HIBERNATE

**Overall Overview**:

1. Spring jdbc to jpa.

1. World before jpa.

2. World after jpa introduced.

2. Simple jdbc application and mitigate it to jpa.

3. Basics of Spring, Spring Boot, Junit.

Almost 65% Java Projects use Spring and Spring boot

1. A big chunk of rest API, web services and microservices are being built with spring on spring boot.

2. Maven to manage dependencies and Git as a version Control

**Tools Necessary**

i) Eclipse -IDE

ii) Maven - dependency management

**Earlier Approaches**: JDBC on Spring JDBC

a)Lot of queries

1)create a query

2)Populate the right values into query.

3)fire it into database.

Problem arised: Queries become complex as application gets more complex and it is difficult to maintain.

Solution: JPA (JAVA PERSISTENCE API) - JPA creates it queries.

When the need arises to define your queries in jpa : you have three options

1. Jpql

2. Criteria query

3. Native Query

**This API requires**

1. What are the java classes or objects that you are creating - define your entities

2. How do they map to your tables - map fields to the columns in database.

3. Relationship among objects

4. JPA creates a query for you.

**Entities** - Object that we store in database

Spring Boot in 10 steps:

1)Enabling building production ready applications faster

2)Provides common non-functional features

-embedded servers

-metrics

-health checks

-externalized configuration

**Spring Boot**

1)does not generate any code

2)It i neither an application server nor a web server

**Features of spring boot**

1)Quick Starter projects with Auto-configuration

spring-boot-starter-web (consist spring-mvc, spring-core, validation)

spring-boot-starter-jpa (default implementation of jpa, hibernate and auto-configuration)

Using spring Initializer to create a Spring Boot Application

To signify something as a control use

**Creating a simple Rest Controller**

@RestController

\* Serves request

@GetMapping(“/url”)

\* To retrieve a list of values

Spring Boot AutoConfiguration

\* Spring context

\* Auto configuration

\* Component scan

**Spring JDBC to JPA**

Required dependencies - h2, jdbc, web, jpa

In application.properties set the below configuration:

spring.database.url = jdbc:h2:mem:testdb

spring.data.jpa.repositories.bootstrap-mode=default

spring.h2.console.enabled = true // to enable h2 console

spring.jpa.show-sql = true // to see the queries being executed

@Entity - to signify it is an entity

@Table(name=”tablename”) - table name to be referred in database

@Column(name=”columnname”) - column name to be referred in database

@id - Primary key

@GeneratedValue - to be automatically incremented

@PersistenceContext

All the operations that are performed on a specific or all entities, are stored in PersistenceContext

@EntityManager

EntityManager is the interface to PersistenceContext

Entity manager has many methods provided such as find, merge, persist, remove.

**JPA AND HIBERNATE**

1)Create entities

Make sure to have a default constructor

2) create a repository ( a method or action to be performed. Ex - findById)

3)create a sql file

4)When application would be launched( it implements a CommandLineRunner and run method would be launched up and tries to find the specified requirement)

5)Create a junit test for that method

@RunWith(SpringRunner.class) // launches a spring context in my Unit test

The spring context to launch is @SpringBootTest

@DirtiesContext - to leave the state of application in its original form.

Hibernate - sequence : it is used to insert the values by default

**EntityManager methods**

1. Flush - changes done until then will be sent out to database.

2. detach(obj) - changes of that particular object are no longer tracked by Entity manager

3. Clear - used to clear all objects that are managed by entity manager

4. Refresh - refresh of entity manager

5. Persist - insert

6. Merge - update

7. Remove - delete

8. Find - to find

9. createQuery - to create a query

**JPQL - Java Persistence Query Language**

We query from entities and we write our queries in the form jpql which is converted into sql queries by JPA implementation( i.e., Hibernate)

**Constraints on Columns**

nullable,unique,insertable,updatable,length,precision,scale

**Annotations**

@UpdateTimestamp - updates time

@CreationTimestamp - creates time

@NamedQuery(name=”query\_name”,query = “select var from Entity var)

@NamedQueries(value={

@NamedQuery(),

@NamedQuery()

})

Use createNamedQuery(“query\_name”) - to call the specified query

Native queries - to use native sql queries in jpa

createNativeQuery(“select \* from course where id=?”,Course.class)

\* You can have positional or named parameters

Establishing Relationship with jpa and hibernate

a)one to one - @OnetoOne

**1)One to one relationship is always Eager Fetch**

Eager fetch - retrieves the details of both entities

To change the fetch use - @OnetoOne(fetch=FetchType.LAZY) -

It retrieves data only from the specified one.

