

- Step 1 : Object Relational Impedence Mismatch - Understanding the problem that JPA solves

Examples of Object Relational Impedence Mismatch

Lets consider a simple example - Employees and Tasks.

Each Employee can have multiple Tasks. Each Task can be shared by multiple Employees. There is a Many to Many relationship between them. Let's consider a few examples of impedance mismatch.

Example 1 : Task table below is mapped to Task Table. However, there are mismatches in column names.

```
public class Task {  
    private int id;  
  
    private String desc;  
  
    private Date targetDate;  
  
    private boolean isDone;  
  
    private List<Employee> employees;
```



```

    private int id;
    private String desc;
    private Date targetDate;
    private boolean isDone;
    private List<Employee> employees;
}

```

```

CREATE TABLE task
(
    id          INTEGER GENERATED BY DEFAULT AS IDENTITY,
    description VARCHAR(255),
    is_done     BOOLEAN,
    target_date TIMESTAMP,
    PRIMARY KEY (id)
)

```

Example 2: Relationships between objects are expressed in a different way compared with

JPA And Hibernate Tutorial For Beginners with Spring Boot and Spring Data JPA

Example 2 : Relationships between objects are expressed in a different way compared with relationship between tables.

Each Employee can have multiple Tasks. Each Task can be shared by multiple Employees. There is a Many to Many relationship between them.

```

public class Employee {
    //Some other code
    private List<Task> tasks;
}

public class Task {
    //Some other code
    private List<Employee> employees;
}

```

4:17 / 57:15

```

CREATE TABLE employee
(
    id          BIGINT NOT NULL,
    OTHER_COLUMNS
)

CREATE TABLE employee_tasks
(
    employees_id BIGINT NOT NULL,
    tasks_id     INTEGER NOT NULL
)

CREATE TABLE task
(
    id          INTEGER GENERATED BY DEFAULT AS IDENTITY,
    OTHER_COLUMNS
)

```

Example 3 : Some times multiple classes are mapped to a single table and vice-versa

Objects

```

public class Employee {
    //Other Employee Attributes
}

public class FullTimeEmployee extends Employee {
    protected Integer salary;
}

public class PartTimeEmployee extends Employee {
    protected Float hourlyWage;
}

```

Tables

```

CREATE TABLE employee
(

```

- Step 2 : World before JPA - JDBC, Spring JDBC and myBatis

JPA And Hibernate Tutorial For Beginners with Spring Boot and Spring Data JPA

Other approaches before JPA - JDBC, Spring JDBC & myBatis

Other approaches before JPA focused on queries and how to translate results from queries to objects.

Any approach using query typically does two things

- Setting parameters to the query. We need to read values from objects and set them as parameters to the query.
- Liquidation of results from the query. The results from the query need to be mapped to the beans.

JDBC

- JDBC stands for Java Database Connectivity
- It used concepts like Statement, PreparedStatement and ResultSet
- In the example below, the query used is `Update todo set user=?, desc=?, target_date=?, is_done=? where id=?`
- The values needed to execute the query are set into the query using different set

7:01 / 57:15

We would get the result of sql query in ResultSet and then get values from resultset and

put it in bean

```
JPA And Hibernate Tutorial For Beginners with Spring Boot and Spring Data JPA
PreparedStatement st = connection.prepareStatement(
    "SELECT * FROM TODO where id=?");

st.setInt(1, id);

ResultSet resultSet = st.executeQuery();

if (resultSet.next()) {
    Todo todo = new Todo();
    todo.setId(resultSet.getInt("id"));
    todo.setUser(resultSet.getString("user"));
    todo.setDesc(resultSet.getString("desc"));
    todo.setTargetDate(resultSet.getTimestamp("target_date"));
    return todo;
}

st.close();
connection.close();

return null;
```

Spring JDBC

- Spring JDBC provides a layer on top of JDBC
- It used concepts like JDBCTemplate
- Typically needs lesser number of lines compared to JDBC as following are simplified
 - mapping parameters to queries
 - liquidating resultsets to beans

Update Todo

```
jdbcTemplate
.update("Update todo set user=?, desc=?, target_date=?, is_done=? where id=?",
    todo.getUser(),
    todo.getDesc(),
    new Timestamp(todo.getTargetDate().getTime()),
    todo.isDone(),
    todo.getId());
```

Retrieve a Todo

```
@Override
public Todo retrieveTodo(int id) {

    return jdbcTemplate.queryForObject(
        "SELECT * FROM TODO where id=?",
        new Object[] { id }, new TodoMapper());

}
```

Reusable Row Mapper

```
// new BeanPropertyRowMapper(TodoMapper.class)
class TodoMapper implements RowMapper<Todo> {
    @Override
    public Todo mapRow(ResultSet rs, int rowNum)
        throws SQLException {
        Todo todo = new Todo();

        todo.setId(rs.getInt("id"));
        todo.setUser(rs.getString("user"));
    }
}
```

Press **esc** to exit full screen

myBatis

MyBatis removes the need for manually writing code to set parameters and retrieve results. It provides simple XML or Annotation based configuration to map Java POJOs to database.

