IN28MINUTES

JPA and Hibernate with Spring Boot - Course Guide

Master JPA using Hibernate as the implementation. Learn the basics of JPA - entities, relationships, entity manager, annotations, JPQL and Criteria API. Take a step into the advanced world of JPA - caching, performance tuning(n + 1 queries), mapping inheritance hierarchies. Get a peek into the magic of Spring Data JPA & Spring Data Rest.

Introduction

The Java Persistence API provides Java developers with an api for mapping java objects to relational data. In this course, you will learn about the JPA API, JPQL (Java Persistence query language), Java Persistence Criteria API and how you can perform ORM (Object Relational Mapping) with JPA.

Hibernate is the most popular implementation of JPA. It was the most popular ORM framework option before JPA emerged and it provides additional features on top of JPA. We will use Hibernate as the implementation in this course.

What You will learn

- You will learn the basics of JPA and Hibernate - Entities, Relationships, Inheritance Mappings and Annotations
- You will understand approaches to querying data using JPA and Hibernate - JPQL, Criteria API and Native Queries
- You will understand JPA and Hibernate Relationships in depth - One to One, Many to One and Many to Many
- You will use a variety of Spring Boot Starters - Spring Boot Starter Web, Starter Data Jpa, Starter Test
- You will learn the basic of performance tuning your JPA application with Hibernate - Solve N+1 Queries Issue.
- You will learn the basics of caching - First Level Cache and Second Level Cache with EhCache
- You will understand the basics of Spring Data JPA and Spring Data REST

Connecting to My SQL and Other Databases

Spring Boot makes it easy to switch databases! Yeah really simple.

Steps

- Install MySQL and Setup Schema
- Remove H2 dependency from pom.xml
- Add MySQL (or your database) dependency to pom.xml ```xml

mysql mysql-connector-java

```
- Configure application.properties

```properties
spring.jpa.hibernate.ddl-auto=none
spring.datasource.url=jdbc:mysql://localhost:3306/person_example
spring.datasource.username=personuser
spring.datasource.password=YOUR_PASSWORD
```

- Restart the app and You are ready!

> *Spring Boot can setup the database for you using Hibernate*

Things to note:

- Spring Boot chooses a default value for you based on whether it thinks your database is embedded (default create-drop) or not (default none).
- `spring.jpa.hibernate.ddl-auto` is the setting to perform SchemaManagementTool actions automatically
  - none : No action will be performed.
  - create-only : Database creation will be generated.
  - drop : Database dropping will be generated.
  - create : Database dropping will be generated followed by database creation.
  - validate : Validate the database schema
  - update : Update the database schema
- Reference : https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#configurations-hbmddl
```

application.properties

```
#none, validate, update, create, create-drop
spring.jpa.hibernate.ddl-auto=create
```

Installing and Setting Up MySQL

- Install MySQL https://dev.mysql.com/doc/en/installing.html
    - More details - http://www.mysqltutorial.org/install-mysql/
    - Trouble Shooting - https://dev.mysql.com/doc/refman/en/problems.html
- Startup the Server (as a service)
- Go to command prompt (or terminal)
    - Execute following commands to create a database and a user

```
mysql --user=user_name --password db_name
create database person_example;
create user 'personuser'@'localhost' identified by 'YOUR_PASSWORD';
grant all on person_example.* to 'personuser'@'localhost';
```

- Execute following sql queries to create the table and insert the data

JDBC TO JPA

Table

```
create table person
(
  id integer not null,
  birth_date timestamp,
  location varchar(255),
  name varchar(255),
  primary key (id)
);
```

Data

```
INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH_DATE ) VALUES(10001,  'Ranga', 'Hyderabad',sysdate());
INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH_DATE ) VALUES(10002,  'James', 'New York',sysdate());
INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH_DATE ) VALUES(10003,  'Pieter', 'Amsterdam',sysdate());
```

JPA in Depth

Tables

```
    create table course (
       id bigint not null,
        created_date timestamp,
        is_deleted boolean not null,
        last_updated_date timestamp,
        name varchar(255) not null,
        primary key (id)
    );
    create table full_time_employee (
       id bigint not null,
        name varchar(255) not null,
        salary decimal(19,2),
        primary key (id)
    );
    create table part_time_employee (
       id bigint not null,
        name varchar(255) not null,
        hourly_wage decimal(19,2),
        primary key (id)
    );
    create table passport (
       id bigint not null,
        number varchar(255) not null,
        primary key (id)
    );
    create table review (
       id bigint not null,
        description varchar(255),
        rating varchar(255),
        course_id bigint,
        primary key (id)
    );
    create table student (
       id bigint not null,
        city varchar(255),
        line1 varchar(255),
        line2 varchar(255),
        name varchar(255) not null,
```

```
    passport_id bigint,
    primary key (id)
);
create table student_course (
   student_id bigint not null,
    course_id bigint not null
)
alter table review
   add constraint FKprox8elgnr8u5wrq1983degk
   foreign key (course_id)
   references course;
alter table student
   add constraint FK6i2dofwfuu97njtfprqv68pib
   foreign key (passport_id)
   references passport;
alter table student_course
   add constraint FKejrkh4gv8iqgmspsanaji90ws
   foreign key (course_id)
   references course;
alter table student_course
   add constraint FKq7yw2wg9wlt2cnj480hcdn6dq
   foreign key (student_id)
   references student;
```

Data

```
insert into course(id, name, created_date, last_updated_date,is_deleted)
values(10001,'JPA in 50 Steps', sysdate(), sysdate(),false);
insert into course(id, name, created_date, last_updated_date,is_deleted)
values(10002,'Spring in 50 Steps', sysdate(), sysdate(),false);
insert into course(id, name, created_date, last_updated_date,is_deleted)
values(10003,'Spring Boot in 100 Steps', sysdate(), sysdate(),false);


insert into passport(id,number)
values(40001,'E123456');
insert into passport(id,number)
values(40002,'N123457');
insert into passport(id,number)
values(40003,'L123890');

insert into student(id,name,passport_id)
values(20001,'Ranga',40001);
insert into student(id,name,passport_id)
values(20002,'Adam',40002);
insert into student(id,name,passport_id)
values(20003,'Jane',40003);

insert into review(id,rating,description,course_id)
values(50001,'FIVE', 'Great Course',10001);
insert into review(id,rating,description,course_id)
values(50002,'FOUR', 'Wonderful Course',10001);
insert into review(id,rating,description,course_id)
values(50003,'FIVE', 'Awesome Course',10003);

insert into student_course(student_id,course_id)
values(20001,10001);
insert into student_course(student_id,course_id)
values(20002,10001);
insert into student_course(student_id,course_id)
values(20003,10001);
insert into student_course(student_id,course_id)
values(20001,10003);
```

Installing Tools

- Installation Video : https://www.youtube.com/playlist?list=PLBBog2r6uMCSmMVTW_QmDLyASBvovyAO3
- GIT Repository For Installation : https://github.com/in28minutes/getting-started-in-5-steps
- PDF : https://github.com/in28minutes/SpringIn28Minutes/blob/master/InstallationGuide-JavaEclipseAndMaven_v2.pdf

Running Examples

- Download the zip or clone the Git repository.
- Unzip the zip file (if you downloaded one)
- Open Command Prompt and Change directory (cd) to folder containing pom.xml
- Open Eclipse
    - File -> Import -> Existing Maven Project -> Navigate to the folder where you unzipped the zip
    - Select the right project
- Choose the Spring Boot Application file (search for @SpringBootApplication)
- Right Click on the file and Run as Java Application
- You are all Set
- For help : use our installation guide - https://www.youtube.com/playlist?list=PLBBog2r6uMCSmMVTW_QmDLyASBvovyAO3

Step By Step Details

JPA Introduction

- 0005 - Quick introduction to JPA

Spring Boot in 10 Steps

- Link - Spring Boot in 5 Steps
- Github Folder -

Journey from JDBC To JPA

- Step 01 - Setting up a project with JDBC, JPA, H2 and Web Dependencies
- Step 02 - Launching up H2 Console
- Step 03 - Creating a Database Table in H2
- Step 04 - Populate data into Person Table
- Step 05 - Implement findAll persons Spring JDBC Query Method
- Step 06 - Execute the findAll method using CommandLineRunner
- Step 07 - A Quick Review - JDBC vs Spring JDBC
- Step 08 - Whats in the background? Understanding Spring Boot Autoconfiguration
- Step 09 - Implementing findById Spring JDBC Query Method
- Step 10 - Implementing deleteById Spring JDBC Update Method
- Step 11 - Implementing insert and update Spring JDBC Update Methods
- Step 12 - Creating a custom Spring JDBC RowMapper
- Step 13 - Quick introduction to JPA
- Step 14 - Defining Person Entity
- Step 15 - Implementing findById JPA Repository Method
- Step 16 - Implementing insert and update JPA Repository Methods
- Step 17 - Implementing deleteById JPA Repository Method
- Step 18 - Implementing findAll using JPQL Named Query

JUnit in 5 Steps

- Link - JUnit in 5 Steps
- Github Folder -

JPA/Hibernate in Depth

- Step 01 - Create a JPA Project with H2 and Spring Boot
- Step 02 - Create JPA Entity Course
- Step 03 - Create findById using JPA Entity Manager
- Step 04 - Configuring application.properties to enable H2 console and logging
- Step 05 - Writing Unit Test for findById method
- Step 06 - Writing a deleteByID method to delete an Entity
- Step 07 - Writing Unit Test for deleteById method
- Step 08 - Writing a save method to update and insert an Entity
- Step 09 - Writing Unit Test for save method
- Step 10 - Quick Review and Debugging Tips
- Step 11 - Playing with Entity Manager
- Step 12 - Entity Manager Methods - clear and detach
- Step 13 - Entity Manager Methods - refresh
- Step 14 - A Quick Review of Entity Manager
- Step 15 - JPQL - Basics
- Step 16 - JPA and Hibernate Annotations - @Table
- Step 17 - JPA and Hibernate Annotations - @Column
- Step 18 - JPA and Hibernate Annotations - @UpdateTimestamp and @CreationTimestamp
- Step 19 - JPA and Hibernate Annotations - @NamedQuery and @NamedQueries
- Step 20 - Native Queries - Basics
- Step 21 - Entities and Relationships - An overview
- Step 22 - Defining Entities - Student, Passport and Review
- Step 23 - Introduction to One to One Relationship
- Step 24 - OneToOne Mapping - Insert Student with Passport
- Step 25 - OneToOne Mapping - Retrieving Student with Passport and Eager Fetch
- Step 26 - OneToOne Mapping - Lazy Fetch
- Step 27 - Transaction, Entity Manager and Persistence Context
- Step 28 - OneToOne Mapping - Bidirectional Relationship - Part 1
- Step 29 - OneToOne Mapping - Bidirectional Relationship - Part 2
- Step 30 - ManyToOne Mapping - Designing the database
- Step 30 - 02 - ManyToOne Mapping - Implementing the Mapping *****
- Step 31 - ManyToOne Mapping - Retrieving and inserting Reviews for Course
- Step 32 - ManyToOne Mapping - Generalizing Insert Reviews
- Step 33 - ManyToOne Mapping - Wrapping up
- Step 34 - ManyToMany Mapping - Table Design
- Step 35 - ManyToMany Mapping - Adding Annotations on Entities
- Step 36 - ManyToMany Mapping - Fixing two join tables problem
- Step 37 - ManyToMany Mapping - Customizing the Join Table
- Step 38 - ManyToMany Mapping - Insert Data and Write Join Query
- Step 39 - ManyToMany Mapping - Retrieve Data using JPA Relationships
- Step 40 - ManyToMany Mapping - Insert Student and Course
- Step 41 - Relationships between JPA Entities - A summary
- Step 42 - Introduction to Inheritance Hierarchies and Mappings

