DATE 14 03 22

Hibernate & JPA

★ Introduction to JPA →

- The java persistence API provides a specification for persisting, reading and managing data from your java object to your relational tables in the database.
- It specifies the set of rules and guidelines for developing interfaces that follow standards.
- It is part of the spring framework.

★ Journey from Spring JDBC to JPA →

1. Step-1 → Setting up a project with JDBC, JPA, H2 and web dependencies.
2. Step-2 → Launching up H2 console.
3. Step-3 → Creating a database table in H2.
4. Step-4 → Populate data into person table.
5. Step-5 → Implement findAll persons using spring JDBC query Method.
6. Step-6 → Execute the findAll Method using CommandLine Runner.
7. Step-7 → A quick review - JDBC vs spring JDBC.

DATE

(8) Step-8 → What's in the background?
Understanding Spring Boot AutoConfiguration

(9) Step-9 → Implementing findById spring JDBC Query Method

(10) Step-10 → Implementing deleteById spring JDBC update Method

(11) Step-11 → Implementing insert and update Spring JDBC Update Methods.

(12) Step-12 → Creating a custom Spring JDBC RowMapper

(13) Step-13 → Quick introduction to JPA.

(14) Step-14 → Defining Person Entity

(15) Step-15 → Implementing findById JPA repository Method

(16) Step-16 → Implementing insert and update JPA repository Methods.

(17) Step-17 → Implementing deleteById JPA repository Method.

(18) Step-18 → Implementing findAll using JPQL named Query.

DATE

★ Introduction to JPA and Hibernate in depth →

① Step-1 → Create a JPA project with H2 and spring boot.

② Step-2 → Create JPA entity course.

③ Step-3 → Create findById using JPA entity Manager.

④ Step-4 → Configuring application.properties to enable H2 console and logging.

⑤ Step-5 → Writing unit test for findById Method.

⑥ Step-6 → Writing a deleteById Method to delete an entity.

⑦ Step-7 → Writing unit test for deleteById Method.

⑧ Step-8 → Writing a save Method to update and insert an entity.

⑨ Step-9 → Writing unit test for save Method.

⑩ Step-10 → Quick review and debugging tips.

⑪ Step-11 → Playing with entity Manager.

DATE

(12) Step-12 → Entity Manager Methods - Clear and detach

(13) Step-13 → Entity Manager Methods - refresh

(14) Step-14 → A quick review of entity manager

(15) Step-15 → JPQL - Basics

- It is Java Persistence Query Language defined in JPA specifications. It is used to create queries against entities to store in a relational database.
- It is based on SQL syntax. But it won't affect the database directly.
- It can retrieve information or data using SELECT clause, can do bulk updates using UPDATE clause and DELETE clause. EntityManager.createQuery() API will support for querying language.

(16) Step-16 → JPA and Hibernate annotations - @Table

(17) Step-17 → JPA and Hibernate annotations - @Column

(18) Step-18 → JPA and Hibernate annotations - @UpdateTimestamp and @CreationTimestamp.

(19) Step-19 → JPA and Hibernate annotations - @NamedQuery and @NamedQueries

DATE

(20) Step-20 → Native queries - Basics

- It refers to actual sql queries (refferring to actual database objects). These queries are the sql statements which can be directly executed in database using a database client.
- Named SQL queries are defined using the @NamedNativeQuery annotation. This annotation may be placed on any entity and defines the name of the query as well as the query text. Like JPQL named queries, the name of the query must be unique within the persistance unit.

★ Establishing relationships with JPA and Hibernate - One To One →

(21) Step-21 → Entities and Relationships - An overview.

(22) Step-22 → Defining Entities - Student, Passport and Review.

Step-23 →

(23) Introduction to one to one relationship →

(24) Step-24 → OneToOne Mapping - Insert student with passport

(25) Step-25 → OneToOne Mapping - Retrieving student with passport and Eager Fetch.

