CHAPTER 3

# SPRING DATA JPA

# Chapter 3: SPRING DATA JPA

## OUTLINE

1. Getting started with Spring Data JPA
2. Create a New Spring Boot Project

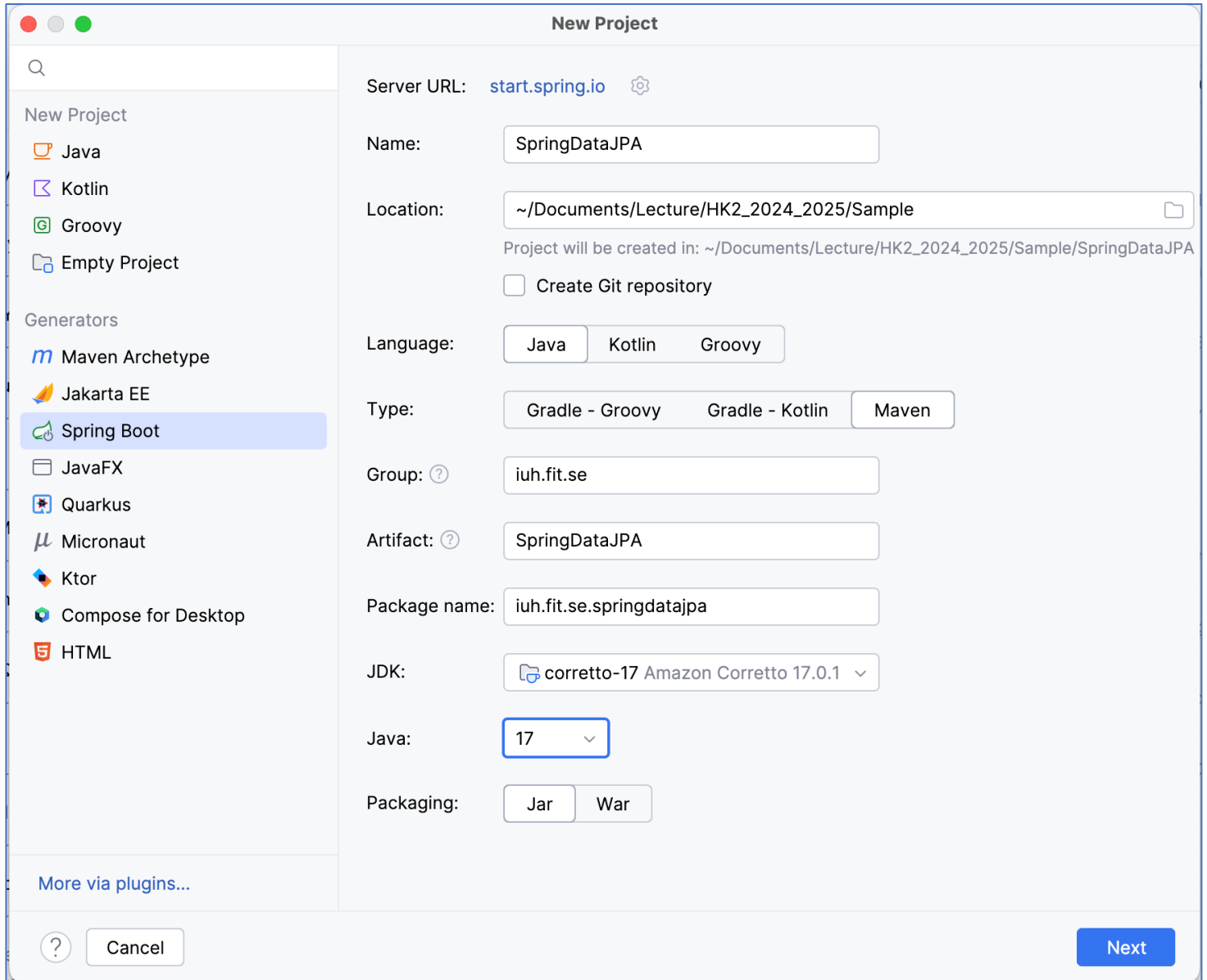
# Getting started with Spring Data JPA

1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary

- Spring Data JPA is a powerful framework that allows users to easily interact with their database while minimizing boilerplate code.
- In this tutorial, we're going to look at how to use Spring Data JPA to insert into and query data from a database.
- We'll create a simple Spring Boot application using IntelliJ IDEA Ultimate to take advantage of its Spring feature support.