- Step 43 - JPA Inheritance Hierarchies and Mappings - Setting up entities
- Step 44 - JPA Inheritance Hierarchies and Mappings - Setting up a Repository
- Step 45 - JPA Inheritance Hierarchies and Mappings - Single Table
- Step 46 - JPA Inheritance Hierarchies and Mappings - Table Per Class
- Step 47 - JPA Inheritance Hierarchies and Mappings - Joined
- Step 48 - JPA Inheritance Hierarchies and Mappings - Mapped Super Class
- Step 49 - JPA Inheritance Hierarchies and Mappings - How to Choose?
- Step 50 - JPQL - Courses without Students
- Step 51 - JPQL - Courses with atleast 2 Students and order by
- Step 52 - JPQL - Courses like 100 Steps
- Step 53 - JPQL - Using Joins
- Step 54 - Criteria Query - Retrieving all courses
- Step 55 - Criteria Query - Courses like 100 Steps
- Step 56 - Criteria Query - Courses without Students
- Step 57 - Criteria Query - Using Joins
- Step 58 - Introduction to Transaction Management
- Step 59 - Transaction Management - ACID Properties
- Step 60 - Understanding Dirty, Phanthom and Non Repeatable Reads
- Step 61 - Understand 4 Isolation Levels
- Step 62 - Choosing between Isolation Levels
- Step 63 - Implementing Transaction Management - 3 Things to Decide
- Step 64 - Introduction to Spring Data JPA
- Step 65 - Testing the Spring Data JPA Repository with findById.
- Step 66 - Spring Data JPA Repository - CRUD Methods
- Step 67 - Sorting using Spring Data JPA Repository
- Step 68 - Pagination using Spring Data JPA Repository
- Step 69 - Custom Queries using Spring Data JPA Repository
- Step 70 - Spring Data REST
- Step 71 - Introduction to Caching
- Step 72 - Hibernate and JPA Caching - First Level Cache
- Step 73 - Hibernate and JPA Caching - Basics of Second Level Cache with EhCache
- Step 74 - Hibernate and JPA Caching - Second Level Cache Part 2
- Step 75 - Hibernate Tips - Hibernate Soft Deletes - @SQLDelete and @Where
- Step 76 - Hibernate Soft Deletes - Part 2
- Step 77 - JPA Entity Life Cycle Methods
- Step 78 - Using Embedded and Embeddable with JPA
- Step 79 - Using Enums with JPA
- Step 80 - JPA Tip - Be cautious with toString method implementations
- Step 81 - JPA Tip - When do you use JPA?
- Step 82 - Performance Tuning - Measure before Tuning
- Step 83 - Performance Tuning - Indexes
- Step 84 - Performance Tuning - Use Appropriate Caching
- Step 85 - Performance Tuning - Eager vs Lazy Fetch
- Step 86 - Performance Tuning - Avoid N+1 Problems
- FAQ 1 - When does Hibernate send updates to the database?
- FAQ 2 - When do we need @Transactional in an Unit Test?
- FAQ 3 - Do read only methods need a transaction?
- FAQ 4 - Why do we use @DirtiesContext in an Unit Test?
- FAQ 5 - How to connect to a different database with Spring Boot?
- FAQ 6 - Approach to design great applications with JPA?
- FAQ 7 - Good Practices for developing JPA Applications

Spring Framework in 10 Steps

- Link - Spring Framework in 10 Steps
- Github Folder -

Step By Step Notes

Journey from JDBC To JPA

> *TODO*

> *TODO*

JPA/Hibernate in Depth

Step 01 - Create a JPA Project with H2 and Spring Boot

Creating a JPA Project with Spring Initializr is a cake walk.

> *Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.*

Notes

- Launch Spring Initializr and choose the following
  - Choose `com.in28minutes.jpa.hibernate` as Group
  - Choose `demo` as Artifact

        - Choose the following Dependencies
            - Web
            - JPA
            - H2
- Click Generate Project.
- Import the project into Eclipse.
- If you want to understand all the files that are part of this project, you can go here.

Step 02 - Create JPA Entity Course

Define an Entity Course with a primary key id.

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java

```
package com.in28minutes.jpa.hibernate.demo.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Course {

  @Id
  @GeneratedValue
  private Long id;

  private String name;

  protected Course() {
  }

  public Course(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public Long getId() {
    return id;
  }
}
```

Important things to note:

- @Entity: Specifies that the class is an entity. This annotation is applied to the entity class.
- @Id: Specifies the primary key of an entity.
- @GeneratedValue: Provides for the specification of generation strategies for the values of primary keys.
- protected Course(): Default constructor to make JPA Happy

Step 03 - Create findById using JPA Entity Manager

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepository.java

```
package com.in28minutes.jpa.hibernate.demo.repository;

import javax.persistence.EntityManager;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.in28minutes.jpa.hibernate.demo.entity.Course;

@Repository
public class CourseRepository {

  @Autowired
  EntityManager em;

  public Course findById(Long id){
    return em.find(Course.class, id);
  }

  //public Course save(Course course) -> insert or update

  //public void deleteById(Long id)

}
```

Enhance DemoApplication to implement CommandLineRunner and invoke the `repository.findById(10001L)` method.

/src/main/java/com/in28minutes/jpa/hibernate/demo/DemoApplication.java

```
@SpringBootApplication
public class DemoApplication implements CommandLineRunner{

  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  private CourseRepository repository;

  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
  }

  @Override
  public void run(String... arg0) throws Exception {
    Course course = repository.findById(10001L);

    logger.info("Course 10001 -> {}", course);

  }
}
```

Step 04 - Configuring application.properties to enable H2 console and logging

Three important things are configured in application.properties

- Enable H2 Console spring.h2.console.enabled=true
  - http://localhost:8080/h2-console
  - Use db url jdbc:h2:mem:testdb
- Turn the hibernate statistics on
- Enable logging of all queries

/src/main/resources/application.properties

```
# Enabling H2 Console
spring.h2.console.enabled=true
#Turn Statistics on
spring.jpa.properties.hibernate.generate_statistics=true
logging.level.org.hibernate.stat=debug
# Show all queries
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.type=trace
```

Insert a row into the Course table

/src/main/resources/data.sql

```
insert into course(id, name) values(10001,'JPA in 50 Steps');
```

Step 05 - Writing Unit Test for findById method

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepositoryTest.java

```
package com.in28minutes.jpa.hibernate.demo.repository;

import static org.junit.Assert.*;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.hibernate.demo.DemoApplication;
import com.in28minutes.jpa.hibernate.demo.entity.Course;

@RunWith(SpringRunner.class)
@SpringBootTest(classes=DemoApplication.class)
public class CourseRepositoryTest {

  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  CourseRepository repository;

  @Test
  public void findById_basic() {
    Course course = repository.findById(10001L);
    assertEquals("JPA in 50 Steps", course.getName());
  }

}
```

Step 06 - Writing a deleteByID method to delete an Entity

class DemoApplication

Modified Lines

```
      repository.deleteById(10001L);
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepository.java Modified Lines

```
  import org.springframework.transaction.annotation.Transactional;
  @Transactional


  public void deleteById(Long id){
    Course course = findById(id);
    em.remove(course);
  }
```

/src/main/resources/data.sql Modified Lines

```
  insert into course(id, name) values(10002,'Spring in 50 Steps');
  insert into course(id, name) values(10003,'Spring Boot in 100 Steps');
```

Step 07 - Writing Unit Test for deleteById method

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepositoryTest.java

```
  //Imports
  import org.springframework.test.annotation.DirtiesContext;
  import org.springframework.transaction.annotation.Transactional;

  //New Method
  @DirtiesContext
  public void deleteById_basic() {
    repository.deleteById(10002L);
    assertNull(repository.findById(10002L));
  }
```

Step 08 - Writing a save method to update and insert an Entity

/src/main/java/com/in28minutes/jpa/hibernate/demo/DemoApplication.java

Modified Lines

```
      repository.save(new Course("Microservices in 100 Steps"));
    }
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepository.java Modified Lines

```
    public Course save(Course course) {

      if (course.getId() == null) {
        em.persist(course);
      } else {
        em.merge(course);
      }

      return course;
    }
```

Step 09 - Writing Unit Test for save method

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepositoryTest.java Modified Lines

```
  @SpringBootTest(classes = DemoApplication.class)


  public void save_basic() {
    // get a course
    // update details
    course.setName("JPA in 50 Steps - Updated");
    repository.save(course);
    // check the value
    Course course1 = repository.findById(10001L);
    assertEquals("JPA in 50 Steps - Updated", course1.getName());
  }
```

Step 10 - Quick Review and Debugging Tips

Step 11 - Playing with Entity Manager

/src/main/java/com/in28minutes/jpa/hibernate/demo/DemoApplication.java Modified Lines

```
      repository.playWithEntityManager();
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepository.java Modified Lines

```
  public void playWithEntityManager() {
    Course course = new Course("Web Services in 100 Steps");
    em.persist(course);
    course.setName("Web Services in 100 Steps - Updated");
  }
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepositoryTest.java

```
    @Test
    @DirtiesContext
    public void playWithEntityManager() {
      repository.playWithEntityManager();
    }
```

Step 12 - Entity Manager Methods - clear and detach

Step 13 - Entity Manager Methods - refresh

Step 14 - A Quick Review of Entity Manager

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepository.java Modified Lines

```
    public void playWithEntityManager() {

      Course course1 = new Course("Web Services in 100 Steps");
      em.persist(course1);

      Course course2 = new Course("Angular Js in 100 Steps");
      em.persist(course2);

      em.flush();

      course1.setName("Web Services in 100 Steps - Updated");
      course2.setName("Angular Js in 100 Steps - Updated");

      em.refresh(course1);

      em.flush();
    }
```

Step 15 - JPQL - Basics

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/JPQLTest.java

```
  @RunWith(SpringRunner.class)
  @SpringBootTest(classes = DemoApplication.class)
  public class JPQLTest {

    private Logger logger = LoggerFactory.getLogger(this.getClass());

    @Autowired
    EntityManager em;

    @Test
    public void jpql_basic() {
      Query query = em.createQuery("Select  c  From Course c");
      List resultList = query.getResultList();
      logger.info("Select  c  From Course c -> {}",resultList);
    }

    @Test
    public void jpql_typed() {
      TypedQuery<Course> query =
            em.createQuery("Select  c  From Course c", Course.class);

      List<Course> resultList = query.getResultList();

      logger.info("Select  c  From Course c -> {}",resultList);
    }

    @Test
    public void jpql_where() {
      TypedQuery<Course> query =
            em.createQuery("Select  c  From Course c where name like '%100 Steps'", Course.class);

      List<Course> resultList = query.getResultList();
```

```
    logger.info("Select  c  From Course c where name like '%100 Steps'-> {}",resultList);
    //[Course[Web Services in 100 Steps], Course[Spring Boot in 100 Steps]]
  }

}
```

Step 16 - JPA and Hibernate Annotations - @Table

Step 17 - JPA and Hibernate Annotations - @Column

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java

```
@Entity
public class Course {

  @Id
  @GeneratedValue
  private Long id;

  @Column(nullable = false)
  private String name;

}
```

This method would throw an exception because is name is not nullable.

```
public void playWithEntityManager() {
  Course course1 = new Course("Web Services in 100 Steps");
  em.persist(course1);

  course1.setName(null);
}
```

Step 18 - JPA and Hibernate Annotations - @UpdateTimestamp and @CreationTimestamp

Step 19 - JPA and Hibernate Annotations - @NamedQuery and @NamedQueries

Step 20 - Native Queries - Basics

Step 21 - Entities and Relationships - An overview

Step 22 - Defining Entities - Student, Passport and Review

Step 23 - Introduction to One to One Relationship

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java Modified Lines

```
import java.time.LocalDateTime;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

@NamedQueries(value = {
    @NamedQuery(name = "query_get_all_courses",
        query = "Select  c  From Course c"),
    @NamedQuery(name = "query_get_100_Step_courses",
    query = "Select  c  From Course c where name like '%100 Steps'") })

@UpdateTimestamp
private LocalDateTime lastUpdatedDate;

@CreationTimestamp
private LocalDateTime createdDate;
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Passport.java New

```
package com.in28minutes.jpa.hibernate.demo.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Passport {

  @Id
  @GeneratedValue
  private Long id;

  @Column(nullable = false)
  private String number;

  protected Passport() {
```

```
  }

  public Passport(String number) {
    this.number = number;
  }

  public String getNumber() {
    return number;
  }

  public void setNumber(String number) {
    this.number = number;
  }

  public Long getId() {
    return id;
  }

  @Override
  public String toString() {
    return String.format("Passport[%s]", number);
  }
}
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Review.java New

```
package com.in28minutes.jpa.hibernate.demo.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Review {

  @Id
  @GeneratedValue
  private Long id;

  private String rating;

  private String description;

  protected Review() {
  }

  public Review(String rating, String description) {
    this.rating = rating;
    this.description = description;
  }

  public String getDescription() {
    return description;
  }

  public void setDescription(String description) {
    this.description = description;
  }

  public String getRating() {
    return rating;
  }

  public void setRating(String rating) {
    this.rating = rating;
  }

  public Long getId() {
    return id;
  }

  @Override
  public String toString() {
    return String.format("Review[%s %s]", rating, description);
  }

}
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Student.java New

```
package com.in28minutes.jpa.hibernate.demo.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
```

```
@Entity
public class Student {

  @Id
  @GeneratedValue
  private Long id;

  @Column(nullable = false)
  private String name;

  @OneToOne
  private Passport passport;

  protected Student() {
  }

  public Student(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public Long getId() {
    return id;
  }

  @Override
  public String toString() {
    return String.format("Student[%s]", name);
  }
}
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepository.java

```
  public void playWithEntityManager() {
    Course course1 = new Course("Web Services in 100 Steps");
    em.persist(course1);

    Course course2 = findById(10001L);

    course2.setName("JPA in 50 Steps - Updated");

  }
```

