JPA

Dependencies

```
<!-- Used to keep the application running -->
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-web</artifactid>
</dependency>
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-data-jpa</artifactid>
</dependency>
<dependency>
  <groupid>com.h2database</groupid>
  <artifactid>h2</artifactid>
</dependency>
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-devtools</artifactid>
</dependency>
```

application.properties

```
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

#spring.datasource.url=jdbc:mysql://localhost/test
#spring.datasource.driverClassName=com.mysql.jdbc.Driver
```

CrudRepository

- Creates standard methods like save / delete / find etcetera
- Some methods are present in the interface
- Other methods can be added easily

JPAApplication

```
@SpringBootApplication
public class JPAApplication {
}
```

Book Entity

```
@Entity
public class Book {
  @Id
  @GeneratedValue
  private Long id;

  private String name;

  private String author;

  public Book(String name, String author) {
    super();
    this.name = name;
    this.author = author;
  }
```

Book Repository

```
public interface BookRepository extends CrudRepository<Book, Long> {
   List<Book> findByName(String name);
   List<Book> findByAuthor(String author);
}
```

@Query

```
@Query("SELECT b FROM Book b WHERE b.name=:name OR b.author=:author")
List<Book> findBooksByNameOrAuthor(
  @Param("name") String name, @Param("author") String author);
```

Streaming

```
@Repository
public interface StreamingBookRepository
    extends CrudRepository<Book, Long> {
  @Query("select b from Book b")
  Stream<Book> findBooks();
}
```

Test setup

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class BookRepositoryTest {

    @Autowired
    private BookRepository bookRepository;

    private Book springBootResultingBook;

    @Before
    public void init() {
      Book springBook = new Book("Pro Spring", "Bob Harrop");
    Book springBootBook =
      new Book("Spring Boot in Action", "Craig Walls");
    bookRepository.save(springBook);
    springBootResultingBook = bookRepository.save(springBootBook);
    }
```

Tests

```
@Test
public void testFindOne() {
  Book book =
    bookRepository.findOne(springBootResultingBook.getId());
  assertEquals("Spring Boot in Action", book.getName());
  assertEquals("Craig Walls", book.getAuthor());
}

@Test
public void testFindByAuthor() {
  List<Book> findByAuthorBookList =
    bookRepository.findByAuthor("Craig Walls");
  Book book = findByAuthorBookList.get(0);
  assertEquals("Spring Boot in Action", book.getName());
  assertEquals("Craig Walls", book.getAuthor());
}
```

SQL Queries

- Number of SQL queries might be different from the number of JPA queries
- Big source of performance issues
- Always verify the (number of) generated SQL queries!

Book

```
@Entity
public class Book {

  @Id
  @GeneratedValue
  private Long id;

  private String name;

  // Constructors
  // Getters and setters
}
```

Course

```
@Entity
public class Course {
  @Id
  @GeneratedValue
  private Long id;

  private String name;

  @OneToMany(fetch = FetchType.EAGER)
  private List<Book> books;

  // Constructors
  // Getters and setters
}
```

Database structure

- http://localhost:8080/h2-console
- Three tables
  - BOOK
  - COURSE
  - COURSE_BOOKS

Enable SQL query logging

Add the following lines to application.properties

```
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.type=trace
```

Creating entities

```
List<Book> springBooks = new ArrayList<Book>();
Book book1 = new Book("Spring in Action");
springBooks.add(book1);
Book book2 = new Book("Spring Boot in Action");
springBooks.add(book2);

List<Book> javaBooks = new ArrayList<Book>();
Book book3 = new Book("Core Java");
javaBooks.add(book3);
Book book4 = new Book("Head First Java");
javaBooks.add(book4);
```

Save entities

```
bookRepository.save(book1);
bookRepository.save(book2);
bookRepository.save(book3);
bookRepository.save(book4);

courseRepository.save(new Course("Spring", springBooks));
courseRepository.save(new Course("Java", javaBooks));
```

CrudRepository findAll()

```
Iterable<Course> courseIterable = courseRepository.findAll();
```

Continuation former slide: Number of SQL queries: 3

```
Hibernate:
    select course0_.id as id1_1_, course0_.name as name2_1_
    from  course course0_
Hibernate:
    select books0_.course_id as course_i1_2_0_,
           book1_.id as id1_0_1_,
    from course_books books0_
    inner join book book1_
    on books0_.books_id=book1_.id
    where books0_.course_id=?
Hibernate:
    select
        books0_.course_id as course_i1_2_0_, book1_.id as id1_0_1_,
    from course_books books0_
    inner join book book1_
    on books0_.books_id=book1_.id
    where  books0_.course_id=?
```

JPQL

```
List<Course> courses = (List<Course>)
  entityManager.createQuery("SELECT c FROM Course c")
               .getResultList();
```

* Resulting courses:

```
Number of courses: 2
Spring : Spring in Action | Spring Boot in Action
Java   : Core Java        | Head First Java
```

- Number of SQL queries: 3

```
Hibernate:
  select course0_.id as id1_1_,
         course0_.name as name2_1_
  from course course0_
Hibernate:
  select books0_.course_id as course_i1_2
         book1_.id as id1_0_1_,
  from  course_books books0_
  inner join book book1_
        on books0_.books_id=book1_.id
  where books0_.course_id=?
Hibernate:
  select books0_.course_id as course_i1_2
         book1_.id as id1_0_1_,
  from course_books books0_
  inner join book book1_
        on books0_.books_id=book1_.id
  where books0_.course_id=?
```

JPQL join fetch

```
List<Course> courses = (List<Course>)
  entityManager.createQuery(
    "SELECT c FROM Course c join fetch c.books").getResultList();
```

- Resulting courses:

```
Spring : Spring in Action | Spring Boot in Action
Spring : Spring in Action | Spring Boot in Action
Java : Core Java | Head First Java
Java : Core Java | Head First Java
```

Number of SQL queries: 1

```
Hibernate:
    select
        course0_.id as id1_1_0_,
        book2_.id as id1_0_1_,
        course0_.name as name2_1_0_,
        book2_.name as name2_0_1_,
        books1_.course_id as course_i1_2_0__,
        books1_.books_id as books_id2_2_0__
    from
        course course0_
    inner join
        course_books books1_
            on course0_.id=books1_.course_id
    inner join
        book book2_
            on books1_.books_id=book2_.id
```

Testing DAO's

- Unit testing doesn’t make sense
  - Queries can only be tested with a real database
- Run automated tests within a Spring container
  - Use a real database or an embedded one

Dependencies

```
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-test</artifactid>
  <scope>test</scope>
</dependency>
```

Testing DAO's

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class BookRepositoryTest {
    @Autowired
    private TestEntityManager entityManager;
    @Autowired
    private BookRepository bookRepository;

    @Test
    public void testFindByLastName() {
        Book book = new Book("Spring in action");
        entityManager.persist(book);

        List<Book> books = bookRepository.findByName(book.getName());
        assertEquals(1, books.size());
        books = bookRepository.findByName("Book does not exist");
        assertEquals(0, books.size());
```

Testing DAO's

- Transactions are rolled back after each test by default
  - no need to re-insert test data for each test
- It's possible to test JPA code using JDBC queries

Turn off Automatic Rollback

- Do not rollback one unit test or a whole unit test class

```
@Transactional(propagation = Propagation.NOT_SUPPORTED)
```