**Spring Boot - REST CRUD Real-Time Project Overview and Database Set Up**

**API Requirements**:

Create a REST API for the Employee Directory. Rest client should able to

1. Get a list of employees
2. Get a single employee by ID
3. Add a new employee
4. Update an existing Employee
5. Delete an employee.

**REST API Endpoints**:

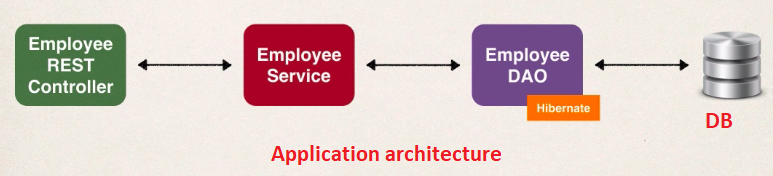
|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **Endpoints** | **CRUD Action** |
| POST | /api/employees | Create a new employee |
| GET | /api/employees | Read a list of employees |
| GET | /api/employees/{employeeId} | Read a single employee |
| PUT | /api/employees | Update an existing employee |
| DELETE | /api/employees/{employeeId} | Delete an existing employee |

**Development process (Step-By-Step)**:

1. Set up Database Dev Environment
2. Create Spring Boot project using Spring Initializer
3. Get list of employees
4. Get single employee by ID
5. Add a new employee
6. Update an existing employee
7. Delete an existing employee

**Application architecture**:

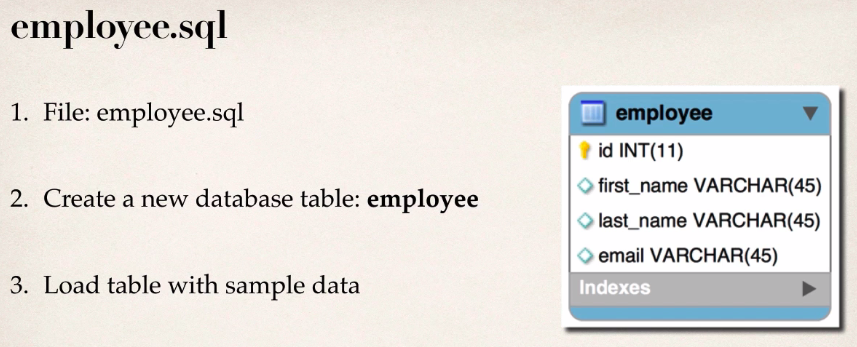
We design our application by using Java configuration. We don’t use XML configuration.



**1) Setup Database Table**:

We create "employee.sql" table in database and insert data.

1. File: employee.sql
2. Create a new database table: employee
3. Load table with sample data



**Script "employee.sql"**:

CREATE DATABASE IF NOT EXISTS `employee\_directory`;

USE `employee\_directory`;

-- Table structure for table `employee`

DROP TABLE IF EXISTS `employee`;

CREATE TABLE `employee` (

  `id` int(11) NOT NULL AUTO\_INCREMENT,

  `first\_name` varchar(45) DEFAULT NULL,

  `last\_name` varchar(45) DEFAULT NULL,

  `email` varchar(45) DEFAULT NULL,

  PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=latin1;

-- Data for table `employee`

INSERT INTO `employee` VALUES

  (1,'Leslie','Andrews','leslie@luv2code.com'),

  (2,'Emma','Baumgarten','emma@luv2code.com'),

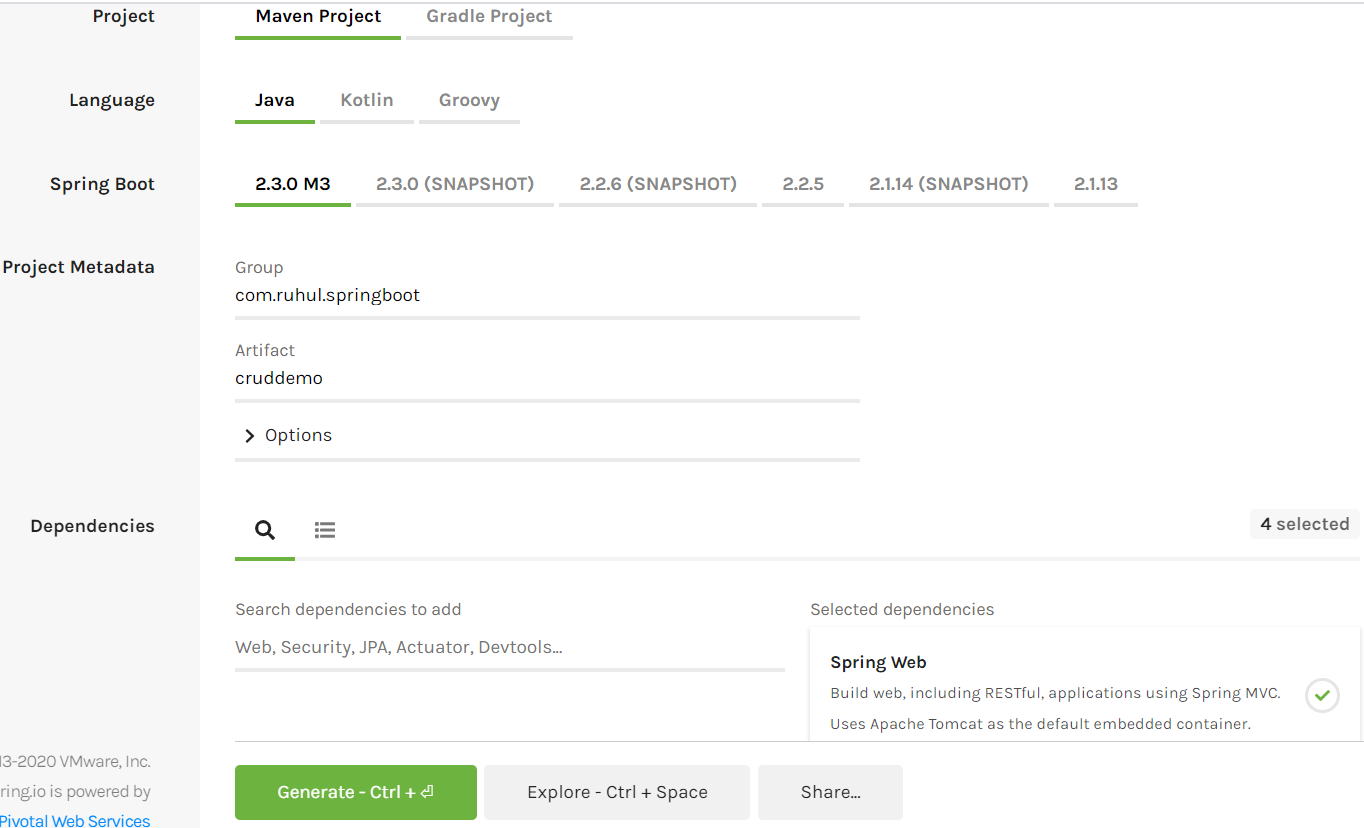
  (3,'Avani','Gupta','avani@luv2code.com'),

  (4,'Yuri','Petrov','yuri@luv2code.com'),

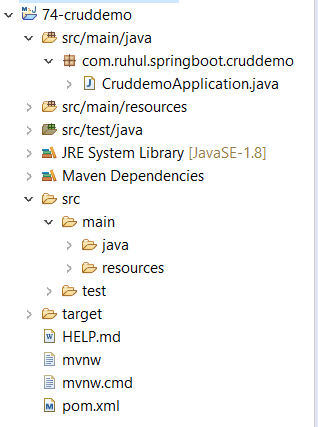
  (5,'Juan','Vega','juan@luv2code.com');

**2) Create Spring Boot project using Spring Initializer**:

1. Go to <https://start.spring.io/>
2. Select Maven project
3. Select language "Java"
4. Select Spring Boot version
5. Select group Id and artifact Id
6. Add dependency
7. Click on Generate button



Then the project will be downloaded in our machine. Now import the project in eclipse.



Now the remaining steps are

1. Get list of employees
2. Get single employee by ID
3. Add a new employee
4. Update an existing employee
5. Delete an existing employee

**The "pom.xml" file**:

<?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 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.Boot</groupId>

<artifactId>spring-Boot-starter-parent</artifactId>

<version>2.3.0.M3</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>com.ruhul.springBoot</groupId>

<artifactId>cruddemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>cruddemo</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!-- For Hibernate ORM support -->

<dependency>

<groupId>org.springframework.Boot</groupId>

<artifactId>spring-Boot-starter-data-jpa</artifactId>

</dependency>

<!-- For MVC and REST support -->

<dependency>

<groupId>org.springframework.Boot</groupId>

<artifactId>spring-Boot-starter-web</artifactId>

</dependency>

<!-- For Automatic reloading support -->

<dependency>

<groupId>org.springframework.Boot</groupId>

<artifactId>spring-Boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<!-- For MySQL JDBC driver support -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<!-- For Unit Testing support -->