/src/main/resources/data.sql Modified Lines

```
insert into course(id, name, created_date, last_updated_date)
values(10001,'JPA in 50 Steps', sysdate(), sysdate());
insert into course(id, name, created_date, last_updated_date)
values(10002,'Spring in 50 Steps', sysdate(), sysdate());
insert into course(id, name, created_date, last_updated_date)
values(10003,'Spring Boot in 100 Steps', sysdate(), sysdate());
insert into passport(id,number)
values(40001,'E123456');
insert into passport(id,number)
values(40002,'N123457');
insert into passport(id,number)
values(40003,'L123890');
insert into student(id,name,passport_id)
values(20001,'Ranga',40001);
insert into student(id,name,passport_id)
values(20002,'Adam',40002);
insert into student(id,name,passport_id)
values(20003,'Jane',40003);
insert into review(id,rating,description)
values(50001,'5', 'Great Course');
insert into review(id,rating,description)
values(50002,'4', 'Wonderful Course');
insert into review(id,rating,description)
values(50003,'5', 'Awesome Course');
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/JPQLTest.java Modified

Full File

```
  @Test
  public void jpql_typed() {
    TypedQuery<Course> query = em.createNamedQuery("query_get_all_courses", Course.class);
```

```
    List<Course> resultList = query.getResultList();

    logger.info("Select  c  From Course c -> {}", resultList);
  }

  @Test
  public void jpql_where() {
    TypedQuery<Course> query = em.createNamedQuery("query_get_100_Step_courses", Course.class);

    List<Course> resultList = query.getResultList();

    logger.info("Select  c  From Course c where name like '%100 Steps'-> {}", resultList);
    // [Course[Web Services in 100 Steps], Course[Spring Boot in 100 Steps]]
  }
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/NativeQueriesTest.java New

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class)
public class NativeQueriesTest {

  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  EntityManager em;

  @Test
  public void native_queries_basic() {
    Query query = em.createNativeQuery("SELECT * FROM COURSE", Course.class);
    List resultList = query.getResultList();
    logger.info("SELECT * FROM COURSE  -> {}", resultList);
    //SELECT * FROM COURSE  -> [Course[Web Services in 100 Steps], Course[JPA in 50 Steps - Updated], Course[Sp
  }

  @Test
  public void native_queries_with_parameter() {
    Query query = em.createNativeQuery("SELECT * FROM COURSE where id = ?", Course.class);
    query.setParameter(1, 10001L);
    List resultList = query.getResultList();
    logger.info("SELECT * FROM COURSE  where id = ? -> {}", resultList);
    //[Course[JPA in 50 Steps - Updated]]
  }

  @Test
  public void native_queries_with_named_parameter() {
    Query query = em.createNativeQuery("SELECT * FROM COURSE where id = :id", Course.class);
    query.setParameter("id", 10001L);
    List resultList = query.getResultList();
    logger.info("SELECT * FROM COURSE  where id = :id -> {}", resultList);
    //[Course[JPA in 50 Steps - Updated]]
  }

  @Test
  @Transactional
  public void native_queries_to_update() {
    Query query = em.createNativeQuery("Update COURSE set last_updated_date=sysdate()");
    int noOfRowsUpdated = query.executeUpdate();
    logger.info("noOfRowsUpdated  -> {}", noOfRowsUpdated);
    //SELECT * FROM COURSE  -> [Course[Web Services in 100 Steps], Course[JPA in 50 Steps - Updated], Course[Sp
  }


}
```

Step 24 - OneToOne Mapping - Insert Student with Passport

Step 25 - OneToOne Mapping - Retrieving Student with Passport and Eager Fetch

Step 26 - OneToOne Mapping - Lazy Fetch

Step 27 - Transaction, Entity Manager and Persistence Context

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Student.java Modified

Full File

```
@Entity
public class Student {

  @Id
  @GeneratedValue
  private Long id;

  @Column(nullable = false)
  private String name;
```

```
  @OneToOne(fetch=FetchType.LAZY)
  private Passport passport;

}
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/StudentRepository.java New

```
@Repository
@Transactional
public class StudentRepository {

  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  EntityManager em;

  public Student findById(Long id) {
    return em.find(Student.class, id);
  }

  public Student save(Student student) {

    if (student.getId() == null) {
      em.persist(student);
    } else {
      em.merge(student);
    }

    return student;
  }

  public void deleteById(Long id) {
    Student student = findById(id);
    em.remove(student);
  }

  public void saveStudentWithPassport() {
    Passport passport = new Passport("Z123456");
    em.persist(passport);

    Student student = new Student("Mike");

    student.setPassport(passport);
    em.persist(student);
  }

  public void someOperationToUnderstandPersistenceContext() {
    //Database Operation 1 - Retrieve student
    Student student = em.find(Student.class, 20001L);
    //Persistence Context (student)


    //Database Operation 2 - Retrieve passport
    Passport passport = student.getPassport();
    //Persistence Context (student, passport)

    //Database Operation 3 - update passport
    passport.setNumber("E123457");
    //Persistence Context (student, passport++)

    //Database Operation 4 - update student
    student.setName("Ranga - updated");
    //Persistence Context (student++ , passport++)
  }

}
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/StudentRepositoryTest.java New

```
package com.in28minutes.jpa.hibernate.demo.repository;

import javax.persistence.EntityManager;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.transaction.annotation.Transactional;

import com.in28minutes.jpa.hibernate.demo.DemoApplication;
import com.in28minutes.jpa.hibernate.demo.entity.Student;

@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class)
public class StudentRepositoryTest {
```

```
  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  StudentRepository repository;

  @Autowired
  EntityManager em;

  //Session & Session Factory


  //EntityManager & Persistence Context
  //Transaction

  @Test
  public void someTest() {
    repository.someOperationToUnderstandPersistenceContext();
  }


  @Test
  @Transactional
  public void retrieveStudentAndPassportDetails() {
    Student student = em.find(Student.class, 20001L);
    logger.info("student -> {}", student);
    logger.info("passport -> {}",student.getPassport());
  }
}
```

TODO CONFUSION Step 27 - Transaction, Entity Manager and Persistence Context

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/StudentRepositoryTest.java

```
  @Test
  @Transactional
  public void someTest() {
    //Database Operation 1 - Retrieve student
    //Database Operation 2 - Retrieve passport
    //Database Operation 3 - update passport
  }
```

Step 28 - OneToOne Mapping - Bidirectional Relationship - Part 1

Step 29 - OneToOne Mapping - Bidirectional Relationship - Part 2

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Passport.java Modified Modified Lines

```
import javax.persistence.FetchType;
import javax.persistence.OneToOne;

@OneToOne(fetch=FetchType.LAZY, mappedBy="passport")
private Student student;
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/StudentRepository.java Modified

```
  public void someOperationToUnderstandPersistenceContext() {
    //Database Operation 1 - Retrieve student
    Student student = em.find(Student.class, 20001L);
    //Persistence Context (student)


    //Database Operation 2 - Retrieve passport
    Passport passport = student.getPassport();
    //Persistence Context (student, passport)

    //Database Operation 3 - update passport
    passport.setNumber("E123457");
    //Persistence Context (student, passport++)

    //Database Operation 4 - update student
    student.setName("Ranga - updated");
    //Persistence Context (student++ , passport++)
  }
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/StudentRepositoryTest.java

```
  @Test
  public void someTest() {
    repository.someOperationToUnderstandPersistenceContext();
  }
```

```
@Test
@Transactional
public void retrievePassportAndAssociatedStudent() {
  Passport passport = em.find(Passport.class, 40001L);
  logger.info("passport -> {}", passport);
  logger.info("student -> {}", passport.getStudent());
}
```

Step 30 - ManyToOne Mapping - Designing the database

Step 30 - 02 - ManyToOne Mapping - Implementing the Mapping ***

Step 31 - ManyToOne Mapping - Retrieving and inserting Reviews for Course

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java Modified

```
@OneToMany(mappedBy="course")
private List<Review> reviews = new ArrayList<>();
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepository.java

Full File

```
public void addReviewsForCourse() {
  //get the course 10003
  Course course = findById(10003L);
  logger.info("course.getReviews() -> {}", course.getReviews());

  //add 2 reviews to it
  Review review1 = new Review("5", "Great Hands-on Stuff.");
  Review review2 = new Review("5", "Hatsoff.");

  //setting the relationship
  course.addReview(review1);
  review1.setCourse(course);

  course.addReview(review2);
  review2.setCourse(course);

  //save it to the database
  em.persist(review1);
  em.persist(review2);
}
```

/src/main/resources/data.sql Modified Lines

```
insert into review(id,rating,description,course_id)
values(50001,'5', 'Great Course',10001);
insert into review(id,rating,description,course_id)
values(50002,'4', 'Wonderful Course',10001);
insert into review(id,rating,description,course_id)
values(50003,'5', 'Awesome Course',10003);
```

Step 32 - ManyToOne Mapping - Generalizing Insert Reviews

/src/main/java/com/in28minutes/jpa/hibernate/demo/DemoApplication.java Modified Modified Lines

```
  List<Review> reviews = new ArrayList<>();
  reviews.add(new Review("5", "Great Hands-on Stuff."));
  reviews.add(new Review("5", "Hatsoff."));
  courseRepository.addReviewsForCourse(10003L, reviews );
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepository.java Modified

```
public void addHardcodedReviewsForCourse() {
  //get the course 10003
  Course course = findById(10003L);
  logger.info("course.getReviews() -> {}", course.getReviews());

  //add 2 reviews to it
  Review review1 = new Review("5", "Great Hands-on Stuff.");
  Review review2 = new Review("5", "Hatsoff.");

  //setting the relationship
  course.addReview(review1);
  review1.setCourse(course);

  course.addReview(review2);
  review2.setCourse(course);

  //save it to the database
  em.persist(review1);
  em.persist(review2);
}
```

```
public void addReviewsForCourse(Long courseId, List<Review> reviews) {
  Course course = findById(courseId);
  logger.info("course.getReviews() -> {}", course.getReviews());
  for(Review review:reviews)
  {
    //setting the relationship
    course.addReview(review);
    review.setCourse(course);
    em.persist(review);
  }
}
```

Step 33 - ManyToOne Mapping - Wrapping up

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepositoryTest.java Modified

```
@Test
@Transactional
public void retrieveReviewsForCourse() {
  Course course = repository.findById(10001L);
  logger.info("{}",course.getReviews());
}

@Test
@Transactional
public void retrieveCourseForReview() {
  Review review = em.find(Review.class, 50001L);
  logger.info("{}",review.getCourse());
}
```

Step 34 - ManyToMany Mapping - Table Design

Step 35 - ManyToMany Mapping - Adding Annotations on Entities

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java

```
@ManyToMany
private List<Student> students = new ArrayList<>();
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Student.java

```
@ManyToMany
private List<Course> courses = new ArrayList<>();
```

Step 36 - ManyToMany Mapping - Fixing two join tables problem

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java Modified

```
@ManyToMany(mappedBy="courses")
private List<Student> students = new ArrayList<>();
```

Step 37 - ManyToMany Mapping - Customizing the Join Table

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Student.java Modified

```
@ManyToMany
@JoinTable(name = "STUDENT_COURSE",
joinColumns = @JoinColumn(name = "STUDENT_ID"),
inverseJoinColumns = @JoinColumn(name = "COURSE_ID"))
private List<Course> courses = new ArrayList<>();
```

Step 38 - ManyToMany Mapping - Insert Data and Write Join Query

Step 39 - ManyToMany Mapping - Retrieve Data using JPA Relationships

/src/main/resources/data.sql Modified Lines

```
insert into student_course(student_id,course_id)
values(20001,10001);
insert into student_course(student_id,course_id)
values(20002,10001);
insert into student_course(student_id,course_id)
values(20003,10001);
insert into student_course(student_id,course_id)
values(20001,10003);
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/StudentRepositoryTest.java Modified

```
@Test
@Transactional
public void retrieveStudentAndCourses() {
  Student student = em.find(Student.class, 20001L);

  logger.info("student -> {}", student);
  logger.info("courses -> {}", student.getCourses());
}
```

Step 40 - ManyToMany Mapping - Insert Student and Course

Step 41 - Relationships between JPA Entities - A summary

/src/main/java/com/in28minutes/jpa/hibernate/demo/DemoApplication.java Modified

```
    studentRepository.insertStudentAndCourse(new Student("Jack"),
        new Course("Microservices in 100 Steps"));
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/StudentRepository.java Modified

```
  public void insertHardcodedStudentAndCourse(){
    Student student = new Student("Jack");
    Course course = new Course("Microservices in 100 Steps");
    em.persist(student);
    em.persist(course);

    student.addCourse(course);
    course.addStudent(student);
    em.persist(student);
  }

  public void insertStudentAndCourse(Student student, Course course){
    //Student student = new Student("Jack");
    //Course course = new Course("Microservices in 100 Steps");
    student.addCourse(course);
    course.addStudent(student);

    em.persist(student);
    em.persist(course);
  }
```