We compare the approaches used to write queries below:

- JDBC or Spring JDBC - Update todo set user=?, desc=?, target_date=?, is_done=? where id=?
- myBatis - Update todo set user=#{user}, desc=#{desc}, target_date=#{targetDate}, is_done=#{isDone} where id=#{id}

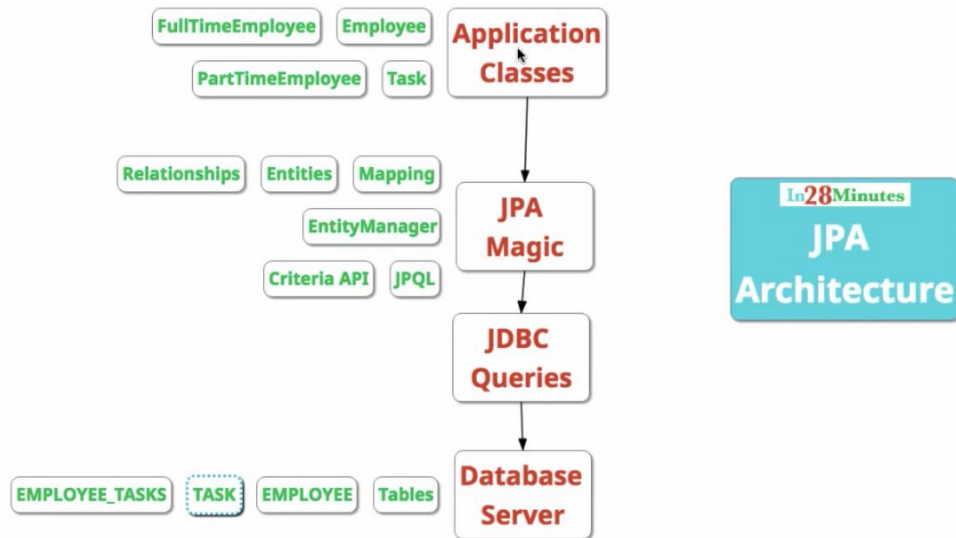
Update Todo and Retrieve Todo

```
@Mapper
public interface TodoMybatisService
    extends TodoDataService {
```

The fundamental thing for all three approaches was the fact they were based on queries. The Problem with writing big queries is that when relation between tables change ,queries had to be changed.

- Step 3 : Introduction to JPA

JPA provides mapping between classes and database using concept of Entities



Example 1

Task table below is mapped to `Task` Table. However, there are mismatches in column names. We use a few JPA annotations to do the mapping

- `@Table(name = "Task")`
- `@Id`
- `@GeneratedValue`
- `@Column(name = "description")`

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "Task")
public class Task {
    @Id
    @GeneratedValue
    private int id;
```

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "Task")
public class Task {
    @Id
    @GeneratedValue
    private int id;

    @Column(name = "description")
    private String desc;

    @Column(name = "target_date")
    private Date targetDate;

    @Column(name = "is_done")
    private boolean isDone;
}
```



```
@Column(name = "description")
private String desc;

@Column(name = "target_date")
private Date targetDate;

@Column(name = "is_done")
private boolean isDone;
}
```

```
CREATE TABLE task
(
    id          INTEGER GENERATED BY DEFAULT AS IDENTITY,
    description VARCHAR(255),
    is_done     BOOLEAN,
    target_date TIMESTAMP,
    PRIMARY KEY (id)
)
```

16:09 / 57:15

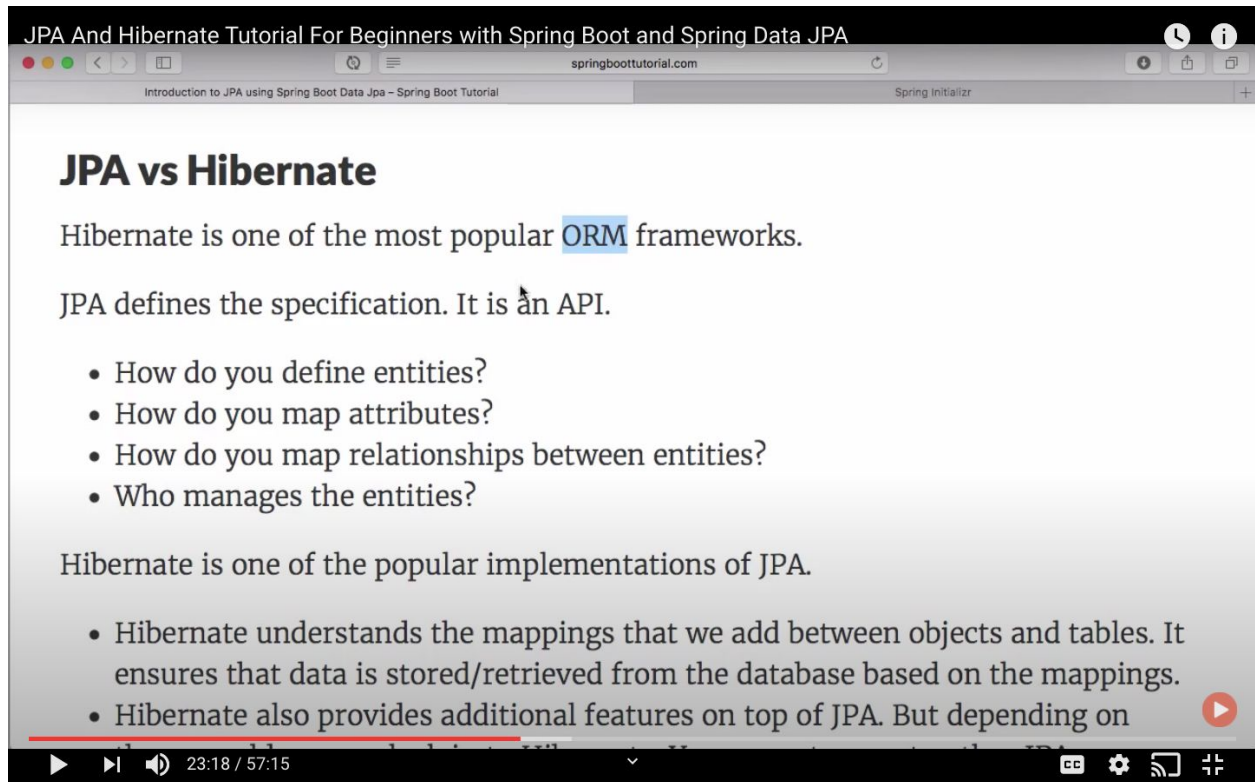
Some times multiple classes are mapped to a single table and vice-versa. In these situations, we define an inheritance strategy. In this example, we use a strategy of `InheritanceType.SINGLE_TABLE`.