DATE

(26) Step-26 → OneToOne Mapping - Lazy fetch

(27) Step-27 → Transaction, Entity Manager and Persistence Context.

(28) Step-28 → OneToOne Mapping - Bidirectional Relationship - Part 1

(29) Step-29 → OneToOne Mapping - Bidirectional Relationship - Part - 2

★ Establishing Relationships with JPA and Hibernate - OneToMany and ManyToMany →

(30) Step-30 → Part-1 → ManyToOne Mapping - Designing the database.

Part-2 → ManyToOne Mapping - Implementing the Mapping

(31) Step-31 → ManyToOne Mapping - Retrieving and inserting reviews for course.

(32) Step-32 → ManyToOne Mapping - Generalizing insert reviews

(33) Step-33 → ManyToOne Mapping - Wrapping up

DATE

(34) Step-34 → Many To Many Mapping - Table design

(35) Step-35 → Many To Many Mapping - Adding annotations on entities.

(36) Step-36 → Many To Many Mapping - Fixing two join tables problem.

(37) Step-37 → Many To Many Mapping - Customizing the join table.

(38) Step-38 → Many To Many Mapping - Insert data and write join query.

(39) Step-39 → Many To Many Mapping - Retrieve data using JPA relationships.

(40) Step-40 → Many To Many Mapping - Insert student and course

(41) Step-41 → Relationships between JPA entities - A SUMMARY.

★ Inheritance Hierarchies with JPA and Hibernate →

(42) Step-42 → Introduction to inheritance hierarchies and mappings.

DATE 15 03 22

- Hierarchial inheritance is one of the types of inheritance in java.
- Inheritance is a mechanism in which one class inherits or acquires all the other class's attributes and behaviours. The class inherits the attributes and behaviours called a parent or super or base class, and the class inherits the attributes and behaviours are called child or derived class.

(43) Step-43 → JPA inheritance hierarchies and mappings - Setting up entities

(44) Step-44 → JPA inheritance hierarchies and mappings - Setting up a repository.

(45) Step-45 → JPA inheritance hierarchies and mappings - Single table

(46) Step-46 → JPA inheritance hierarchies and mappings - Table per class.

(47) Step-47 → JPA inheritance hierarchies and mappings - Joined

(48) Step-48 → JPA inheritance hierarchies and mappings - Mapped Super Class.

DATE

(49) Step-49 → JPA "inheritance hierarchies and mapping - How to choose?

★ Queries with entities using JPQL →

(50) Step-50 → JPQL → Courses without students.

(51) Step-51 → JPQL - Courses with atleast 2 students and order by

(52) Step-52 → JPQL - Courses like 100 steps

(53) Step-53 → JPQL - using joins

★ Queries using java API - Criteria Queries →

(54) Step-54 → Criteria Query - Retrieving all courses.

(55) Step-55 → Criteria Query - Courses like 100 steps

(56) Step-56 → Criteria Query - Courses without students

(57) Step-57 → Criteria Query - Using joins

★ Transaction Management →

(58) Step-58 → Introduction to transaction management.

DATE

(59) Step-59 → Transaction Management - ACID properties

- A database transaction is a sequence of actions that are treated as a single unit of work. These actions should either complete entirely or take no effect at all. ~~Transaction Mang~~
- It is an important part of RDBMS-oriented enterprise application to ensure data integrity and consistency.

ACID Properties →

- Atomicity → A transaction should be treated as a single unit of operation, which means either the entire sequence of operations is successful or unsuccessful.

- Consistency → This represents the consistency of the referential integrity of the database, unique primary keys in tables, etc.

- Isolation → There may be many transaction processing with the same data set at the same time. Each transaction should be isolated from others to prevent data corruption.

- Durability → Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.

DATE

(60) Step-60 → Understanding dirty, phantom and non-repeatable reads.