Step 42 - Introduction to Inheritance Hierarchies and Mappings

Step 43 - JPA Inheritance Hierarchies and Mappings - Setting up entities

Step 44 - JPA Inheritance Hierarchies and Mappings - Setting up a Repository

/src/main/java/com/in28minutes/jpa/hibernate/demo/DemoApplication.java Modified Lines

```
    employeeRepository.insert(new PartTimeEmployee("Jill", new BigDecimal("50")));
    employeeRepository.insert(new FullTimeEmployee("Jack", new BigDecimal("10000")));
    logger.info("All Employees -> {}", employeeRepository.retrieveAllEmployees());
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Employee.java New

```
package com.in28minutes.jpa.hibernate.demo.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public abstract class Employee {

  @Id
  @GeneratedValue
  private Long id;

  @Column(nullable = false)
  private String name;

  protected Employee() {
  }

  public Employee(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
```

```
  }

  public Long getId() {
    return id;
  }

  @Override
  public String toString() {
    return String.format("Employee[%s]", name);
  }
}
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/FullTimeEmployee.java New

```
package com.in28minutes.jpa.hibernate.demo.entity;

import java.math.BigDecimal;

import javax.persistence.Entity;

@Entity
public class FullTimeEmployee extends Employee {
  protected FullTimeEmployee() {
  }

  public FullTimeEmployee(String name, BigDecimal salary) {
    super(name);
    this.salary = salary;
  }

  private BigDecimal salary;

}
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/PartTimeEmployee.java New

```
package com.in28minutes.jpa.hibernate.demo.entity;

import java.math.BigDecimal;

import javax.persistence.Entity;

@Entity
public class PartTimeEmployee extends Employee {

  protected PartTimeEmployee() {
  }

  public PartTimeEmployee(String name, BigDecimal hourlyWage) {
    super(name);
    this.hourlyWage = hourlyWage;
  }

  private BigDecimal hourlyWage;

}
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/EmployeeRepository.java New

```
package com.in28minutes.jpa.hibernate.demo.repository;

import java.util.List;

import javax.persistence.EntityManager;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.in28minutes.jpa.hibernate.demo.entity.Employee;

@Repository
@Transactional
public class EmployeeRepository {

  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  EntityManager em;
```

```
  public void insert(Employee employee) {
    em.persist(employee);
  }

  public List<Employee> retrieveAllEmployees() {
    return em.createQuery("select e from Employee e", Employee.class).getResultList();
  }
}
```

Step 45 - JPA Inheritance Hierarchies and Mappings - Single Table

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Employee.java Modified

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="EmployeeType")
public abstract class Employee {
```

Step 46 - JPA Inheritance Hierarchies and Mappings - Table Per Class

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Employee.java

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Employee {
```

Step 47 - JPA Inheritance Hierarchies and Mappings - Joined

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Employee.java

```
//@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public abstract class Employee {
```

- Step 48 - JPA Inheritance Hierarchies and Mappings - Mapped Super Class
- Step 49 - JPA Inheritance Hierarchies and Mappings - How to Choose?

/src/main/java/com/in28minutes/jpa/hibernate/demo/DemoApplication.java Modified Modified Lines

```
    logger.info("Full Time Employees -> {}",
        employeeRepository.retrieveAllFullTimeEmployees());

    logger.info("Part Time Employees -> {}",
        employeeRepository.retrieveAllPartTimeEmployees());
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Employee.java Modified

```
@MappedSuperclass
//@Entity
//@Inheritance(strategy=InheritanceType.JOINED)
public abstract class Employee {
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/EmployeeRepository.java Modified

```
  public List<PartTimeEmployee> retrieveAllPartTimeEmployees() {
    return em.createQuery("select e from PartTimeEmployee e", PartTimeEmployee.class).getResultList();
  }

  public List<FullTimeEmployee> retrieveAllFullTimeEmployees() {
    return em.createQuery("select e from FullTimeEmployee e", FullTimeEmployee.class).getResultList();
  }
```

Step 50 - JPQL - Courses without Students

Step 51 - JPQL - Courses with atleast 2 Students and order by

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/JPQLTest.java Modified

```
  @Test
  public void jpql_courses_without_students() {
    TypedQuery<Course> query = em.createQuery("Select c from Course c where c.students is empty", Course.class)
    List<Course> resultList = query.getResultList();
    logger.info("Results -> {}", resultList);
    // [Course[Spring in 50 Steps]]
  }
```

```
  @Test
  public void jpql_courses_with_atleast_2_students() {
    TypedQuery<Course> query = em.createQuery("Select c from Course c where size(c.students) >= 2", Course.clas
    List<Course> resultList = query.getResultList();
    logger.info("Results -> {}", resultList);
    //[Course[JPA in 50 Steps]]
  }

  @Test
  public void jpql_courses_ordered_by_students() {
    TypedQuery<Course> query = em.createQuery("Select c from Course c order by size(c.students) desc", Course.c
    List<Course> resultList = query.getResultList();
    logger.info("Results -> {}", resultList);
  }
```

Step 52 - JPQL - Courses like 100 Steps

Step 53 - JPQL - Using Joins

Step 54 - Criteria Query - Retrieving all courses

Step 55 - Criteria Query - Courses like 100 Steps

Step 56 - Criteria Query - Courses without Students

Step 57 - Criteria Query - Using Joins

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CriteriaQueryTest.java New

```
package com.in28minutes.jpa.hibernate.demo.repository;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Join;
import javax.persistence.criteria.JoinType;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.hibernate.demo.DemoApplication;
import com.in28minutes.jpa.hibernate.demo.entity.Course;

@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class)
public class CriteriaQueryTest {

  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  EntityManager em;

  @Test
  public void all_courses() {
    // "Select c From Course c"

    // 1. Use Criteria Builder to create a Criteria Query returning the
    // expected result object
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Course> cq = cb.createQuery(Course.class);

    // 2. Define roots for tables which are involved in the query
    Root<Course> courseRoot = cq.from(Course.class);

    // 3. Define Predicates etc using Criteria Builder

    // 4. Add Predicates etc to the Criteria Query

    // 5. Build the TypedQuery using the entity manager and criteria query
    TypedQuery<Course> query = em.createQuery(cq.select(courseRoot));

    List<Course> resultList = query.getResultList();

    logger.info("Typed Query -> {}", resultList);
    // [Course[JPA in 50 Steps], Course[Spring in 50 Steps], Course[Spring
    // Boot in 100 Steps]]
  }

  @Test
  public void all_courses_having_100Steps() {
    // "Select c From Course c where name like '%100 Steps' "
```

```
    // 1. Use Criteria Builder to create a Criteria Query returning the
    // expected result object
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Course> cq = cb.createQuery(Course.class);

    // 2. Define roots for tables which are involved in the query
    Root<Course> courseRoot = cq.from(Course.class);

    // 3. Define Predicates etc using Criteria Builder
    Predicate like100Steps = cb.like(courseRoot.get("name"), "%100 Steps");

    // 4. Add Predicates etc to the Criteria Query
    cq.where(like100Steps);

    // 5. Build the TypedQuery using the entity manager and criteria query
    TypedQuery<Course> query = em.createQuery(cq.select(courseRoot));

    List<Course> resultList = query.getResultList();

    logger.info("Typed Query -> {}", resultList);
    // [Course[Spring Boot in 100 Steps]]
  }

  @Test
  public void all_courses_without_students() {
    // "Select c From Course c where c.students is empty"

    // 1. Use Criteria Builder to create a Criteria Query returning the
    // expected result object
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Course> cq = cb.createQuery(Course.class);

    // 2. Define roots for tables which are involved in the query
    Root<Course> courseRoot = cq.from(Course.class);

    // 3. Define Predicates etc using Criteria Builder
    Predicate studentsIsEmpty = cb.isEmpty(courseRoot.get("students"));

    // 4. Add Predicates etc to the Criteria Query
    cq.where(studentsIsEmpty);

    // 5. Build the TypedQuery using the entity manager and criteria query
    TypedQuery<Course> query = em.createQuery(cq.select(courseRoot));

    List<Course> resultList = query.getResultList();

    logger.info("Typed Query -> {}", resultList);
    // [Course[Spring in 50 Steps]]
  }

  @Test
  public void join() {
    // "Select c From Course c join c.students s"

    // 1. Use Criteria Builder to create a Criteria Query returning the
    // expected result object
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Course> cq = cb.createQuery(Course.class);

    // 2. Define roots for tables which are involved in the query
    Root<Course> courseRoot = cq.from(Course.class);

    // 3. Define Predicates etc using Criteria Builder
    Join<Object, Object> join = courseRoot.join("students");

    // 4. Add Predicates etc to the Criteria Query

    // 5. Build the TypedQuery using the entity manager and criteria query
    TypedQuery<Course> query = em.createQuery(cq.select(courseRoot));

    List<Course> resultList = query.getResultList();

    logger.info("Typed Query -> {}", resultList);
    // [Course[JPA in 50 Steps], Course[JPA in 50 Steps], Course[JPA in 50
    // Steps], Course[Spring Boot in 100 Steps]]
  }

  @Test
  public void left_join() {
    // "Select c From Course c left join c.students s"

    // 1. Use Criteria Builder to create a Criteria Query returning the
    // expected result object
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Course> cq = cb.createQuery(Course.class);

    // 2. Define roots for tables which are involved in the query
    Root<Course> courseRoot = cq.from(Course.class);

    // 3. Define Predicates etc using Criteria Builder
    Join<Object, Object> join = courseRoot.join("students", JoinType.LEFT);

    // 4. Add Predicates etc to the Criteria Query

    // 5. Build the TypedQuery using the entity manager and criteria query
    TypedQuery<Course> query = em.createQuery(cq.select(courseRoot));

    List<Course> resultList = query.getResultList();
```

```
    logger.info("Typed Query -> {}", resultList);
    // [Course[JPA in 50 Steps], Course[JPA in 50 Steps], Course[JPA in 50
    // Steps], Course[Spring in 50 Steps], Course[Spring Boot in 100 Steps]]
  }

}
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/JPQLTest.java Modified

```
  //like
  //BETWEEN 100 and 1000
  //IS NULL
  //upper, lower, trim, length

  //JOIN => Select c, s from Course c JOIN c.students s
  //LEFT JOIN => Select c, s from Course c LEFT JOIN c.students s
  //CROSS JOIN => Select c, s from Course c, Student s
  //3 and 4 =>3 * 4 = 12 Rows
  @Test
  public void join(){
    Query query = em.createQuery("Select c, s from Course c JOIN c.students s");
    List<Object[]> resultList = query.getResultList();
    logger.info("Results Size -> {}", resultList.size());
    for(Object[] result:resultList){
      logger.info("Course{} Student{}", result[0], result[1]);
    }
  }

  @Test
  public void left_join(){
    Query query = em.createQuery("Select c, s from Course c LEFT JOIN c.students s");
    List<Object[]> resultList = query.getResultList();
    logger.info("Results Size -> {}", resultList.size());
    for(Object[] result:resultList){
      logger.info("Course{} Student{}", result[0], result[1]);
    }
  }

  @Test
  public void cross_join(){
    Query query = em.createQuery("Select c, s from Course c, Student s");
    List<Object[]> resultList = query.getResultList();
    logger.info("Results Size -> {}", resultList.size());
    for(Object[] result:resultList){
      logger.info("Course{} Student{}", result[0], result[1]);
    }
  }
```

Step 58 - Introduction to Transaction Management

Step 59 - Transaction Management - ACID Properties

Step 60 - Understanding Dirty, Phanthom and Non Repeatable Reads

Step 61 - Understand 4 Isolation Levels

Step 62 - Choosing between Isolation Levels

Step 63 - Implementing Transaction Management - 3 Things to Decide

Step 64 - Introduction to Spring Data JPA

Step 65 - Testing the Spring Data JPA Repository with findById.

Step 66 - Spring Data JPA Repository - CRUD Methods

Step 67 - Sorting using Spring Data JPA Repository

Step 68 - Pagination using Spring Data JPA Repository

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseSpringDataRepository.java New

```
package com.in28minutes.jpa.hibernate.demo.repository;

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

import com.in28minutes.jpa.hibernate.demo.entity.Course;

public interface CourseSpringDataRepository extends JpaRepository<Course, Long> {

}
```

/src/main/resources/application.properties Modified Lines

```
spring.jpa.properties.hibernate.connection.isolation=2
# Performance
spring.jpa.properties.hibernate.jdbc.batch_size=10
```

/src/main/resources/data.sql Modified Lines