Objects

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "EMPLOYEE_TYPE")
public class Employee {
    //Other Employee Attributes
}

public class FullTimeEmployee extends Employee {
    protected Integer salary;
}

public class PartTimeEmployee extends Employee {
    protected Float hourlyWage;
}
```



- Step 4 : Creating a JPA Project using Spring Initializr
- Step 5 : Defining a JPA Entity - User

Using Entity Manager

@Entity: Map public class User to Table User

@Id : map class members as a primary key

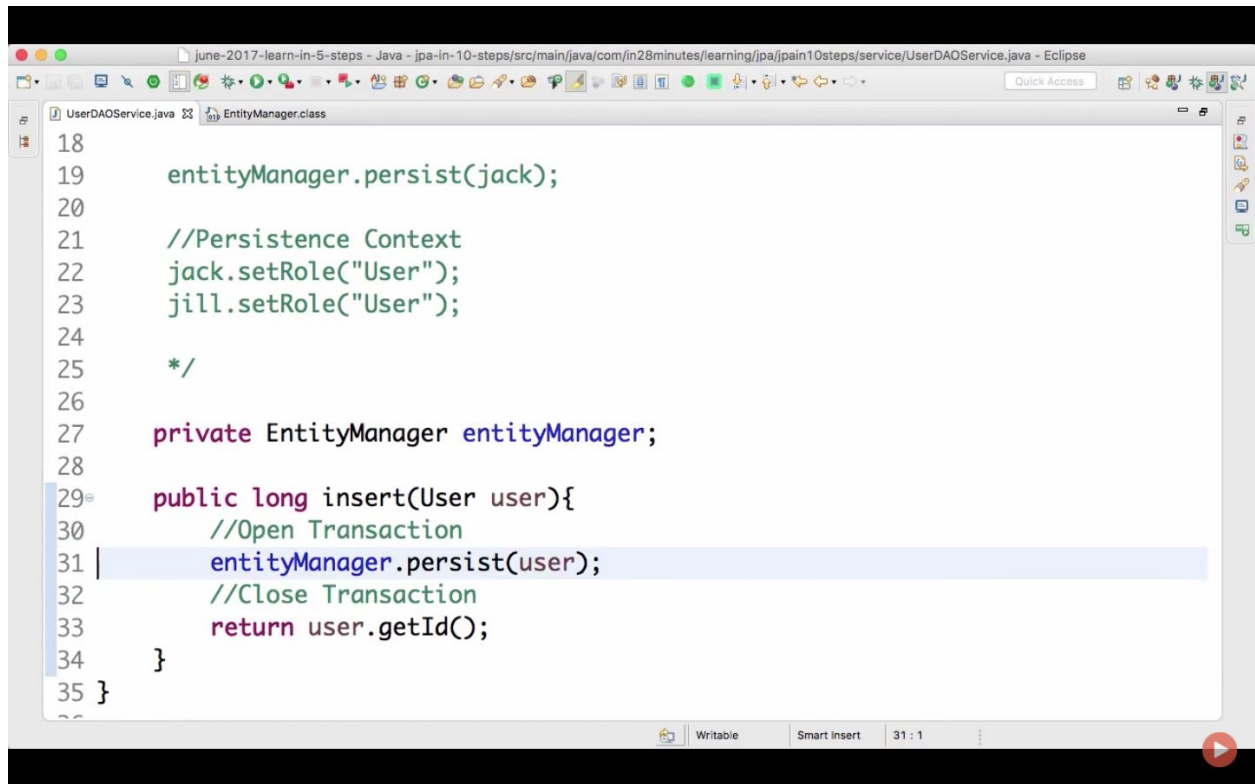
@GeneratedValue : we don't assign value this attribute, automatically generated

@Repository : class that helps store things to database

class UserDAOService: Data Access Object, helps us access data from database

Whenever we modify database using any method

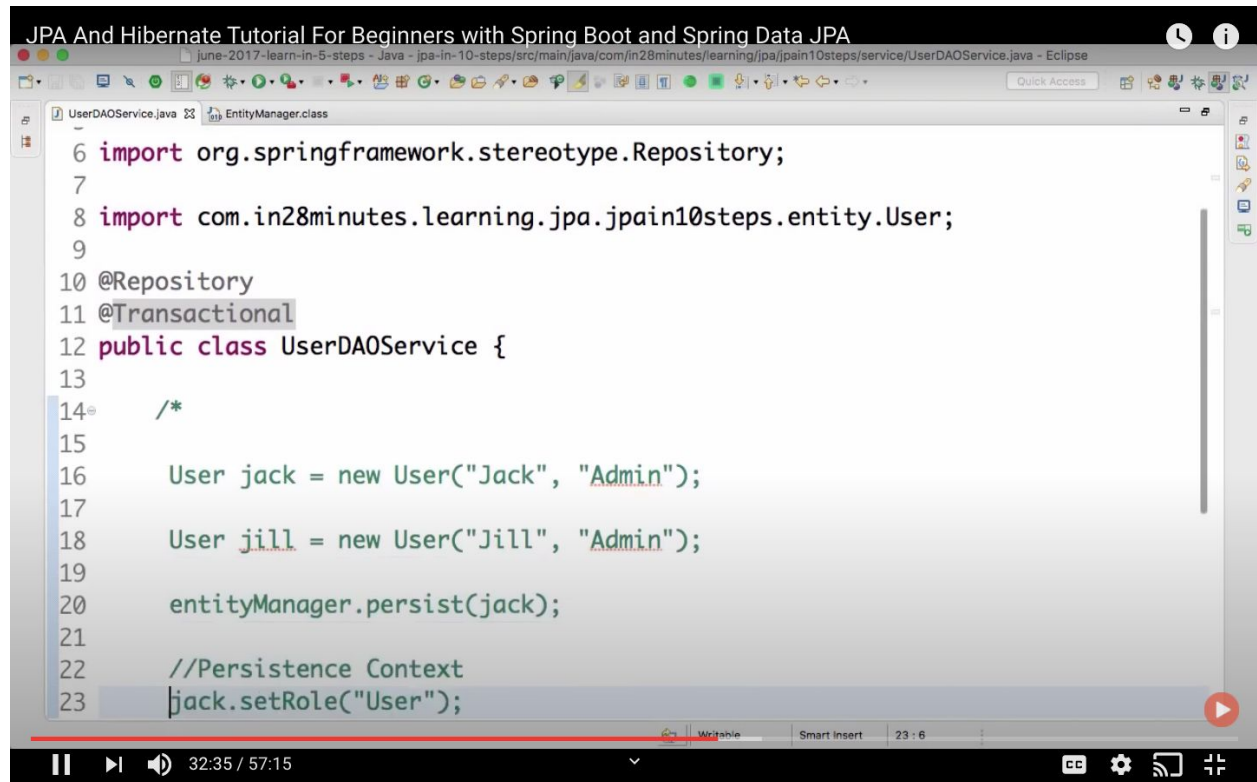
There should be a transaction opened before modify and closed after modify