<dependency>

<groupId>org.springframework.Boot</groupId>

<artifactId>spring-Boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<!-- Spring Boot Maven plugin (Packaging and running) -->

<plugins>

<plugin>

<groupId>org.springframework.Boot</groupId>

<artifactId>spring-Boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

</pluginRepository>

</pluginRepositories>

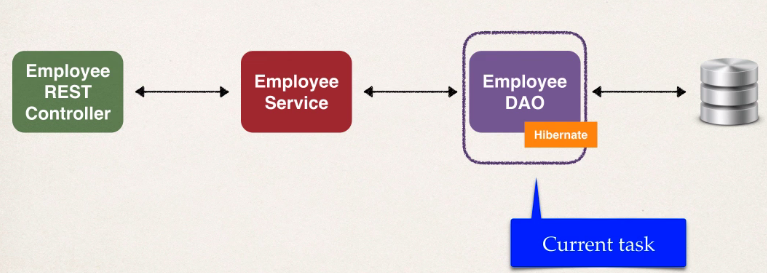
</project>

Now we have to doing the following works for complete the above steps.

**Build a DAO layer**:

Import downloaded project in eclipse.

**Build a DAO layer**:



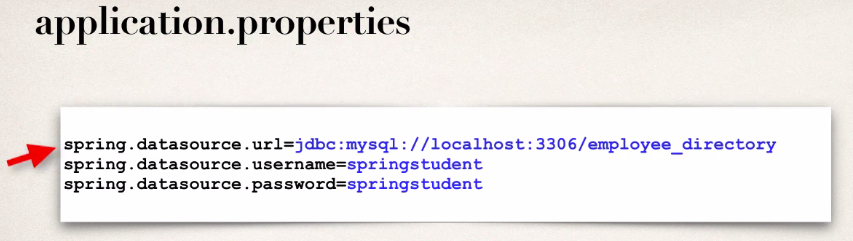
In traditional Spring we write the code manually for database connection. We use XML configuration for this. This is very error-prone and easy to make a simple mistake. Spring Boot make solve this and make the connection very easy.

Spring Boot automatically configure our data source for us. This is based on entries from Maven pom file. Spring Boot use

JDBC driver: mysql-connector-Java

Spring Data (ORM): spring-Boot-starter-data-jpa

DB connection from: application.properties



No need to give JDBC class name. Spring Boot will automatically detect it based on URL.

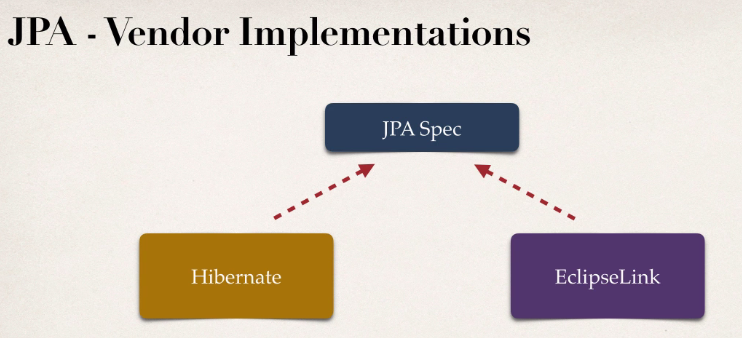
**Additional Data Source Properties**:

[https://docs.spring.io/spring-Boot/docs/current/reference/html/appendix-application-properties.html#core-properties](https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html#core-properties)

**Auto data source configuration**:

1. Based on configs, Spring Boot will automatically create the beans
   1. DataSource and EntityManager.
2. We can then inject those into our app. For example, DAO.
3. EntityManager is for Java Persistence API (JPA).
   1. Standard API for Object-Relation-Mapping (ORM)
4. JPA is only a specification. It defines a set of interfaces.

**JPA Vendor Implementation**:



**Auto Data Source Configuration**:

In Spring Boot, Hibernate is default implementation of JPA.

EntityManager is similar to Hibernate SessionFactory.

EntityManager can serve as a wrapper for Hibernate Session object.

We can inject the EntityManager into our DAO.