```
values(10004,'Dummy1', sysdate(), sysdate());
values(10005,'Dummy2', sysdate(), sysdate());
values(10006,'Dummy3', sysdate(), sysdate());
values(10007,'Dummy4', sysdate(), sysdate());
values(10008,'Dummy5', sysdate(), sysdate());
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepositoryTest.java

```
  @Test
  @Transactional
  @DirtiesContext
  public void performance() {
    //for (int i = 0; i < 20; i++)
      //em.persist(new Course("Something" + i));
    //em.flush();

    //EntityGraph graph = em.getEntityGraph("graph.CourseAndStudents");

    EntityGraph<Course> graph = em.createEntityGraph(Course.class);
      Subgraph<List<Student>> bookSubGraph = graph.addSubgraph("students");

      List<Course> courses = em.createQuery("Select c from Course c", Course.class)
          .setHint("javax.persistence.loadgraph", graph)
          .getResultList();
      for (Course course : courses) {
        System.out.println(course + " " + course.getStudents());
      }
  }

  @Test
  @Transactional
  @DirtiesContext
  public void performance_without_hint() {
      List<Course> courses = em.createQuery("Select c from Course c", Course.class)
          //.setHint("javax.persistence.loadgraph", graph)
          .getResultList();
      for (Course course : courses) {
        System.out.println(course + " " + course.getStudents());
      }
  }
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CourseSpringDataRepositoryTest.java New

```
package com.in28minutes.jpa.hibernate.demo.repository;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import java.util.Optional;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.hibernate.demo.DemoApplication;
import com.in28minutes.jpa.hibernate.demo.entity.Course;

@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class)
public class CourseSpringDataRepositoryTest {

  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  CourseSpringDataRepository repository;

  @Test
  public void findById_CoursePresent() {
    Optional<Course> courseOptional = repository.findById(10001L);
    assertTrue(courseOptional.isPresent());
  }

  @Test
  public void findById_CourseNotPresent() {
    Optional<Course> courseOptional = repository.findById(20001L);
```

```
    assertFalse(courseOptional.isPresent());
  }

  @Test
  public void playingAroundWithSpringDataRepository() {
    //Course course = new Course("Microservices in 100 Steps");
    //repository.save(course);

    //course.setName("Microservices in 100 Steps - Updated");
    //repository.save(course);
    logger.info("Courses -> {} ", repository.findAll());
    logger.info("Count -> {} ", repository.count());
  }

  @Test
  public void sort() {
    Sort sort = new Sort(Sort.Direction.ASC, "name");
    logger.info("Sorted Courses -> {} ", repository.findAll(sort));
    //Courses -> [Course[JPA in 50 Steps], Course[Spring in 50 Steps], Course[Spring Boot in 100 Steps]]
  }

  @Test
  public void pagination() {
    PageRequest pageRequest = PageRequest.of(0, 3);

    Page<Course> firstPage = repository.findAll(pageRequest);
    logger.info("First Page -> {} ", firstPage);
  }

}
```

Step 69 - Custom Queries using Spring Data JPA Repository

Step 70 - Spring Data REST

/pom.xml

Modified Lines

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-rest</artifactId>
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java Modified

```
  @ManyToMany(mappedBy="courses")
  @JsonIgnore
  private List<Student> students = new ArrayList<>();
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseSpringDataRepository.java Modified

Full File

```
@RepositoryRestResource(path="courses")
public interface CourseSpringDataRepository extends JpaRepository<Course, Long> {
  List<Course> findByNameAndId(String name, Long id);

  List<Course> findByName(String name);

  List<Course> countByName(String name);

  List<Course> findByNameOrderByIdDesc(String name);

  List<Course> deleteByName(String name);

  @Query("Select  c  From Course c where name like '%100 Steps'")
  List<Course> courseWith100StepsInName();

  @Query(value = "Select  *  From Course c where name like '%100 Steps'", nativeQuery = true)
  List<Course> courseWith100StepsInNameUsingNativeQuery();

  @Query(name = "query_get_100_Step_courses")
  List<Course> courseWith100StepsInNameUsingNamedQuery();
}
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CourseSpringDataRepositoryTest.java

```
  @Test
  public void pagination() {
    PageRequest pageRequest = PageRequest.of(0, 3);
    Page<Course> firstPage = repository.findAll(pageRequest);
    logger.info("First Page -> {} ", firstPage.getContent());
```

```
    Pageable secondPageable = firstPage.nextPageable();
    Page<Course> secondPage = repository.findAll(secondPageable);
    logger.info("Second Page -> {} ", secondPage.getContent());
  }

  @Test
  public void findUsingName() {
    logger.info("FindByName -> {} ", repository.findByName("JPA in 50 Steps"));
  }

  @Test
  public void findUsingStudentsName() {
    logger.info("findUsingStudentsName -> {} ", repository.findByName("Ranga"));
  }
```

Step 71 - Introduction to Caching

Step 72 - Hibernate and JPA Caching - First Level Cache

Step 73 - Hibernate and JPA Caching - Basics of Second Level Cache with EhCache

Step 74 - Hibernate and JPA Caching - Second Level Cache Part 2

/pom.xml

Modified Lines

```
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-ehcache</artifactId>
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java Modified

```
@Entity
@NamedQueries(value = {
    @NamedQuery(name = "query_get_all_courses",
        query = "Select  c  From Course c"),
    @NamedQuery(name = "query_get_100_Step_courses",
    query = "Select  c  From Course c where name like '%100 Steps'") })
@Cacheable
public class Course {
```

/src/main/resources/application.properties Modified Lines

```
# Second Level Cache - Ehcache
#1. enable second level cache
spring.jpa.properties.hibernate.cache.use_second_level_cache=true
#2. specify the caching framework - EhCache
spring.jpa.properties.hibernate.cache.region.factory_class=org.hibernate.cache.ehcache.EhCacheRegionFactory
#3. Only cache what I tell to cache.
spring.jpa.properties.javax.persistence.sharedCache.mode=ENABLE_SELECTIVE
logging.level.net.sf.ehcache=debug
#4. What data to cache?
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepositoryTest.java Modified

```
  @Test
  public void findById_firstLevelCacheDemo() {

    Course course = repository.findById(10001L);
    logger.info("First Course Retrieved {}", course);

    Course course1 = repository.findById(10001L);
    logger.info("First Course Retrieved again {}", course1);

    assertEquals("JPA in 50 Steps", course.getName());

    assertEquals("JPA in 50 Steps", course1.getName());
  }
```

Step 75 - Hibernate Tips - Hibernate Soft Deletes - @SQLDelete and @Where

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java Modified

```
@Entity
@NamedQueries(value = {
    @NamedQuery(name = "query_get_all_courses",
        query = "Select  c  From Course c"),
    @NamedQuery(name = "query_get_100_Step_courses",
    query = "Select  c  From Course c where name like '%100 Steps'") })
@Cacheable
@SQLDelete(sql="update course set is_deleted=true where id=?")
```

```
@Where(clause="is_deleted = false")
public class Course {

  //Other Fields

  private boolean isDeleted;
```

/src/main/resources/data.sql Modified Lines

```
insert into course(id, name, created_date, last_updated_date,is_deleted)
values(10001,'JPA in 50 Steps', sysdate(), sysdate(),false);
insert into course(id, name, created_date, last_updated_date,is_deleted)
values(10002,'Spring in 50 Steps', sysdate(), sysdate(),false);
insert into course(id, name, created_date, last_updated_date,is_deleted)
values(10003,'Spring Boot in 100 Steps', sysdate(), sysdate(),false);
```

Step 76 - Hibernate Soft Deletes - Part 2

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java Modified

```
  @PreRemove
  private void preRemove(){
    LOGGER.info("Setting isDeleted to True");
    this.isDeleted = true;
  }
```

Step 77 - JPA Entity Life Cycle Methods

Step 78 - Using Embedded and Embeddable with JPA

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Address.java New

```
package com.in28minutes.jpa.hibernate.demo.entity;

import javax.persistence.Embeddable;

@Embeddable
public class Address {
  protected Address() {}

  public Address(String line1, String line2, String city) {
    super();
    this.line1 = line1;
    this.line2 = line2;
    this.city = city;
  }

  private String line1;
  private String line2;
  private String city;


}
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Student.java Modified

```
  @Embedded
  private Address address;

  @ManyToMany
  @JoinTable(name = "STUDENT_COURSE", joinColumns = @JoinColumn(name = "STUDENT_ID"), inverseJoinColumns = @Joi
  private List<Course> courses = new ArrayList<>();
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/StudentRepositoryTest.java Modified

```
  @Test
  @Transactional
  public void setAddressDetails() {
    Student student = em.find(Student.class, 20001L);
    student.setAddress(new Address("No 101", "Some Street", "Hyderabad"));
    em.flush();
  }
```

Step 79 - Using Enums with JPA

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Review.java Modified

```
  @Enumerated(EnumType.STRING)
  private ReviewRating rating;

  protected Review() {
  }

  public Review(ReviewRating rating, String description) {
    this.rating = rating;
    this.description = description;
  }
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/ReviewRating.java

```java
package com.in28minutes.jpa.hibernate.demo.entity;

public enum ReviewRating {
  ZERO, ONE, TWO, THREE, FOUR, FIVE
}
```

/src/main/java/com/in28minutes/jpa/hibernate/demo/repository/CourseRepository.java Modified

```
  public void addHardcodedReviewsForCourse() {
    //get the course 10003
    Course course = findById(10003L);
    logger.info("course.getReviews() -> {}", course.getReviews());

    //add 2 reviews to it
    Review review1 = new Review(ReviewRating.FIVE, "Great Hands-on Stuff.");
    Review review2 = new Review(ReviewRating.FIVE, "Hatsoff.");

    //setting the relationship
    course.addReview(review1);
    review1.setCourse(course);

    course.addReview(review2);
    review2.setCourse(course);

    //save it to the database
    em.persist(review1);
    em.persist(review2);
  }
```

/src/main/resources/data.sql Modified Lines

```
insert into review(id,rating,description,course_id)
values(50001,'FIVE', 'Great Course',10001);
insert into review(id,rating,description,course_id)
values(50002,'FOUR', 'Wonderful Course',10001);
insert into review(id,rating,description,course_id)
values(50003,'FIVE', 'Awesome Course',10003);
```

Step 80 - JPA Tip - Be cautious with toString method implementations

Step 81 - JPA Tip - When do you use JPA?

Step 82 - Performance Tuning - Measure before Tuning

Step 83 - Performance Tuning - Indexes

Step 84 - Performance Tuning - Use Appropriate Caching

Step 85 - Performance Tuning - Eager vs Lazy Fetch

Step 86 - Performance Tuning - Avoid N+1 Problems

/src/main/java/com/in28minutes/jpa/hibernate/demo/entity/Course.java Modified

```
@Entity
@NamedQueries(value = {
    @NamedQuery(name = "query_get_all_courses",
        query = "Select  c  From Course c"),
    @NamedQuery(name = "query_get_all_courses_join_fetch",
    query = "Select  c  From Course c JOIN FETCH c.students s"),
    @NamedQuery(name = "query_get_100_Step_courses",
    query = "Select  c  From Course c where name like '%100 Steps'") })
@Cacheable
@SQLDelete(sql="update course set is_deleted=true where id=?")
@Where(clause="is_deleted = false")
public class Course {
```

/src/test/java/com/in28minutes/jpa/hibernate/demo/repository/PerformanceTuningTest.java New

```
package com.in28minutes.jpa.hibernate.demo.repository;

import java.util.List;

import javax.persistence.EntityGraph;
import javax.persistence.EntityManager;
import javax.persistence.Subgraph;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.transaction.annotation.Transactional;

import com.in28minutes.jpa.hibernate.demo.DemoApplication;
import com.in28minutes.jpa.hibernate.demo.entity.Course;

@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class)
public class PerformanceTuningTest {

  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  EntityManager em;

  @Test
  @Transactional
  public void creatingNPlusOneProblem() {
    List<Course> courses = em
        .createNamedQuery("query_get_all_courses", Course.class)
        .getResultList();
    for(Course course:courses){
      logger.info("Course -> {} Students -> {}",course, course.getStudents());
    }
  }

  @Test
  @Transactional
  public void solvingNPlusOneProblem_EntityGraph() {

    EntityGraph<Course> entityGraph = em.createEntityGraph(Course.class);
    Subgraph<Object> subGraph = entityGraph.addSubgraph("students");

    List<Course> courses = em
        .createNamedQuery("query_get_all_courses", Course.class)
        .setHint("javax.persistence.loadgraph", entityGraph)
        .getResultList();

    for(Course course:courses){
      logger.info("Course -> {} Students -> {}",course, course.getStudents());
    }
  }

  @Test
  @Transactional
  public void solvingNPlusOneProblem_JoinFetch() {
    List<Course> courses = em
        .createNamedQuery("query_get_all_courses_join_fetch", Course.class)
        .getResultList();
    for(Course course:courses){
      logger.info("Course -> {} Students -> {}",course, course.getStudents());
    }
  }

}
```

FAQ 1 - When does Hibernate send updates to the database?

FAQ 2 - When do we need @Transactional in an Unit Test?

FAQ 3 - Do read only methods need a transaction?

FAQ 4 - Why do we use @DirtiesContext in an Unit Test?

FAQ 5 - How to connect to a different database with Spring Boot?

FAQ 6 - Approach to design great applications with JPA?