```
18
19     entityManager.persist(jack);
20
21     //Persistence Context
22     jack.setRole("User");
23     jill.setRole("User");
24
25     */
26
27     private EntityManager entityManager;
28
29     public long insert(User user){
30         //Open Transaction
31         entityManager.persist(user);
32         //Close Transaction
33         return user.getId();
34     }
35 }
```

We would have to define this for every modification method

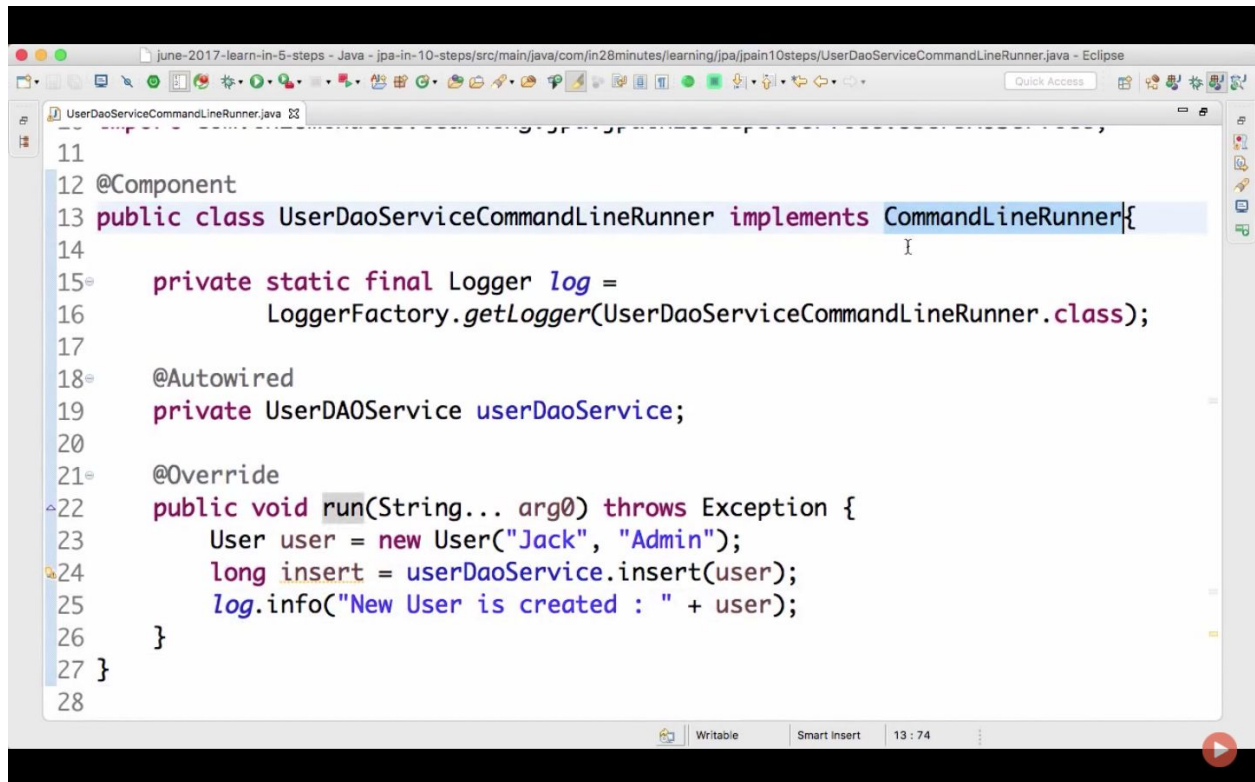
To overcome this use `@Transactional`