(61) Step-61 → Understand 4 isolation levels.
- READ UNCOMMITTED.
- READ COMMITTED (protecting against dirty reads).
- REPEATABLE READ (protecting against dirty and non-repeatable reads).
- SERIALIZABLE (protecting against dirty, non-repeatable reads and phantom reads).

(62) Step-62 → Choosing b/w isolation levels

(63) Step-63 → Implementing transaction Management - 3 things to decide.

★ Spring data JPA and Spring Data REST →

(64) Step-64 → Introduction to spring data JPA
- It is a part of the larger spring data family, makes it easy to easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier to build spring-powered applications that use data access technologies.

(65) Step-65 → Testing the spring data JPA repository with findById.

DATE 17 03 22

(66) Step-66 → Spring data JPA repository - CRUD Methods

(67) Step-67 → Sorting using Spring data JPA repository.

(68) Step-68 → Pagination using spring data JPA repository

(69) Step-69 → Custom queries using spring data JPA repository.

(70) Step-70 → Spring Data REST.

- It is a part of the umbrella spring data project and makes it easy to build hypermedia-driven REST web services on top of Spring data repositories.
- It builds on top of spring data repositories, analyzes your application's domain model and exposes hypermedia-driven HTTP resources for aggregates contained in the model.

★ Caching with Hibernate and JPA →

(71) Step-71 → Introduction to caching.

- Caching structurally implies a temporary store to keep data for quicker access later on. Second level shared cache is an auxiliary technique, Mainly used in JPA to enhance performance.
- It is used especially during large in flow, out flow of data b/w the database and the application.

DATE

(72) Step-72 → Hibernate and JPA caching - First level cache.

- It is enabled by default and you do not need to do anything to get this functionality working. In fact, you cannot disable it even forcefully.

(73) Step-73 → Hibernate and JPA caching - Basics of second level cache with EhCache.

- It uses a common cache for all the session object of a session factory. It is useful if you have multiple session objects from a session factory.
- Diff. vendors have provided the implementation of second level cache -
  - EH Cache
  - OS Cache
  - Swarm Cache
  - JBoss Cache

(74) Step-74 → Hibernate and JPA Caching - Second level Cache part 2.

★ Hibernate and JPA tips →

(75) Step-75 → Hibernate tips - Hibernate soft deletes - @SQLDelete and @Where

(76) Step-76 → Hibernate soft deletes - Part 2

DATE

(77) Step-77 → JPA entity Life Cycle Methods

- before persist is called for a new entity - @PrePersist
- after persist is called for a new entity - @PostPersist
- before an entity is removed - @PreRemove
- after an entity has been deleted - @PostRemove
- before the update operation - @PreUpdate
- after an entity is updated - @PostUpdate
- after an entity has been loaded - @PostLoad

(78) Step-78 → Using embedded and embeddable with JPA.

(79) Step-79 → Using enums with JPA.

(80) Step-80 → JPA tip - be cautious with tostring Method implementations.

(81) Step-81 → JPA tip - When do you use JPA?

- JPA allows you to avoid writing DML in the database specific dialect of SQL. JPA allows you to load and save java objects and graphs without any DML language at all. When you do need to perform queries JPQL allows you to express the queries in terms of the java entities rather than the (native) SQL tables.

DATE

★ Performance tuning tips with hibernate & JPA →

(82) Step-82 → Performance tuning - Measure before tuning.

(83) Step-83 → Performance tuning - indexes.

(84) Step-84 → Performance tuning - use appropriate caching

(85) Step-85 → Performance tuning - Eager vs Lazy fetch

(86) Step-86 → Performance tuning - Avoid N+1 problems.

- Tips →

1. Fetching only the data you really need.
2. Open session in view and temporary session anti-patterns
3. Streaming pitfalls
4. Optimizing the no. of database roundtrips
5. Read-only queries
6. Statement caching
7. Statement batching
8. Connection Management
9. Logging
10. Mapping
11. Database-level processing
12. Caching
13. Query plan cache

2022/03/20 15:27