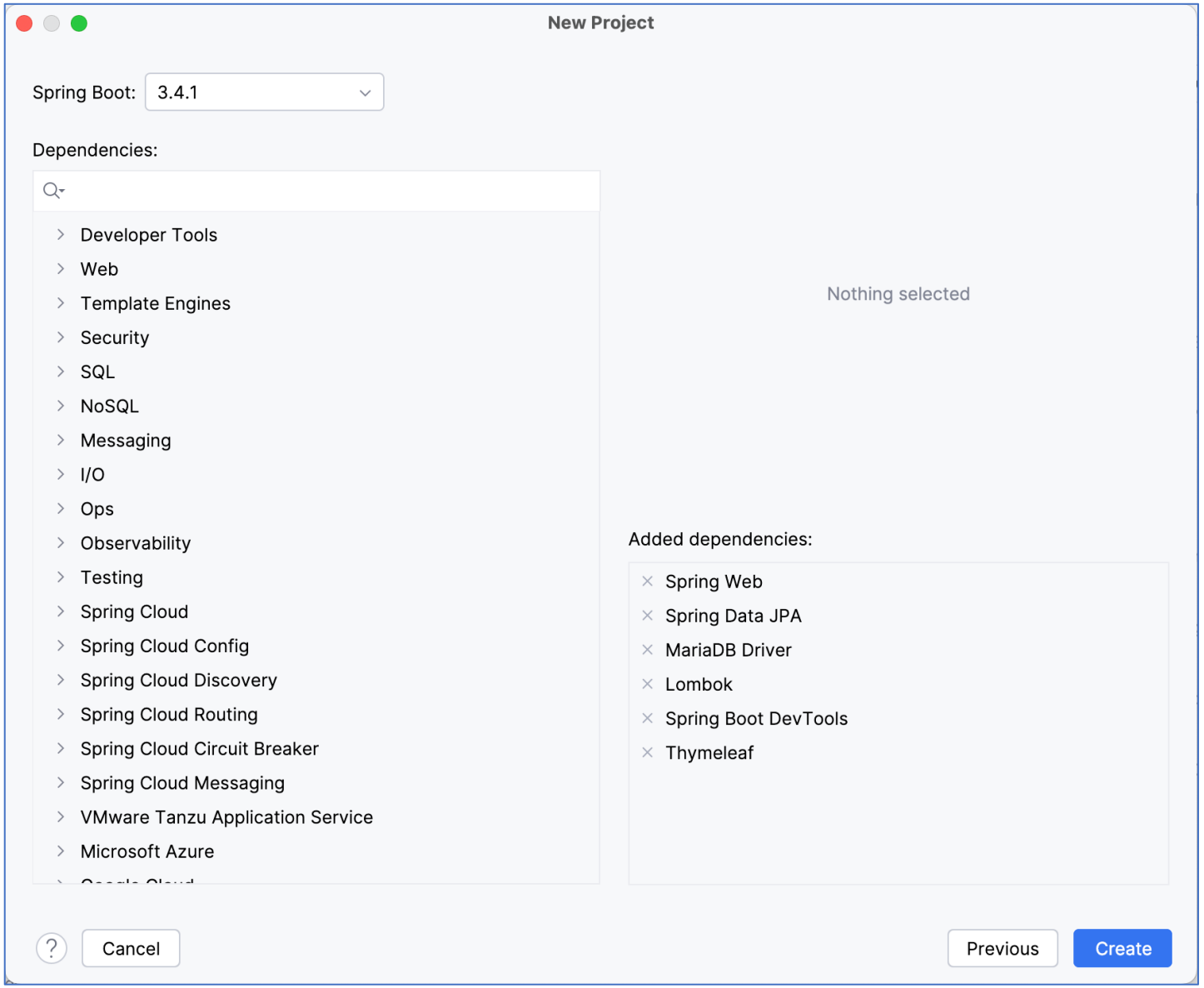
# Creating a New Spring Boot Project

1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary



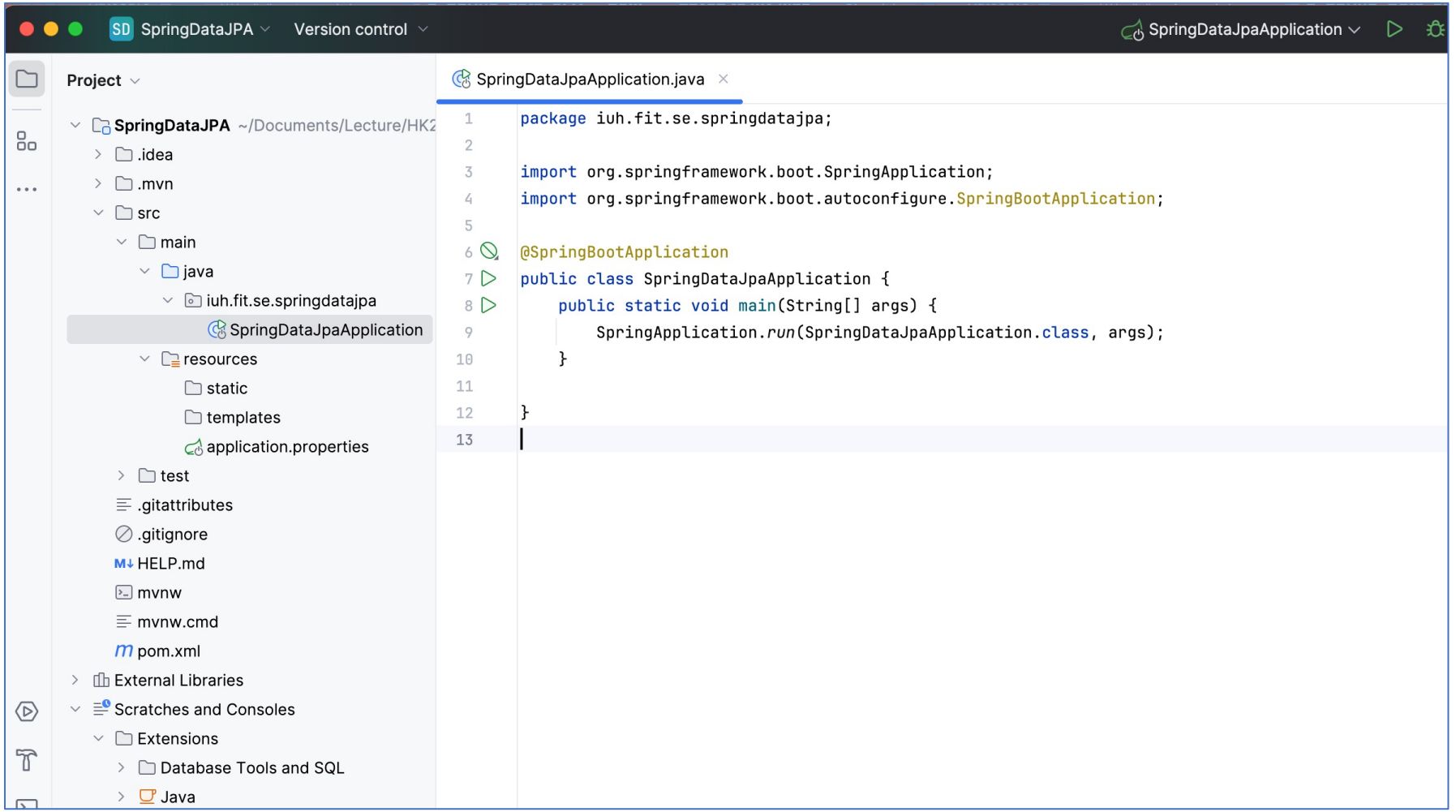
# Creating a New Spring Boot Project (cont.)