```
6 import org.springframework.stereotype.Repository;
7
8 import com.in28minutes.learning.jpa.jpain10steps.entity.User;
9
10 @Repository
11 @Transactional
12 public class UserDAOService {
13
14     /*
15
16     User jack = new User("Jack", "Admin");
17
18     User jill = new User("Jill", "Admin");
19
20     entityManager.persist(jack);
21
22     //Persistence Context
23     jack.setRole("User");
```

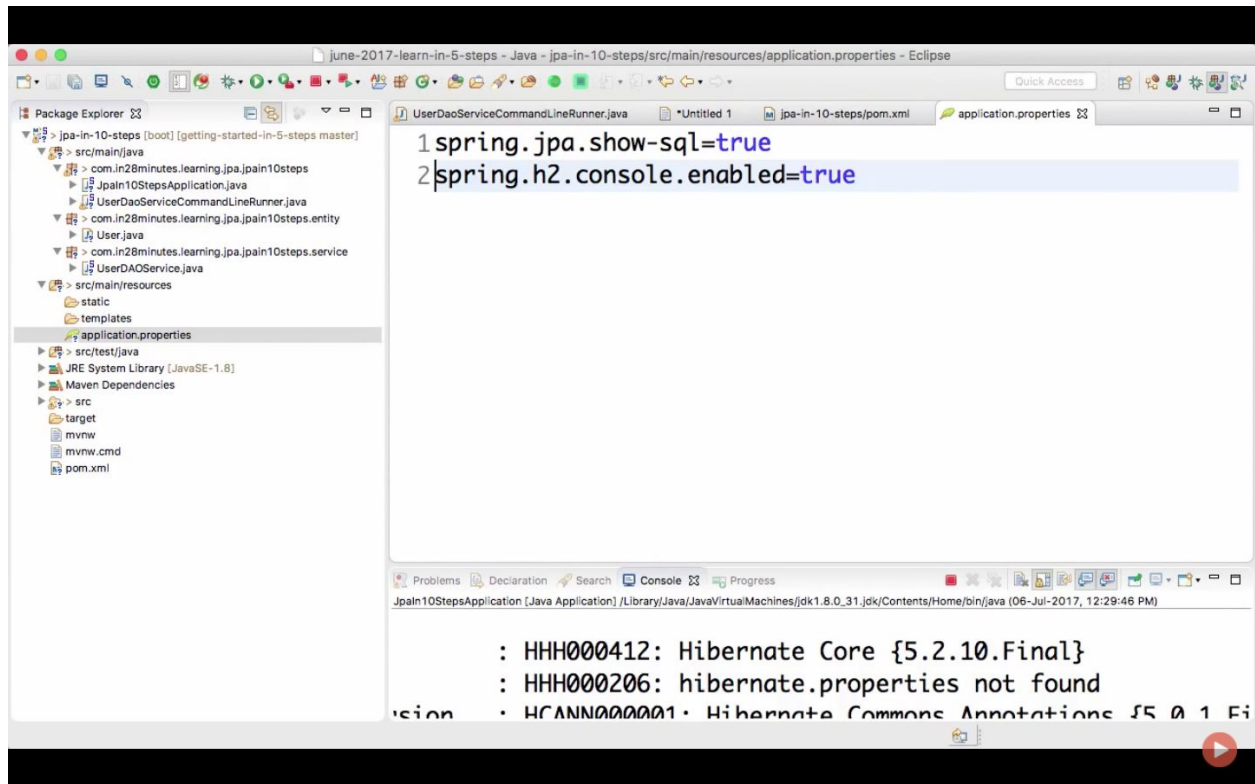
@PersistenceContext == @Autowired ??

- Step 6 : Defining a Service to manage the Entity - UserService and EntityManager
- Step 7 : Using a Command Line Runner to save the User to database.