FAQ 7 - Good Practices for developing JPA Applications

FAQ

- Do Read Only methods need a transaction?
    - Entities - User, Comment ``` @Transactional List someReadOnlyMethod() {

  User user = em.find(User.class, 1L);

  List comments = user.getComments();//

  return comments; } ```

- Why do we need @Transactional in Unit Tests some times?
    - Unit Test -> Repository -> EntityManager
    - Unit Test -> EntityManager
- When does Hibernate fire queries down to database? ``` @Transactional void someMethodWithChange() {

  //Create Objects em.persist(user1); em.persist(user2);

  em.flush();

  //Change user1 //Change user2 } //all changes are saved down to the database!

```
## Hibernate Mapping



#### Entity Manager
- detach method
- clear method
- flush method

#### Play with Annotations
- @Table : Indicates table name.
- @Column : Defining Constraints on Columns.
- @NamedQueries Indicates list of named queries.
- @NamedQuery Indicates a Query using static name.
- @CreationTimestamp
- @UpdateTimestamp
- @Transient : Column will not be persisted.
- @PostLoad

## Basics of JPQL
- Discuss the basics here
```

@NamedQueries({ @NamedQuery(name="name1", query="Query1"), @NamedQuery(name="name2", query="Query2"), })

```
- We can discuss the stuff needing Join after discussing relationships

#### Native Queries
- Database specific feature

#### Relationships
- Three types of relationships

#### Inheritance
- best performance - single table strategy
- best data integrity - joined

- JPQL queries with Joins
  - basic_empty_courses
  - basic_courses_with_min_three_students

#### Criteria API



// 1. Use Criteria Builder to create a Criteria Query returning the
// expected result object
// 2. Define roots for tables which are involved in the query
// 3. Define Predicates etc using Criteria Builder
// 4. Add Predicates etc to the Criteria Query
// 5. Build the TypedQuery using the entity manager and criteria query ```
```

Transaction Management

- Spring vs JPA @Transactional
- Isolation Levels
- Transactions have importance even within read-only context - like specifying a database isolation-level. We strongly recommend that all database operations occur within the scope of some transaction.

Advanced JPQL

- basic_courses_order_by
- join
- left_outer_join
- cross_join

Advanced Criterial Query

- join
- left_outer_join

Spring Data JPA

- Spring Data REST

Caching

- First Level Cache Demo

- Second Level Cache Demo

Hibernate Tips

- Implementing Soft Deletes - @Where and @SQLDelete
- Embedded Entities
- Using Enumerations With JPA
- Be cautious about toString
- Be cautious about Eager fetching on both sides of a relationship
- When do you use JPA?
    - SQL Database + Static Domain Model + Mostly CRUD + Mostly Simple Queries/Mappings and Updates with few Stored Procedures

Performance Tuning

- Zero Performance Tuning without Statistics. Check Stats in atleast one environment.
- Do not use JPA/Hibernate for Database intensive Batch Operations - Use Stored Procedures
- Add the right indexes on the database - Execution Plan
- Use Appropriate Caching
    - Be careful about the size of First Level Cache
- Eager vs Lazy Fetch - Use Lazy fetching mostly
    - Remember that all mapping *ToOne (@ManyToOne and @OneToOne) are EAGER by default.
- Avoid N+1
    - Entity Graph & Named Entity Graphs & Dynamic Entity Graphs
    - Join Fetch Clause
- Use Pagination & Batch Updates

```
query.setFirstResult(0);
query.setMaxResults(10);
```

- @Immutable - zero dirty checks!
- Read only transactions

Complete Code Example

Hibernate/JPA in Depth

/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.in28minutes.jpa</groupId>
  <artifactId>jpa-demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>jpa-demo</name>
  <description>Demo project for Spring Boot</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-jpamodelgen</artifactId>
    </dependency>

    <dependency>
      <groupId>org.hibernate</groupId>
```

```
      <artifactId>hibernate-ehcache</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>

      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
          <compilerArguments>
            <processor>org.hibernate.jpamodelgen.JPAMetaModelEntityProcessor</processor>
          </compilerArguments>
        </configuration>
      </plugin>

      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>build-helper-maven-plugin</artifactId>
        <executions>
          <execution>
            <phase>process-sources</phase>
            <configuration>
              <sources>
                <source>${project.build.directory}/generated-sources/annotations</source>
              </sources>
            </configuration>
            <goals>
              <goal>add-source</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>

  <repositories>
    <repository>
      <id>spring-snapshots</id>
      <name>Spring Snapshots</name>
      <url>https://repo.spring.io/snapshot</url>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </repository>
    <repository>
      <id>spring-milestones</id>
      <name>Spring Milestones</name>
      <url>https://repo.spring.io/milestone</url>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      <id>spring-snapshots</id>
      <name>Spring Snapshots</name>
      <url>https://repo.spring.io/snapshot</url>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </pluginRepository>
    <pluginRepository>
      <id>spring-milestones</id>
      <name>Spring Milestones</name>
      <url>https://repo.spring.io/milestone</url>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
    <pluginRepository>
      <id>maven-annotation</id>
      <url>http://maven-annotation-plugin.googlecode.com/svn/trunk/mavenrepo</url>
    </pluginRepository>
  </pluginRepositories>


</project>
```

/src/main/java/com/in28minutes/jpa/jpademo/controller/CourseController.java

```
package com.in28minutes.jpa.jpademo.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import com.in28minutes.jpa.jpademo.relationships.entity.Course;
import com.in28minutes.jpa.jpademo.relationships.repository.CourseRepository;

@RestController
public class CourseController {

  @Autowired
  CourseRepository repository;

  @GetMapping("/courses/{id}")
  public  Course retrieveCourse(@PathVariable long id){
    return repository.retrieveCourse(id);
  }
}
```

/src/main/java/com/in28minutes/jpa/jpademo/embedded/entity/Name.java

```
package com.in28minutes.jpa.jpademo.embedded.entity;

import javax.persistence.Embeddable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
//@Embeddable
public class Name {

  @Id
  @GeneratedValue
  protected Long id;
  protected String firstName;
  protected String middleName;
  protected String lastName;
}
```

/src/main/java/com/in28minutes/jpa/jpademo/embedded/entity/Person.java

```
package com.in28minutes.jpa.jpademo.embedded.entity;

import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;

@Entity
public class Person {
  @Id
  @GeneratedValue
  protected Long id;

  @OneToOne
  //@Embedded
  protected Name name;

}
```

/src/main/java/com/in28minutes/jpa/jpademo/inheritence/entity/Employee.java

```
package com.in28minutes.jpa.jpademo.inheritence.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.MappedSuperclass;

//@MappedSuperclass
```

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
// @DiscriminatorColumn(name = "disc_type")
public abstract class Employee {

  public Employee() {
  }

  public Employee(String name) {
    this.name = name;
  }

  @GeneratedValue
  @Id
  protected Integer id;

  private String name;

}
```

/src/main/java/com/in28minutes/jpa/jpademo/inheritence/entity/FullTimeEmployee.java

```
package com.in28minutes.jpa.jpademo.inheritance.entity;

import java.math.BigDecimal;

import javax.persistence.Entity;

@Entity
public class FullTimeEmployee extends Employee {
  public FullTimeEmployee(){}

  public FullTimeEmployee(String name, BigDecimal salary) {
    super(name);
    this.salary = salary;
  }

  protected BigDecimal salary;
}
```

/src/main/java/com/in28minutes/jpa/jpademo/inheritence/entity/PartTimeEmployee.java

```
package com.in28minutes.jpa.jpademo.inheritance.entity;

import java.math.BigDecimal;

import javax.persistence.Entity;

@Entity
public class PartTimeEmployee extends Employee {

  public PartTimeEmployee(){}

  public PartTimeEmployee(String name, BigDecimal hourlyWage) {
    super(name);
    this.hourlyWage = hourlyWage;
  }

  protected BigDecimal hourlyWage;
}
```

/src/main/java/com/in28minutes/jpa/jpademo/inheritence/repository/EmployeeRepositor

```
package com.in28minutes.jpa.jpademo.inheritance.repository;

import java.util.List;

import javax.persistence.EntityManager;
import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.in28minutes.jpa.jpademo.inheritance.entity.Employee;

@Transactional
@Repository
public class EmployeeRepository {

  @Autowired
  EntityManager entityManager;
```

```
  public void insertEmployee(Employee employee) {
    entityManager.persist(employee);
  }

  public List<Employee> allEmployees() {
    return entityManager.createQuery("Select e from Employee e", Employee.class).getResultList();
  }


}
```

/src/main/java/com/in28minutes/jpa/jpademo/JpaDemoApplication.java

```
package com.in28minutes.jpa.jpademo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.in28minutes.jpa.jpademo.inheritence.repository.EmployeeRepository;
import com.in28minutes.jpa.jpademo.relationships.repository.StudentRepository;

@SpringBootApplication
public class JpaDemoApplication implements CommandLineRunner {

  @Autowired
  StudentRepository studentRepository;

  @Autowired
  EmployeeRepository employeeRepository;

  public static void main(String[] args) {
    SpringApplication.run(JpaDemoApplication.class, args);
  }

  @Override
  public void run(String... args) throws Exception {
  }

}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/entity/Course.java

```
package com.in28minutes.jpa.jpademo.relationships.entity;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

import javax.persistence.Cacheable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.NamedAttributeNode;
import javax.persistence.NamedEntityGraph;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

@Entity
@Table(name = "Course")
@NamedQuery(query = "select c from Course c", name = "QUERY_ALL_COURSES")
@NamedEntityGraph(name = "graph.CourseAndStudents",
attributeNodes = @NamedAttributeNode(value = "students"/*, subgraph = "students"),*/)
/*subgraphs = @NamedSubgraph(name = "students", attributeNodes = @NamedAttributeNode("passport"))*/)
@Cacheable
public class Course {

  public Course() {
  }

  public Course(String name) {
    super();
    this.name = name;
  }

  @Id
  @GeneratedValue
  protected Long id;
```

```
  protected String name;

  // @OneToMany
  @OneToMany(mappedBy = "course")
  protected List<Review> reviews = new ArrayList<>();

  @ManyToMany
  // @JoinTable(name = "COURSE_STUDENT",
  // joinColumns = @JoinColumn(name = "COURSE_ID"),
  // inverseJoinColumns = @JoinColumn(name = "STUDENT_ID"))
  protected List<Student> students = new ArrayList<>();

  @CreationTimestamp
  private LocalDateTime createDateTime;

  private LocalDate activeFrom;

  @UpdateTimestamp
  private LocalDateTime updateDateTime;

  public Long getId() {
    return id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public List<Review> getReviews() {
    return reviews;
  }

  public void addReview(Review review) {
    this.reviews.add(review);
  }

  public List<Student> getStudents() {
    return students;
  }

  public void addStudent(Student student) {
    this.students.add(student);
  }

  public LocalDate getActiveFrom() {
    return activeFrom;
  }

  public void setActiveFrom(LocalDate activeFrom) {
    this.activeFrom = activeFrom;
  }

  @Override
  public String toString() {
    return String.format("Course[%s]", name);
  }

}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/entity/Passport.java

```
package com.in28minutes.jpa.jpademo.relationships.entity;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.validation.constraints.NotNull;

import org.hibernate.annotations.Where;

@Entity
@Where(clause = "1=1")
public class Passport {

  protected Passport() {
  }

  public Passport(String number) {
    super();
    this.number = number;
  }

  @Id
  @GeneratedValue
  protected Long id;
```

```
  @NotNull
  protected String number;

  // Inverse Relationship
  // bi-directional OneToOne relationship
  // Column will not be created in the table
  // Try removing mappedBy = "passport" => You will see a student_id column
  // will be created in passport
  // @OneToOne
  @OneToOne(fetch = FetchType.LAZY, mappedBy = "passport")
  protected Student student;

  public Long getId() {
    return id;
  }

  public String getNumber() {
    return number;
  }

  public void setNumber(String number) {
    this.number = number;
  }

  public Student getStudent() {
    return student;
  }

  public void setStudent(Student student) {
    this.student = student;
  }
}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/entity/Review.java

```
package com.in28minutes.jpa.jpademo.relationships.entity;

import javax.persistence.Entity;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

@Entity
public class Review {

  private Review() {}

  public Review(ReviewRating rating, String description) {
    super();
    this.rating = rating;
    this.description = description;
  }

  @Id
  @GeneratedValue
  protected Long id;

  @Enumerated
  protected ReviewRating rating;

  protected String description;

  @ManyToOne
  // @JoinColumn(name="COURSE_ID")
  protected Course course;

  public Long getId() {
    return id;
  }

  public ReviewRating getRating() {
    return rating;
  }

  public void setRating(ReviewRating rating) {
    this.rating = rating;
  }

  public String getDescription() {
    return description;
  }

  public void setDescription(String description) {
    this.description = description;
  }

  public Course getCourse() {
    return course;
  }

  public void setCourse(Course course) {
```

```
    this.course = course;
  }

}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/entity/ReviewRating.java

```
package com.in28minutes.jpa.jpademo.relationships.entity;
public enum ReviewRating {
  ONE,TWO,THREE,FOUR,FIVE
}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/entity/Student.java