Ex: each student have a unique passport

**Make sure to declare a owning side of relationship** :

Add this on non-owning side of entity @OnetoOne(mappedby=”owningentityname”)

@Transactional - to make a transaction a complete success or nothing succed.

If any operation fails, all operations done must be rolled back

As soon as you create a transaction, you would create a persistence context( a place where it holds all the entities that are being operated on)

A way to interact with persistence context is by using entity manager.

**2)OnetoMany -**

Default fetching - lazy

ex: course can have many reviews or no reviews

**3)ManytoOne -**

Default fetching is eager

**4)ManytoMany**

Default fetching is lazy

Ex: course can have multiple students enrolled and a student can enroll for multiple courses

@JoinTable(name=”jointablename”,

joinColumns=@JoinColumn(name=”owning\_entityname”),

inverseJoinColumns=@JoinColum(name=”nonowning\_entityname”))

**Queries using JAVA API - Criteria Queries**

1.Use Criteria Builder to create a criteria Query returning the expected result object

2.Define roots for tables which are invloved in the query

3.Define Predicates etc using Criteria Builder

4.Add predicates etc to the Criteria Query

5.Build the TypedOuery using the entity manager and criteria query

**TRANSACTION MANAGEMENT**

It is an important concept while making changes or updates to database.

Transaction management make sure that when a failure occurs, to roll back to its previous state.

Spring uses annotation - @Transctional

**ACID PROPERTIES**

1)Atomicity - Transaction must be completely successful or all the changes that are done by transaction must be reverted back when it fails.

2)Consistency - Transaction must leave the system in consistent state

3)Isolation - How changes within a transaction must be visible to other transactions

4)Durability - Once a transaction is successful, even when the system fails or crashes, the end state of transaction must be persistence.

**IMPORTANT CONCEPTS RELATED TO ISOLATION LEVEL**

1)Dirty Read - Two transactions are running in parallel, another transaction reading the modified value before the given transaction is committed.

2)Non-repeatable Read - When i am reading the same thing twice, i'm getting two different values.

3)Phanthom Read - at different times, i'm getting different number of rows for the same transaction.

Four Isolation Levels

1)Read uncommitted - no restriction on data (no locks at all)

problems faced - dirty read, non-repeatable read, phanthom read

2)Readcommitted - read only the transactions of data which is committed by another transaction.(lock is set until the transaction is committed)

problems solved - dirty read

3)Repeatable read - locks both modified as well as read values during the transaction.

problems solved - dirty read, non-repeatable read

4)Serializable - table lock on constraint : cant update, delete or insert until that transaction has been completed.

problems solved - dirty read, non-repeatable read, phanthom read

**THREE THINGS TO DECIDE WHILE DEALING WITH TRANSACTION MANAGEMENT**

1)Spring or jpa transaction management

2)decide the isolation level

3)default isolation level by configuring the settings in spring.jpa.hibernate.connection.isolation = 2;

values

1 - Read uncommitted

2 - Readcommitted

4 - Repeatable read

8 - Serializable

**Spring Data JPA and Spring Data REST**

earlier problems - lot of duplication amon repositories, proliferation of data source

so spring provides simple abstraction being able to access any kind of data.

Spring data JPA is the JPA specific implementation of spring data

Spring data Jpa repository returns an Optional tag

Optional tag provides to check if entity object exist.

CRUD Repository is one of the fundamental Spring Data Repositories

**Basic Crud operations**

methods supported

1)findById()

2)existsById()

3)findAll()

4)findByAllId(Iterable)

5)count()

6)deleteById(ID)

7)deleteAll()

8)save(s) -- > either to create or update

Paging and sorting repository provides facilities methods to sort based on specific field, add

pagination to the results that are delivered by the crud repository

**JPA Repository has specific methods**

1)flush()

2)saveAndFlush()

3)deleteInBatch()