```
11
12 @Component
13 public class UserDaoServiceCommandLineRunner implements CommandLineRunner{
14
15     private static final Logger log =
16         LoggerFactory.getLogger(UserDaoServiceCommandLineRunner.class);
17
18     @Autowired
19     private UserDaoService userDaoService;
20
21     @Override
22     public void run(String... args) throws Exception {
23         User user = new User("Jack", "Admin");
24         long insert = userDaoService.insert(user);
25         log.info("New User is created : " + user);
26     }
27 }
28
```

- Step 8 : Magic of Spring Boot and In Memory Database H2



Questions

- Where is the database created?
 - In Memory – Using H2
- What schema is used to create the tables?
 - Created based on the entities defined
- Where are the tables created?
 - Created based on the entities defined
 - In Memory – Using H2
- Can I see the data in the database?
 - <http://localhost:8080/h2-console>
 - Use db url `jdbc:h2:mem:testdb`
- Where is Hibernate coming in from?
 - Through Spring Data JPA Starter
- How is a datasource created?
 - Through Spring Boot Auto Configuration

<http://localhost:8080/h2-console>

Magic of Spring Boot and in Memory Database

- Zero project setup or infrastructure

- Step 9 : Introduction to Spring Data JPA

When creating more than one entities some common methods such as delete and update are repeated again and again for all different entities

For 15 entities ,there would be 15 DAOservices have almost same logic

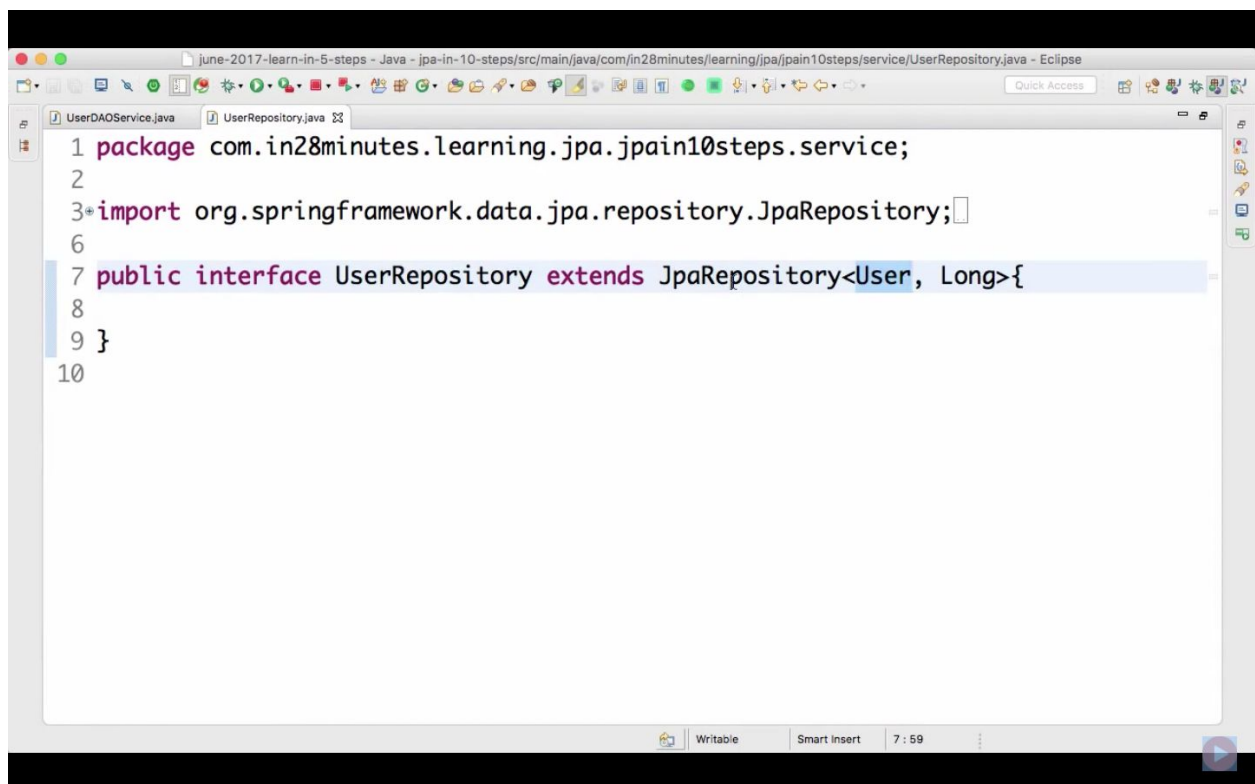
Spring Data solves this problem by letting u create a repository interface and it will take care of implementation by talking with entity manager

- Step 10 : More JPA Repository : findByld and findAll

Basically we create this interface

Pass in

<entity that you want to be managed,Primary key of that entity>

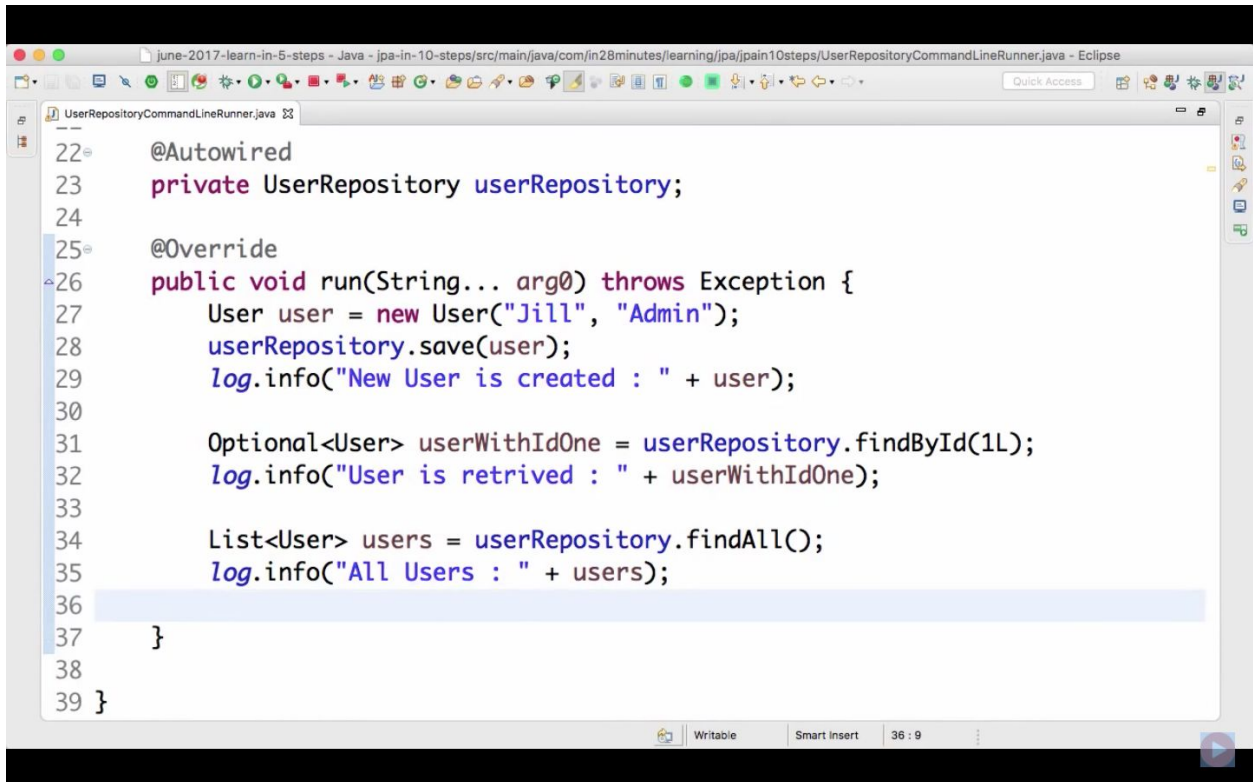