1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary



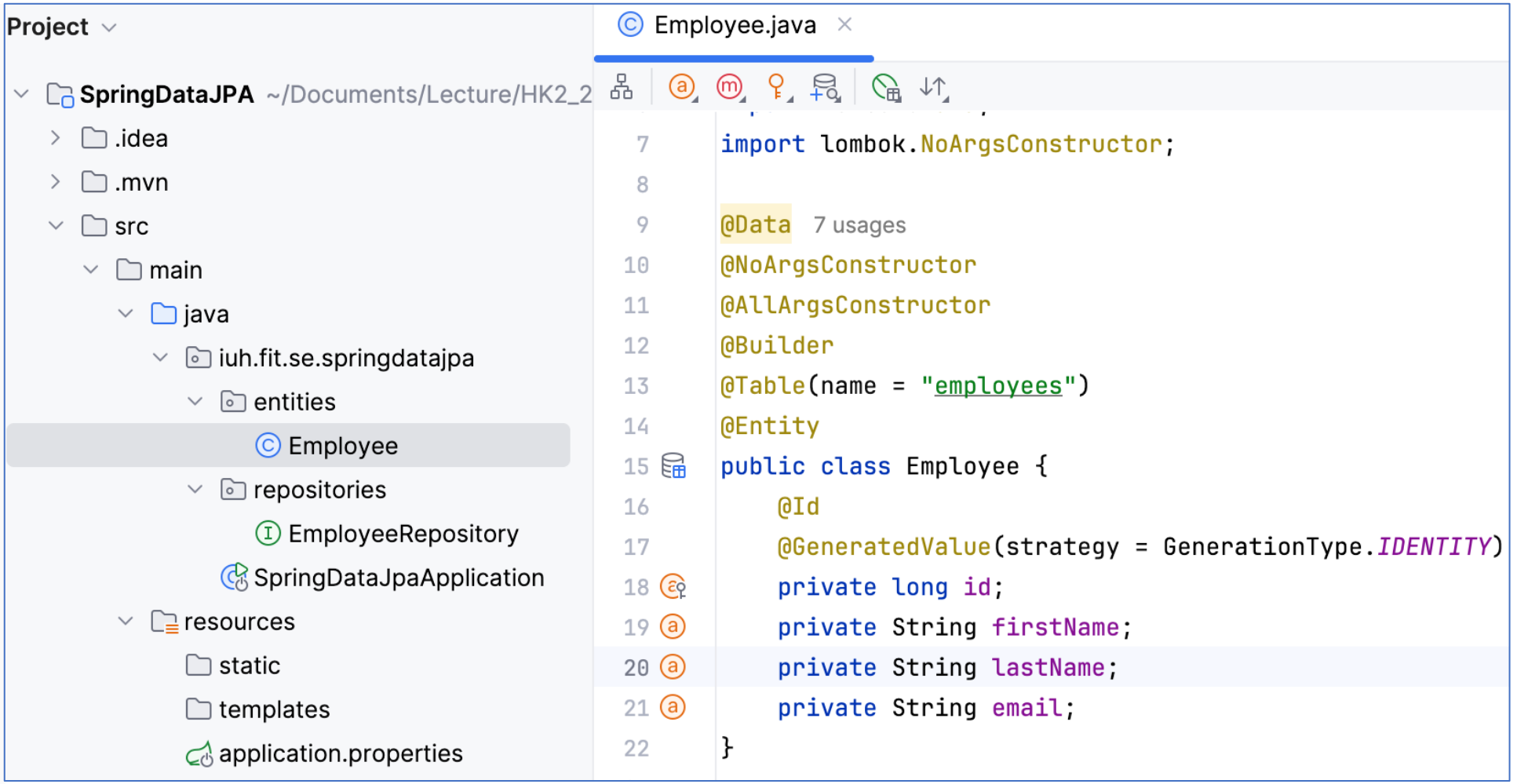
# Creating a New Spring Boot Project (cont.)





# Creating an Employee Entity

1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary



# Creating an Employee Repository Interface



Create a Spring Data repository for CRUD operations.
In order for our interface to be a repository, we'll need it to extend the **CrudRepository<T, ID>** or **JpaRespository<T, ID>** interface with the generic parameters being our entity class and entity's id type.
So for our application, our repository interface definition would be:

**public interface EmployeeRepository extends JpaRepository<Employee, Long>**

# Creating an Employee Repository Interface (cont.)



```
@Repository  no usages
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    List<Employee> f  no usages
}
```

findAllBy
findBy
findEmployeeBy
findEmployeesBy
findDistinctBy
findDistinctFirstBy
findDistinctTopBy
findFirstBy
findTopBy
getDistinctFirstBy
getFirstBy

Press ⏎ to insert, ⇥ to replace

# Creating an Employee Repository Interface (cont.)



```
@Repository  no usages
public interface EmployeeRepository extends
        JpaRepository<Employee, Long> {
    List<Employee> findByLastName(String lastName);  no usages
    List<Employee> findEmployeeByLastNameContaining(String lastName);
}
```