```
package com.in28minutes.jpa.jpademo.relationships.entity;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.MapKeyColumn;
import javax.persistence.MapKeyEnumerated;
import javax.persistence.OneToOne;

import org.hibernate.annotations.SQLDelete;

@Entity
@SQLDelete(sql = "UPDATE student SET state = 'DELETED' WHERE id = ?")
public class Student {

  private Student() {
  }

  public Student(String name, StudentType studentType) {
    super();
    this.name = name;
    this.studentType = studentType;
  }

  @Id
  @GeneratedValue
  protected Long id;

  protected String name;

  @OneToOne(fetch = FetchType.LAZY)
  protected Passport passport;

  // @ManyToMany
  @ManyToMany(mappedBy = "students")
  protected List<Course> courses = new ArrayList<>();

  // @Enumerated
  @Enumerated(EnumType.STRING)
  private StudentType studentType;

  @ElementCollection
  @CollectionTable(name = "STUDENT_PHONE")
  @MapKeyEnumerated(EnumType.STRING)
  @MapKeyColumn(name = "PHONE_TYPE")
  @Column(name = "PHONE_NUM")
  private Map<PhoneType, String> phoneNumbers;

  enum PhoneType {
    Home, Mobile, Work
  }

  public Long getId() {
    return id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
```

```
    this.name = name;
  }

  public Passport getPassport() {
    return passport;
  }

  public void setPassport(Passport passport) {
    this.passport = passport;
  }

  public List<Course> getCourses() {
    return courses;
  }

  public void addCourse(Course course) {
    courses.add(course);
  }

  public StudentType getStudentType() {
    return studentType;
  }

  public void setStudentType(StudentType studentType) {
    this.studentType = studentType;
  }

  public Map<PhoneType, String> getPhoneNumbers() {
    return phoneNumbers;
  }

  public void addPhoneNumber(PhoneType phoneType, String number) {
    phoneNumbers.put(phoneType, number);
  }

  @Override
  public String toString() {
    return String.format("Student[%s]", name);
  }

}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/entity/StudentType.java

```
package com.in28minutes.jpa.jpademo.relationships.entity;
public enum StudentType {
  FullTime, PartTime
}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/repository/CourseRepository.

```
package com.in28minutes.jpa.jpademo.relationships.repository;

import java.util.List;

import javax.persistence.EntityGraph;
import javax.persistence.EntityManager;
import javax.persistence.Subgraph;
import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.in28minutes.jpa.jpademo.relationships.entity.Course;
import com.in28minutes.jpa.jpademo.relationships.entity.Course_;
import com.in28minutes.jpa.jpademo.relationships.entity.Review;
import com.in28minutes.jpa.jpademo.relationships.entity.Student;

@Repository
@Transactional
public class CourseRepository {

  @Autowired
  EntityManager entityManager;

  public void createCourse(Course course) {
    entityManager.persist(course);
  }

  public Course retrieveCourse(Long id) {
    return entityManager.find(Course.class, id);
  }

  public void printAllCourseAndStudents() {
    EntityGraph graph = entityManager.getEntityGraph("graph.CourseAndStudents");
    List<Course> courses = entityManager.createQuery("Select c from Course c", Course.class)
        .setHint("javax.persistence.loadgraph", graph).getResultList();
```

```
    for (Course course : courses) {
      System.out.println(course + " " + course.getStudents());
    }
  }

  public void printAllCourseAndStudentsDynamicSubgraph() {
    EntityGraph<Course> graph = entityManager.createEntityGraph(Course.class);
    Subgraph<List<Student>> bookSubGraph = graph.addSubgraph(Course_.students);

    List<Course> courses = entityManager.createQuery("Select c from Course c", Course.class)
        .setHint("javax.persistence.loadgraph", graph).getResultList();

    for (Course course : courses) {
      System.out.println(course + " " + course.getStudents());
    }
  }

  public void printAllCourseAndStudentsJoinFetch() {
    List<Course> courses = entityManager.createQuery("Select c from Course c JOIN FETCH c.students s", Course.c
        .getResultList();
    for (Course course : courses) {
      System.out.println(course + " " + course.getStudents());
    }
  }

  public void updateCourse(Course course) {
    entityManager.merge(course);
  }

  public void createCourseWithStudents(Course course, Student... students) {

    for (Student student : students) {
      course.addStudent(student);
      student.addCourse(course);
      if (student.getId() == null) {
        entityManager.persist(student);
      }
    }

    createCourse(course);

  }

  public void createReviewsForCourse(Course course, Review... reviews) {
    for (Review review : reviews) {
      course.addReview(review);
      review.setCourse(course);
      if (review.getId() == null) {
        entityManager.persist(review);
      }
    }
  }

}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/repository/EntityManagerRe

```
package com.in28minutes.jpa.jpademo.relationships.repository;

import javax.persistence.EntityManager;
import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.in28minutes.jpa.jpademo.relationships.entity.Passport;

@Repository
@Transactional
public class EntityManagerRepository {

  @Autowired
  EntityManager entityManager;

  public void doSomething() {
    Passport passport = new Passport("E123456");
    entityManager.persist(passport);
    entityManager.flush();
    passport.setNumber("E123457");
    // entityManager.clear();
    // entityManager.detach(passport);
    // entityManager.refresh(passport);
    // entityManager.remove(passport);
    // entityManager.merge(passport);
    // Queries
    // Entity Graphs
  }
}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/repository/StudentRepository

```
package com.in28minutes.jpa.jpademo.relationships.repository;

import javax.persistence.EntityManager;
import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.in28minutes.jpa.jpademo.relationships.entity.Course;
import com.in28minutes.jpa.jpademo.relationships.entity.Passport;
import com.in28minutes.jpa.jpademo.relationships.entity.Review;
import com.in28minutes.jpa.jpademo.relationships.entity.Student;

@Repository
@Transactional
public class StudentRepository {

  @Autowired
  EntityManager entityManager;

  public void createStudentWithPassport(Student student, Passport passport) {
    student.setPassport(passport);
    //passport.setStudent(student);
    entityManager.persist(passport);
    entityManager.persist(student);

  }
}
```

/src/main/java/com/in28minutes/jpa/jpademo/relationships/repository/TransactionManag

```
package com.in28minutes.jpa.jpademo.relationships.repository;

import javax.persistence.EntityManager;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.in28minutes.jpa.jpademo.relationships.entity.Passport;

@Repository
@Transactional
public class TransactionManagementRepository {

  @Autowired
  EntityManager entityManager;

  public void doSomething() {
    Passport passport1 = new Passport("E123456");
    entityManager.persist(passport1);
    Passport passport2 = new Passport(null);
    entityManager.persist(passport2);
  }
}
```

/src/main/resources/application.properties

```
spring.h2.console.enabled=true
spring.jpa.properties.hibernate.generate_statistics=true
logging.level.org.hibernate.stat=debug
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
#logging.level.org.hibernate.type=TRACE
spring.jpa.properties.hibernate.cache.use_second_level_cache=true
spring.jpa.properties.hibernate.cache.region.factory_class=org.hibernate.cache.ehcache.EhCacheRegionFactory
spring.jpa.properties.javax.persistence.sharedCache.mode=ENABLE_SELECTIVE
logging.level.net.sf.ehcache=debug
```

/src/main/resources/data.sql

```
insert into course(id, name)
values(10101,'Caching in 100 Steps');
```

/src/test/java/com/in28minutes/jpa/jpademo/CriteriaQueryDemoApplicationTest.java

```
package com.in28minutes.jpa.jpademo;

import static org.junit.Assert.assertEquals;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Join;
import javax.persistence.criteria.JoinType;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.jpademo.relationships.entity.Course;
import com.in28minutes.jpa.jpademo.relationships.entity.Course_;
import com.in28minutes.jpa.jpademo.relationships.entity.Student;

@RunWith(SpringRunner.class)
@SpringBootTest // (webEnvironment = WebEnvironment.RANDOM_PORT)
public class CriteriaQueryDemoApplicationTest {

  // @LocalServerPort
  // String port;

  @Autowired
  EntityManager entityManager;

  @Test
  public void basic() {

    CriteriaBuilder cb = entityManager.getCriteriaBuilder();

    CriteriaQuery<Course> cq = cb.createQuery(Course.class);
    Root<Course> root = cq.from(Course.class);

    TypedQuery<Course> query = entityManager.createQuery(cq.select(root));

    List<Course> courses = query.getResultList();
    System.out.println(courses);
  }

  @Test
  public void basic2() {
    CriteriaBuilder cb = entityManager.getCriteriaBuilder();
    CriteriaQuery<Course> cq = cb.createQuery(Course.class);

    Root<Course> course = cq.from(Course.class);
    Predicate condition = cb.like(course.get(Course_.name), "%100 Steps");
    cq.where(condition);

    TypedQuery<Course> query = entityManager.createQuery(cq.select(course));

    List<Course> courses = query.getResultList();
    System.out.println(courses);

    assertEquals(2, courses.size());
    System.out.println(courses);
  }

  @Test
  public void basic_empty_courses() {

    CriteriaBuilder cb = entityManager.getCriteriaBuilder();
    CriteriaQuery<Course> cq = cb.createQuery(Course.class);

    Root<Course> course = cq.from(Course.class);
    Predicate condition = cb.isEmpty(course.get(Course_.students));
    cq.where(condition);

    TypedQuery<Course> query = entityManager.createQuery(cq.select(course));

    List<Course> courses = query.getResultList();
    System.out.println(courses);

    assertEquals(1, courses.size());
    System.out.println(courses);
  }

  @Test
  public void basic_courses_order_by() {
    CriteriaBuilder cb = entityManager.getCriteriaBuilder();

    CriteriaQuery<Course> cq = cb.createQuery(Course.class);
    Root<Course> course = cq.from(Course.class);
    cq.orderBy(cb.desc(course.get(Course_.name)));
    TypedQuery<Course> query = entityManager.createQuery(cq.select(course));
```

```
    List<Course> courses = query.getResultList();
    System.out.println(courses);

  }

  @Test
  public void join() {
    CriteriaBuilder cb = entityManager.getCriteriaBuilder();

    CriteriaQuery<Course> cq = cb.createQuery(Course.class);
    Root<Course> course = cq.from(Course.class);
    Join<Course, Student> student = course.join(Course_.students);

    TypedQuery<Course> query = entityManager.createQuery(cq.select(course));

    List<Course> courses = query.getResultList();
    System.out.println(courses);
    assertEquals(5, courses.size());
  }

  @Test
  public void left_outer_join() {
    CriteriaBuilder cb = entityManager.getCriteriaBuilder();

    CriteriaQuery<Course> cq = cb.createQuery(Course.class);
    Root<Course> course = cq.from(Course.class);
    Join<Course, Student> student = course.join(Course_.students, JoinType.LEFT);

    TypedQuery<Course> query = entityManager.createQuery(cq.select(course));

    List<Course> courses = query.getResultList();
    System.out.println(courses);
    assertEquals(6, courses.size());
  }
}
```

/src/test/java/com/in28minutes/jpa/jpademo/EntityManagerDemoApplicationTests.java

```
package com.in28minutes.jpa.jpademo;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.jpademo.relationships.repository.EntityManagerRepository;

@RunWith(SpringRunner.class)
@SpringBootTest // (webEnvironment = WebEnvironment.RANDOM_PORT)
public class EntityManagerDemoApplicationTests {

  @Autowired
  EntityManagerRepository entityManagerRepository;

  @Test
  public void someTest() {
    entityManagerRepository.doSomething();
  }
}
```

/src/test/java/com/in28minutes/jpa/jpademo/InheritanceDemoApplicationTest.java

```
package com.in28minutes.jpa.jpademo;

import java.math.BigDecimal;

import javax.persistence.EntityManager;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.jpademo.inheritence.entity.FullTimeEmployee;
import com.in28minutes.jpa.jpademo.inheritence.entity.PartTimeEmployee;
import com.in28minutes.jpa.jpademo.inheritence.repository.EmployeeRepository;

@RunWith(SpringRunner.class)
@SpringBootTest // (webEnvironment = WebEnvironment.RANDOM_PORT)
public class InheritanceDemoApplicationTest {

  // @LocalServerPort
  // String port;
```

```
  @Autowired
  EntityManager entityManager;

  @Autowired
  EmployeeRepository employeeRepository;

  @Test
  public void basic() {
    employeeRepository.insertEmployee(new PartTimeEmployee("PartTimeEE", new BigDecimal(100)));
    employeeRepository.insertEmployee(new FullTimeEmployee("FullTimeEE", new BigDecimal(10)));
    System.out.println(employeeRepository.allEmployees());
  }
}
```

/src/test/java/com/in28minutes/jpa/jpademo/JpaDemoApplicationTests.java