```
1 package com.in28minutes.learning.jpa.jpain10steps.service;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7 public interface UserRepository extends JpaRepository<User, Long>{
8
9 }
10
```

Spring Data provides methods like save which is used to insert

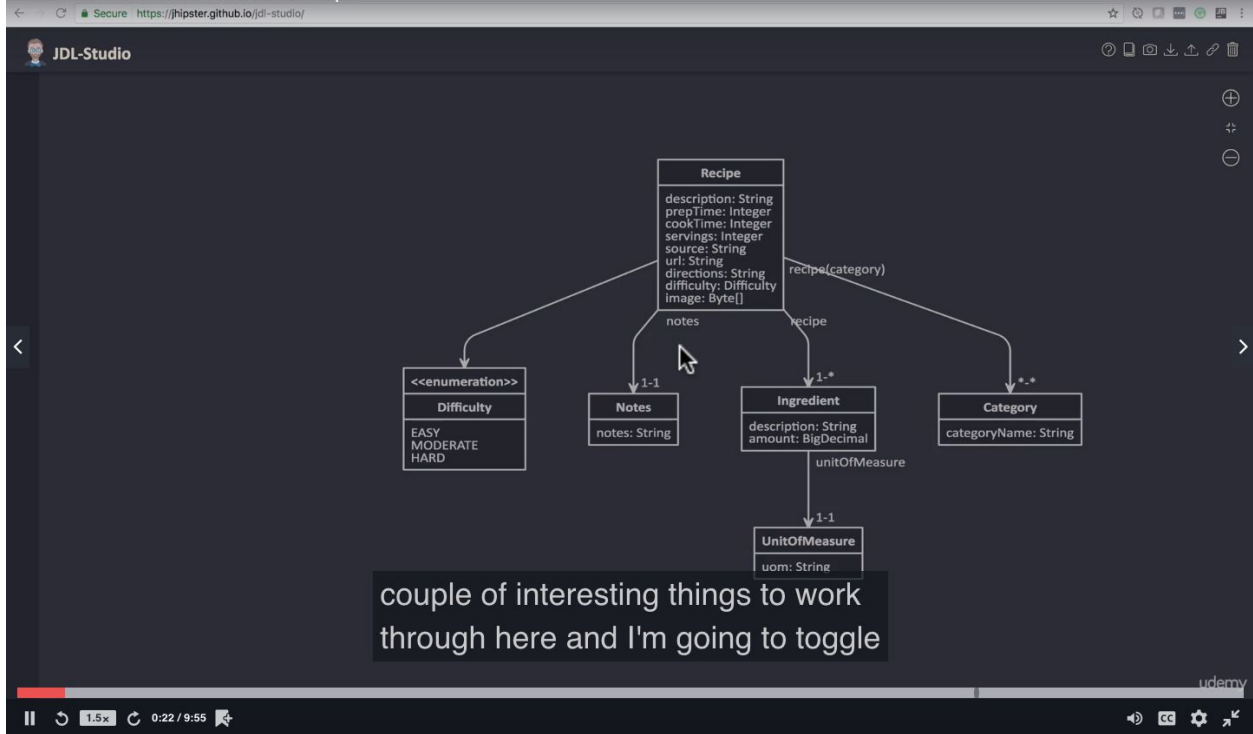
findAll : works Select *

And many predefined methods



```
22= @Autowired
23 private UserRepository userRepository;
24
25= @Override
26 public void run(String... arg0) throws Exception {
27     User user = new User("Jill", "Admin");
28     userRepository.save(user);
29     log.info("New User is created : " + user);
30
31     Optional<User> userWithIdOne = userRepository.findById(1L);
32     log.info("User is retrived : " + userWithIdOne);
33
34     List<User> users = userRepository.findAll();
35     log.info("All Users : " + users);
36
37 }
38
39 }
```

132. One To One JPA Relationships



Recipe.java - spring5-recipe-app - [~/src/springframework.guru/spring5/spring5-recipe-app]