# Configuring Database

1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary

application.properties

```
1  spring.application.name=SpringDataJPA
2
3  # Setting Database
4  spring.data
5
6
```

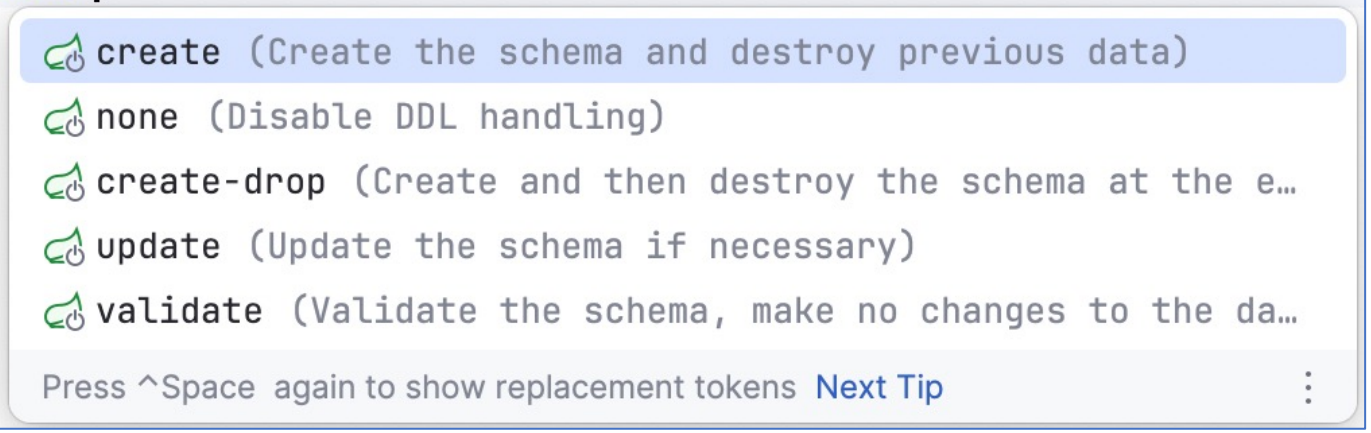
```
spring.datasource.password (Login password of the database) String
spring.datasource.username (Login username of the database) String
spring.datasource.url (JDBC URL of the database) String
spring.datasource.driver-class-name (Fully qualified name … String
spring.data.cassandra.repositories.type=auto (Type… RepositoryType
spring.data.couchbase.auto-index=false (Automatically cre… Boolean
spring.data.couchbase.bucket-name (Name of the bucket to c… String
spring.data.couchbase.field-naming-strategy (Fully quali… Class<?>
spring.data.couchbase.repositories.type=auto (Type… RepositoryType
spring.data.couchbase.scope-name (Name of the scope used f… String
spring.data.couchbase.type-key=_class (Name of the field t… String
```

Press ^. to choose the selected (or first) suggestion and insert a dot afterwards Next Tip