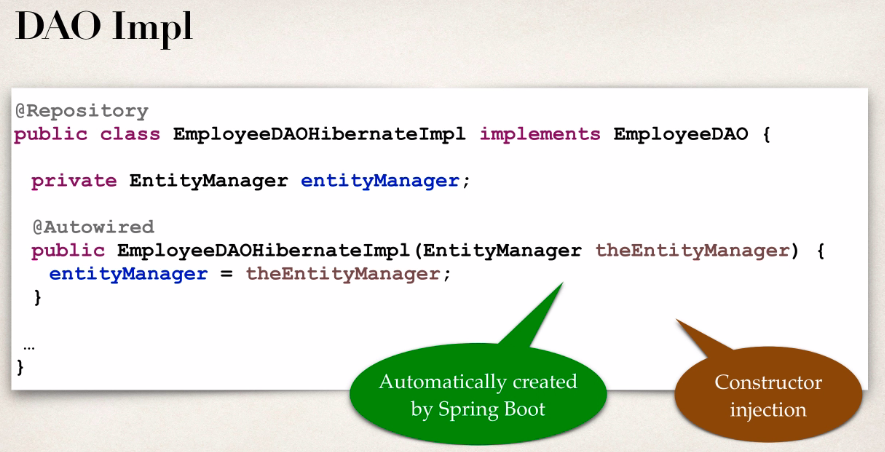
**Various DAO Techniques**:

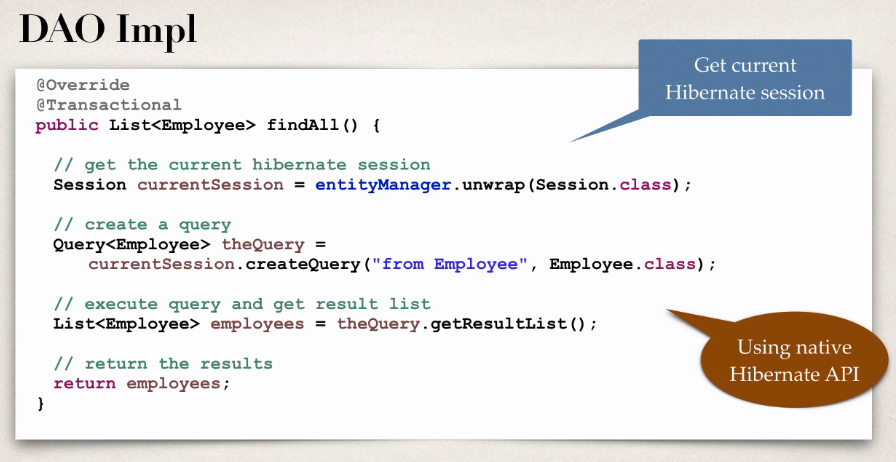
1. Use EntityManager but leverage native Hibernate API.
2. Use EntityManager and standard JPA API.
3. Use Spring Data JPA.

**DAO Interface**:

public interface

**DAO implementation**:





**Development Process**:

1. Update BD configs in application.properties
2. Create Employee entity.
3. Create DAO interface
4. Create DAO implementation
5. Create REST controller to use DAO.

1) **Update BD configs in application.properties**:

#JDBC properties

spring.datasource.url=jdbc:mysql://localhost:3306/employee\_directory

spring.datasource.username = root

spring.datasource.password = 1234

**Employee.java**:

**package** com.ruhul.springboot.cruddemo.entity;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "employee")

**public** **class** Employee {

// define fields

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** **int** id;

@Column(name = "first\_name")

**private** String firstName;

@Column(name = "last\_name")

**private** String lastName;

@Column(name = "email")

**private** String email;

// define constructors

**public** Employee() {

}

**public** Employee(String firstName, String lastName, String email) {

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.email = email;

}

// define getter/setter

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

// define toString

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", email=" + email + "]";

}

}

**EmployeeRestController**:

**package** com.ruhul.springboot.cruddemo.controller;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.DeleteMapping;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.PutMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.ruhul.springboot.cruddemo.entity.Employee;

**import** com.ruhul.springboot.cruddemo.service.EmployeeService;

@RestController

@RequestMapping("/api")

**public** **class** EmployeeRestController {

**private** EmployeeService employeeService;

// quick and dirty: inject employee service

@Autowired

**public** EmployeeRestController(EmployeeService theEmployeeService) {

employeeService = theEmployeeService;

}

// expose "/employees" and return list of employees

@GetMapping("/employees")

**public** List<Employee> findAll() {

**return** employeeService.findAll();

}

// add mapping for GET /employee/{employeeId}

@GetMapping("/employees/{employeeId}")

**public** Employee getEmployee(@PathVariable **int** employeeId) {

Employee theEmployee = employeeService.findById(employeeId);

**if** (theEmployee == **null**) {

**throw** **new** RuntimeException("Employee id not found - " + employeeId);

}

**return** theEmployee;

}

// add mapping for POST /employees - add new employee

@PostMapping("/employees")

