** | create-drop | Ensures fresh schema every test run |

**🔹 spring.jpa.show-sql=true**

✅ **What it does:**

* Tells Hibernate to **print all the SQL queries** it's executing in the console/log.

📌 Example:

sql

CopyEdit

select \* from employee where id=1

🧠 **Why useful?**

* Helps you **debug** and understand what SQL Hibernate is generating behind the scenes.

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

✅ **What it does:**

* **Formats the printed SQL** nicely with indentation and line breaks.

📌 Without formatting:

sql

CopyEdit

select \* from employee where id=1

📌 With formatting:

sql

CopyEdit

select \*

from employee

where id = 1

🧠 **Why useful?**

* Makes the SQL more **readable**, especially when queries get long.

**🔹 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect**

✅ **What it does:**

* Tells Hibernate which **SQL dialect** (flavor) to use for generating database-specific queries.

📌 Why?

* Different databases have slightly different SQL syntax.
* This setting ensures Hibernate **uses MySQL-compatible SQL**.

🧠 **Example:**

* In PostgreSQL: LIMIT 10 OFFSET 20
* In MySQL: LIMIT 20, 10  
  → Hibernate uses this because of the MySQL dialect

🧱 Hibernate has different dialects for different DBs:

| **Dialect Class** | **For Database** |
| --- | --- |
| MySQL5Dialect | MySQL 5.x |
| MySQL8Dialect | MySQL 8.x |
| PostgreSQLDialect | PostgreSQL |
| Oracle12cDialect | Oracle 12c |
| SQLServerDialect | SQL Server |

**✅ Summary**

| **Setting** | **Purpose** |
| --- | --- |
| show-sql=true | Logs all SQL statements to console |
| format\_sql=true | Makes the logged SQL easy to read |
| hibernate.dialect=... | Makes sure SQL matches the database syntax |

Spring Boot SQL Initialization Properties

**spring.jpa.defer-datasource-initialization=true**

Tells Spring Boot to wait until JPA (Hibernate) sets up tables before executing SQL scripts like data.sql. Prevents errors if data.sql tries to insert into tables that aren’t created yet.

**spring.sql.init.mode=always**

Always runs SQL initialization scripts (like data.sql) regardless of the database type (embedded or not). Useful in development and production environments.

**spring.sql.init.data-locations=classpath:data.sql**

Specifies the location of the SQL script to be executed. In this case, Spring will run data.sql located in the resources folder (src/main/resources).

Key features of JPA

* Entity management: Defines how entities are persisted to the database
* Query Language: Provide JPQL for querying entities
* Transactions: Manages transactions, making it easier to handle database operations within a transactional context
* Entity Relationships: Supports defining relationships between entities (one to one, one to many, many to one, many to many)

Spring Data JPA

* Spring Data JPA is a part of the larger Spring Data Family.
* It builds on top of JPA, providing a higher-level and more convenient abstraction for data access.
* Spring Data JPA makes it easier to implement JPA-Based repositories by providing boilerplate code, custom query methods, and various utilities to reduce the amount of code you need to write.

Key features of Spring Data JPA

* Repository Abstraction: Provides a Repository interface with methods for common data access operations.
* Custom Query methods: Allows defining custom query methods by simply declaring method names.
* Pagination and Sorting: Offers built-in support for pagination and sorting
* Query Derivation: Automatically generates queries from method names

Rules for Method names

* The name of the query method must start with one of the following prefixes: find…By, read…By, query…by and get….by. Examples findByname, queryByname, getByname
* If we want to limit the number of returned query results, we can add the first or the top keyword before the first by word. Examples: findFirstByname, readFirst2Byname, findTop10byName
* If we want to select unique results, we have to add the distinct keyword before the first By word. Examples: findDistinctByName or findNameDistinctBy
* Combine property expression with AND and OR. Examples: findByNameOrDescription, findByNameAndDescription

Sorting with method queries

* @Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findAllByOrderByNamesAcs();

List<Employee> findAllByOrderByNamesDesc();

Sorting with the Sort class

* Sort Parameter in query Methods
* @Repository

Public interface EmployeeRepository extends JpaRepository<Emplyee,Long> {

List<Employee> findByDepartment(String department, Sort sort)

Using the Sort Class

Sort sort = Sort.by(Sort.Direction.ASC, sortField);

Sort sort = Sort.by(Sort.Order.asc(“name”),Sort.Order.desc(“salary”));

Pagination

* Page: A single chunk of data that contains a subset of the total dataset. It is an interface representing a page of data, including information about the total number of pages, total number of elements, and the current page’s data
* Pageable: An interface that provides pagination information such as page number, page size, and sorting options
* PageRequest: A concrete implementation of Pageable that provides methods to create pagination and sorting information.
* Using Pageable

Public interface UserRepository extends JpaRepository<User, Long> {

Page<User> findAll(Pageable pageable);

Page<User> findByLastName(String lastName, Pageable pageable)}

Creating Pageable instance:

* Pageable pageable = PageRequest.of(pageNumber, size, Sort.by(“lastName”).ascending());

Mappings

* One to one mapping
* Many to One mapping
* One to Many mapping

Key concepts of Mappings

* Entity: A class that is mapped to a database table
* Primary Key: A unique identifier for a database record
* Foreign Key: A field in one table that uniquely identifies a row in another table
* Cascade: A set of operations that can be propagated from a parent entity to a related entity
  + CascadeType.ALL: Propagates all operations (e.g., persist, merge, remove, refresh, detach) from the parent to the child entity.
  + CascadeType.PERSIST: Propagates the persist (save) operation
  + CascadeType.Merge: Propagates the merge operation
  + CascadeType.REMOVE: Propagates the remove (delete) operation
  + CascadeType.REFRESH: Propagates the refresh
  + CascadeType.DETACH: Propagates the detach operation
* Fetch type: Defines the strategy for fetching related entities. (eager or lazy loading)

Relationship Annotations

* @OneToOne
* @OneToMany
* @ManyToOne
* @ManyToMany
* @JoinColumn
* @JoinTable