# Configuring Database (cont.)

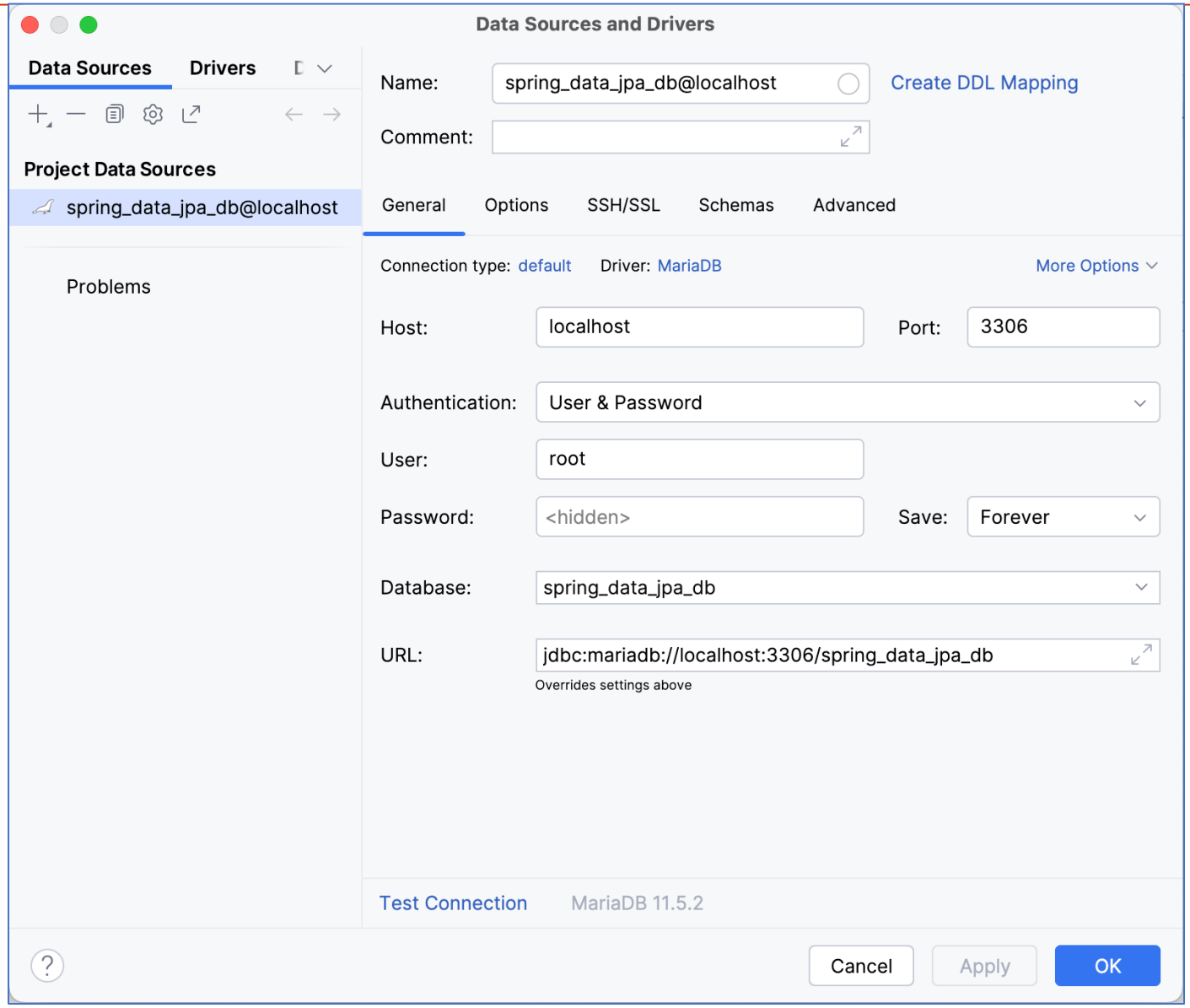


```
# Generate table
spring.jpa.hibernate.ddl-auto=
spring.jpa.show-sql=true
```



# Configuring Database (cont.)

1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary



# Configuring Database (cont.)



application.properties

```
spring.application.name=SpringDataJPA

# Setting Database
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://localhost:3306/spring_data_jpa_db?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=123456

# Generate table
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
```

# Configuring Database (cont.)



**Disable Default Schema Generation and Use SQL Scripts – Create schema and Import data (resources/schema.sql, resources/data.sql)**

```
# Generate table
spring.jpa.hibernate.ddl-auto=none
spring.jpa.show-sql=true

spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode=always
```

**Disable Default Schema Generation and Use SQL Scripts – Create schema and Import data (not default folder)**

```
# Generate table
spring.jpa.hibernate.ddl-auto=none
spring.jpa.show-sql=true

spring.sql.init.mode=always
spring.sql.init.schema-locations=classpath:sql/schema.sql
spring.sql.init.data-locations=classpath:sql/data.sql
```

# Writing from your Application to the DB

1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary

```
private void insertEmployee(EmployeeRepository employeeRepository) {
    int number = new Random().nextInt( bound: 100) + 1;
    Employee employee = Employee.builder()
            .firstName("Van")
            .lastName("Anh" + number)
            .email("vananh" + number + "@gmail.com")
            .build();
    employeeRepository.save(employee);
}
```

# Writing from your Application to the DB (cont.)

1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary

```
@Bean
public CommandLineRunner run(EmployeeRepository employeeRepository) {
    return args -> {
        insertEmployee(employeeRepository);
        System.out.println("Employee List: ");
        employeeRepository.findAll().forEach(System.out::println);
    };
}
```