**public** Employee addEmployee(@RequestBody Employee theEmployee) {

// also just in case they pass an id in JSON ... set id to 0

// this is to force a save of new item ... instead of update

theEmployee.setId(0);

employeeService.save(theEmployee);

**return** theEmployee;

}

// add mapping for PUT /employees - update existing employee

@PutMapping("/employees")

**public** Employee updatEmployee(@RequestBody Employee theEmployee) {

employeeService.save(theEmployee);

**return** theEmployee;

}

// add mapping for DELETE /employees/{employeeId} - delete employee

@DeleteMapping("/employees/{employeeId}")

**public** String deleteEmployee(@PathVariable **int** employeeId) {

Employee theEmployee = employeeService.findById(employeeId);

**if** (theEmployee == **null**) {

**throw** **new** RuntimeException("Employee id not found - " + employeeId);

}

**try** {

employeeService.deleteById(employeeId);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** "Deleted employee id - " + employeeId;

}

}

**EmployeeService.java**:

**package** com.ruhul.springboot.cruddemo.service;

**import** java.util.List;

**import** com.ruhul.springboot.cruddemo.entity.Employee;

**public** **interface** EmployeeService {

**public** List<Employee> findAll();

**public** Employee findById(**int** theId);

**public** **void** save(Employee theEmployee);

**public** **void** deleteById(**int** theId);

}

**EmployeeServiceImpl.java**:

**package** com.ruhul.springboot.cruddemo.service;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** org.springframework.transaction.annotation.Transactional;

**import** com.ruhul.springboot.cruddemo.dao.EmployeeDAO;

**import** com.ruhul.springboot.cruddemo.entity.Employee;

@Service

**public** **class** EmployeeServiceImpl **implements** EmployeeService {

**private** EmployeeDAO employeeDAO;

@Autowired

**public** EmployeeServiceImpl(EmployeeDAO theEmployeeDAO) {

employeeDAO = theEmployeeDAO;

}

@Override

@Transactional

**public** List<Employee> findAll() {

**return** employeeDAO.findAll();

}

@Override

@Transactional

**public** Employee findById(**int** theId) {

**return** employeeDAO.findById(theId);

}

@Override

@Transactional

**public** **void** save(Employee theEmployee) {

employeeDAO.save(theEmployee);

}

@Override

@Transactional

**public** **void** deleteById(**int** theId) {

employeeDAO.deleteById(theId);

}

}

**EmployeeDao.java**:

**package** com.ruhul.springboot.cruddemo.dao;

**import** java.util.List;

**import** com.ruhul.springboot.cruddemo.entity.Employee;

**public** **interface** EmployeeDAO {

**public** List<Employee> findAll();

**public** Employee findById(**int** theId);

**public** **void** save(Employee theEmployee);

**public** **void** deleteById(**int** theId);

}

**EmployeeDAOHibernateImpl.java**:

**package** com.ruhul.springboot.cruddemo.dao;

**import** java.util.List;

**import** javax.persistence.EntityManager;

**import** org.hibernate.Session;

**import** org.hibernate.query.Query;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Repository;

**import** com.ruhul.springboot.cruddemo.entity.Employee;

@Repository

**public** **class** EmployeeDAOHibernateImpl **implements** EmployeeDAO {

// define field for entitymanager

**private** EntityManager entityManager;

// set up constructor injection

@Autowired

**public** EmployeeDAOHibernateImpl(EntityManager theEntityManager) {

entityManager = theEntityManager;

}

@Override

**public** List<Employee> findAll() {

// get the current hibernate session

Session currentSession = entityManager.unwrap(Session.**class**);

// create a query

Query<Employee> theQuery = currentSession.createQuery("from Employee", Employee.**class**);

// execute query and get result list

List<Employee> employees = theQuery.getResultList();

// return the result

**return** employees;

}

@Override

**public** Employee findById(**int** theId) {

// get the current hibernate session

Session currentSession = entityManager.unwrap(Session.**class**);

// get the employee

Employee theEmployee = currentSession.get(Employee.**class**, theId);

// return the employee

**return** theEmployee;

}

@Override

**public** **void** save(Employee theEmployee) {

// get the current hibernate session

Session currentSession = entityManager.unwrap(Session.**class**);

// save employee

currentSession.saveOrUpdate(theEmployee);

}

@Override

**public** **void** deleteById(**int** theId) {

// get the current hibernate session

Session currentSession = entityManager.unwrap(Session.**class**);

// delete object with primary key

Query theQuery = currentSession.createQuery("DELETE FROM Employee WHERE id =:employeeId");

theQuery.setParameter("employeeId", theId);

theQuery.executeUpdate();

}

}

DAO