```
package com.in28minutes.jpa.jpademo;

import java.time.LocalDate;
import java.time.Month;

import javax.persistence.EntityManager;
import javax.transaction.Transactional;

import org.junit.After;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.jpademo.relationships.entity.Course;
import com.in28minutes.jpa.jpademo.relationships.entity.Passport;
import com.in28minutes.jpa.jpademo.relationships.entity.Review;
import com.in28minutes.jpa.jpademo.relationships.entity.ReviewRating;
import com.in28minutes.jpa.jpademo.relationships.entity.Student;
import com.in28minutes.jpa.jpademo.relationships.entity.StudentType;
import com.in28minutes.jpa.jpademo.relationships.repository.CourseRepository;
import com.in28minutes.jpa.jpademo.relationships.repository.StudentRepository;

@RunWith(SpringRunner.class)
@SpringBootTest // (webEnvironment = WebEnvironment.RANDOM_PORT)
public class JpaDemoApplicationTests {

  // @LocalServerPort
  // String port;

  @Autowired
  EntityManager entityManager;

  @Autowired
  CourseRepository courseRepository;

  @Autowired
  StudentRepository studentRepository;

  @Test
  public void createCourseWithStudents() {
    Student student = new Student("Ranga", StudentType.FullTime);
    courseRepository.createCourseWithStudents(new Course("Spring in 100 Steps"), student);
    courseRepository.createCourseWithStudents(new Course("Spring Boot in 100 Steps"), student);
  }

  @Test
  public void createReviewsForCourse() {
    Course course = new Course("JPA in 100 Steps");
    courseRepository.createCourse(course);
    courseRepository.createReviewsForCourse(course, new Review(ReviewRating.FIVE, "Awesome Course"),
        new Review(ReviewRating.FIVE, "Wow!"));

  }

  @Test
  public void createCourse() {
    courseRepository.createCourse(new Course("JPA in 100 Steps"));
  }

  @Test
  @Transactional
  public void createStudentWithPassport() {
    Student student = new Student("Ranga", StudentType.FullTime);
    Passport passport = new Passport("A12345678");
    studentRepository.createStudentWithPassport(student, passport);
  }

  @Test
  public void updateCourse() {
    Course course = courseRepository.retrieveCourse(10001L);
    course.setName("JPA in 100 Steps - updated");
    course.setActiveFrom(LocalDate.of(2018, Month.APRIL, 10));
    courseRepository.updateCourse(course);
  }
```

```
  @After
  public void printAllData() {
    System.out.println("Dummy");
  }
}
```

/src/test/java/com/in28minutes/jpa/jpademo/JPQLDemoApplicationTest.java

```
package com.in28minutes.jpa.jpademo;

import static org.junit.Assert.assertEquals;

import java.util.Arrays;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.Query;
import javax.persistence.TypedQuery;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.jpademo.relationships.entity.Course;
import com.in28minutes.jpa.jpademo.relationships.entity.Student;

@RunWith(SpringRunner.class)
@SpringBootTest // (webEnvironment = WebEnvironment.RANDOM_PORT)
public class JPQLDemoApplicationTest {

  // @LocalServerPort
  // String port;

  @Autowired
  EntityManager entityManager;

  @Test
  public void basic() {
    Query query = entityManager.createQuery("SELECT c FROM Course c");
    System.out.println(query.getResultList());
  }

  @Test
  public void basic_typed() {
    TypedQuery<Course> query = entityManager.createQuery("SELECT c FROM Course c", Course.class);
    List<Course> resultList = query.getResultList();
    System.out.println(resultList);
  }

  @Test
  public void basic2() {
    Query query = entityManager.createQuery("SELECT c FROM Course c WHERE c.name like '%100 Steps'");
    List resultList = query.getResultList();
    assertEquals(2, resultList.size());
    System.out.println(resultList);
  }

  @Test
  public void basic_empty_courses() {
    Query query = entityManager.createQuery("SELECT c FROM Course c WHERE c.students IS EMPTY");
    List resultList = query.getResultList();
    assertEquals(1, resultList.size());
    System.out.println(resultList);
  }

  @Test
  public void basic_courses_with_min_three_students() {
    Query query = entityManager.createQuery("SELECT c FROM Course c WHERE size(c.students) >= 3");
    List resultList = query.getResultList();
    assertEquals(1, resultList.size());
    System.out.println(resultList);
  }

  @Test
  public void basic_courses_order_by() {
    Query query = entityManager.createQuery("SELECT c FROM Course c ORDER BY size(c.students) DESC");
    List resultList = query.getResultList();
    assertEquals(3, resultList.size());
    System.out.println(resultList);
  }

  @Test
  public void basic3() {
    Query query = entityManager.createQuery("SELECT s FROM Student s WHERE s.passport.number like 'N%'");
    List resultList = query.getResultList();
    assertEquals(1, resultList.size());
    System.out.println(resultList);
  }
```

```
  @Test
  public void basic4() {
    Query query = entityManager.createQuery("SELECT s FROM Student s WHERE s.passport.number like 'N%'");
    List resultList = query.getResultList();
    assertEquals(1, resultList.size());
    System.out.println(resultList);
  }

  // BETWEEN 100 and 1000
  // IS NULL
  // upper, lower, trim, length
  // Group by, having


  @Test
  public void join() {
    Query query = entityManager.createQuery("SELECT c, s FROM Course c JOIN c.students s");
    List resultList = query.getResultList();
    System.out.println(resultList.get(1).getClass());
    assertEquals(5, resultList.size());
    System.out.println(resultList);
  }

  @Test
  public void left_outer_join() {
    Query query = entityManager.createQuery("SELECT c, s FROM Course c LEFT JOIN c.students s");
    List<Object[]> resultList = query.getResultList();
    assertEquals(6, resultList.size());
    for (Object[] result : resultList) {
      Course course = (Course) result[0];
      Student student = (Student) result[1];
      System.out.println(course + " " + student);
    }
  }

  @Test
  public void cross_join() {
    Query query = entityManager.createQuery("SELECT c, s FROM Course c, Student s");
    List resultList = query.getResultList();
    assertEquals(12, resultList.size());
    System.out.println(resultList);
  }

}
```

/src/test/java/com/in28minutes/jpa/jpademo/NativeQueriesDemoApplicationTest.java

```
package com.in28minutes.jpa.jpademo;

import static org.junit.Assert.assertEquals;

import java.util.Arrays;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.Query;
import javax.persistence.TypedQuery;
import javax.transaction.Transactional;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.jpademo.relationships.entity.Course;
import com.in28minutes.jpa.jpademo.relationships.entity.Student;

@RunWith(SpringRunner.class)
@SpringBootTest // (webEnvironment = WebEnvironment.RANDOM_PORT)
public class NativeQueriesDemoApplicationTest {

  // @LocalServerPort
  // String port;

  @Autowired
  EntityManager entityManager;

  @Test
  public void basic() {
    Query query = entityManager.createNativeQuery("SELECT * FROM Course c");
    System.out.println(query.getResultList());
  }

  @Test
  public void basic_with_parameter() {
    Query query = entityManager.createNativeQuery("SELECT * FROM Course c where c.id = ?");
    query.setParameter(1  , 10001L);

    List resultList = query.getResultList();
    assertEquals(1,resultList.size());
```

```
  System.out.println(resultList);
}

@Test
public void basic_with_named_parameter() {
  Query query = entityManager.createNativeQuery("SELECT * FROM Course c where c.id = :id");
  query.setParameter("id", 10001L);

  List resultList = query.getResultList();
  assertEquals(1,resultList.size());

  System.out.println(resultList);
}

@Test
public void basic_with_named_native_query() {
}

@Test
@Transactional
public void updating_a_number_of_rows() {
  Query query = entityManager.createNativeQuery("Update Course Set create_date_time=sysdate()");
  int executeUpdate = query.executeUpdate();
  System.out.println(executeUpdate);
}


@Test
public void basic_typed() {
  TypedQuery<Course> query = entityManager.createQuery("SELECT c FROM Course c", Course.class);
  List<Course> resultList = query.getResultList();
  System.out.println(resultList);
}

@Test
public void basic2() {
  Query query = entityManager.createQuery("SELECT c FROM Course c WHERE c.name like '%100 Steps'");
  List resultList = query.getResultList();
  assertEquals(2, resultList.size());
  System.out.println(resultList);
}

@Test
public void basic_empty_courses() {
  Query query = entityManager.createQuery("SELECT c FROM Course c WHERE c.students IS EMPTY");
  List resultList = query.getResultList();
  assertEquals(1, resultList.size());
  System.out.println(resultList);
}

@Test
public void basic_courses_with_min_three_students() {
  Query query = entityManager.createQuery("SELECT c FROM Course c WHERE size(c.students) >= 3");
  List resultList = query.getResultList();
  assertEquals(1, resultList.size());
  System.out.println(resultList);
}

@Test
public void basic_courses_order_by() {
  Query query = entityManager.createQuery("SELECT c FROM Course c ORDER BY size(c.students) DESC");
  List resultList = query.getResultList();
  assertEquals(3, resultList.size());
  System.out.println(resultList);
}

@Test
public void basic3() {
  Query query = entityManager.createQuery("SELECT s FROM Student s WHERE s.passport.number like 'N%'");
  List resultList = query.getResultList();
  assertEquals(1, resultList.size());
  System.out.println(resultList);
}

@Test
public void basic4() {
  Query query = entityManager.createQuery("SELECT s FROM Student s WHERE s.passport.number like 'N%'");
  List resultList = query.getResultList();
  assertEquals(1, resultList.size());
  System.out.println(resultList);
}

// BETWEEN 100 and 1000
// IS NULL
// upper, lower, trim, length
// Group by, having


@Test
public void join() {
  Query query = entityManager.createQuery("SELECT c, s FROM Course c JOIN c.students s");
  List resultList = query.getResultList();
  System.out.println(resultList.get(1).getClass());
  assertEquals(5, resultList.size());
  System.out.println(resultList);
}

@Test
```

```
  public void left_outer_join() {
    Query query = entityManager.createQuery("SELECT c, s FROM Course c LEFT JOIN c.students s");
    List<Object[]> resultList = query.getResultList();
    assertEquals(6, resultList.size());
    for (Object[] result : resultList) {
      Course course = (Course) result[0];
      Student student = (Student) result[1];
      System.out.println(course + " " + student);
    }
  }

  @Test
  public void cross_join() {
    Query query = entityManager.createQuery("SELECT c, s FROM Course c, Student s");
    List resultList = query.getResultList();
    assertEquals(12, resultList.size());
    System.out.println(resultList);
  }

}
```

/src/test/java/com/in28minutes/jpa/jpademo/PerformanceDemoApplicationTest.java

```
package com.in28minutes.jpa.jpademo;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.jpademo.relationships.repository.CourseRepository;

@RunWith(SpringRunner.class)
@SpringBootTest
public class PerformanceDemoApplicationTest {

  @Autowired
  CourseRepository courseRepository;

  @Test
  public void testNplus1(){
    courseRepository.printAllCourseAndStudents();
    //courseRepository.printAllCourseAndStudentsDynamicSubgraph();
    //courseRepository.printAllCourseAndStudentsJoinFetch();
  }


}
```

/src/test/java/com/in28minutes/jpa/jpademo/TransactionManagementDemoApplicationT

```
package com.in28minutes.jpa.jpademo;

import static org.junit.Assert.assertNull;

import javax.persistence.EntityManager;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.in28minutes.jpa.jpademo.relationships.entity.Passport;
import com.in28minutes.jpa.jpademo.relationships.repository.TransactionManagementRepository;

@RunWith(SpringRunner.class)
@SpringBootTest // (webEnvironment = WebEnvironment.RANDOM_PORT)
public class TransactionManagementDemoApplicationTests {

  @Autowired
  TransactionManagementRepository transactionManagementRepository;

  @Autowired
  EntityManager entityManager;

  @Test
  public void someTest() {

    try {
      transactionManagementRepository.doSomething();
    } catch (Exception e) { }

    assertNull(entityManager.find(Passport.class, 1L));
  }
}
```

/src/test/resources/data.sql

```
insert into passport(id, number)
values(40001, 'L123456');

insert into passport(id, number)
values(40002, 'M123456');

insert into passport(id, number)
values(40003, 'N123456');

insert into passport(id, number)
values(40004, 'O123456');


insert into course(id, name)
values(10001,'Spring in 100 Steps');

insert into course(id, name)
values(10002,'Spring Boot in 100 Steps');

insert into course(id, name)
values(10003,'JPA in 50 Steps');

insert into student(id, name,passport_id)
values(20001, 'Adam',40001);

insert into student(id, name,passport_id)
values(20002, 'Buck',40002);

insert into student(id, name,passport_id)
values(20003, 'Chris',40003);

insert into student(id, name,passport_id)
values(20004, 'Dennis',40004);


insert into course_students(courses_id,students_id)
values(10001,20001);

insert into course_students(courses_id,students_id)
values(10001,20002);

insert into course_students(courses_id,students_id)
values(10001,20003);

insert into course_students(courses_id,students_id)
values(10002,20001);

insert into course_students(courses_id,students_id)
values(10002,20002);
```