```

package guru.springframework.domain;
import javax.persistence.*;
import java.util.Set;

/**
 * Created by jt on 8/13/17.
 */
@Entity
public class Recipe {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String description;
    private Integer prepTime;
    private Integer cookTime;
    private Integer servings;
    private String source;
    private String url;
    private String directions;
    //todo add
    //private Difficulty difficulty;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "recipe")
    private Set<Ingredient> ingredients;

    @Lob
    private Byte[] image;

    @OneToOne(cascade = CascadeType.ALL)
    private Notes notes;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getDescription() {

```

Version Control: Local Changes Log Console

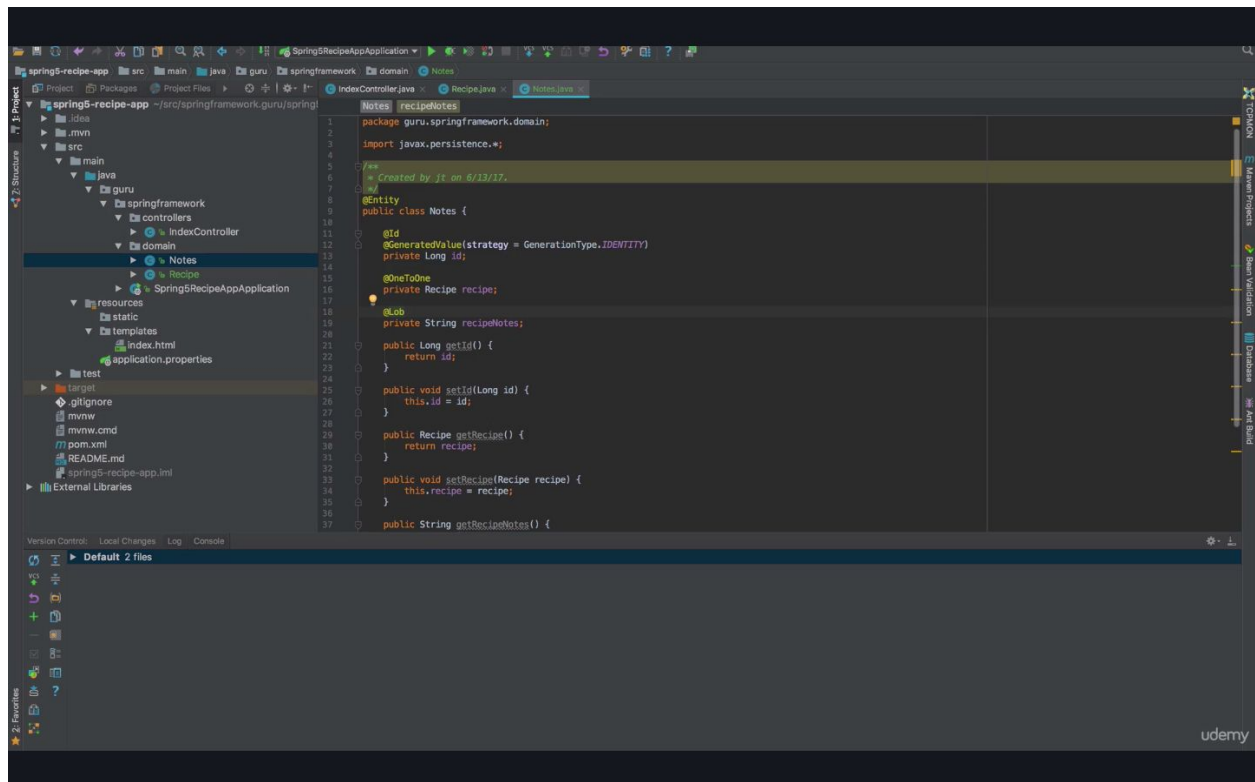
Default 2 files

udemy

By using cascade all we make recipe the owner entity of Notes entity

Note in image below we don't use cascade with recipe property as on deleting Notes class Recipe class should not be deleted but if we delete Recipe, Notes will also be deleted

@Lob= Large objects



133. One To Many JPA Relationships

```
import javax.persistence.*;
import java.math.BigDecimal;

/**
 * Created by it on 6/13/17.
 */
@Entity
public class Ingredient {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String description;
    private BigDecimal amount;
    //private UnitOfMeasure uom;

    @ManyToOne
    private Recipe recipe;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getDescription() {
        return description;
    }

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

    public BigDecimal getAmount() {
        return amount;
    }

    public void setAmount(BigDecimal amount) {
        this.amount = amount;
    }
}
```

The screenshot shows an IDE window titled "133. One To Many JPA Relationships". The main editor displays the `Ingredient.java` file. The code defines an `@Entity` class `Ingredient` with the following attributes and methods:

- `@Id` and `@GeneratedValue(strategy = GenerationType.IDENTITY)` on the `id` field.
- `private Long id;`
- `private String description;`
- `private BigDecimal amount;`
- `//private UnitOfMeasure uom;` (commented out)
- `@ManyToOne` on the `recipe` field.
- `private Recipe recipe;`
- Methods: `getId()`, `setId(Long id)`, `getDescription()`, `setDescription(String description)`, `getAmount()`, and `setAmount(BigDecimal amount)`.

The IDE interface includes a project explorer on the left showing the project structure, a code editor in the center, and a console at the bottom. The console shows "Default 2 files". The IDE is running on a Windows operating system, as indicated by the taskbar at the bottom.