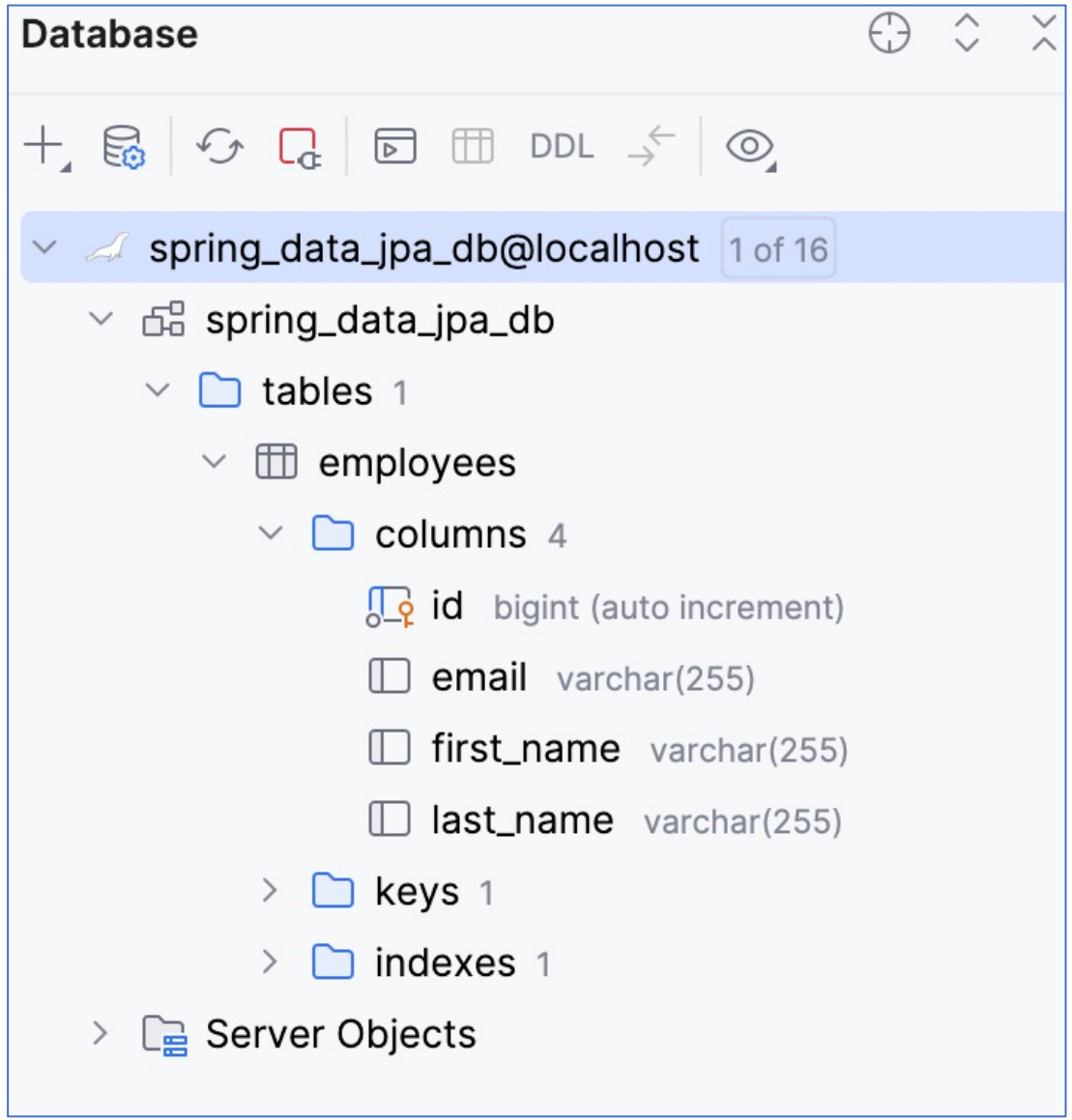
# Writing from your Application to the DB (cont.)

1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary

```
@SpringBootApplication
public class SpringDataJpaApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringDataJpaApplication.class, args);
    }

    private void insertEmployee(EmployeeRepository employeeRepository) {...}

    @Bean
    public CommandLineRunner run(EmployeeRepository employeeRepository) {...}
}
```

# Creating a Data Source

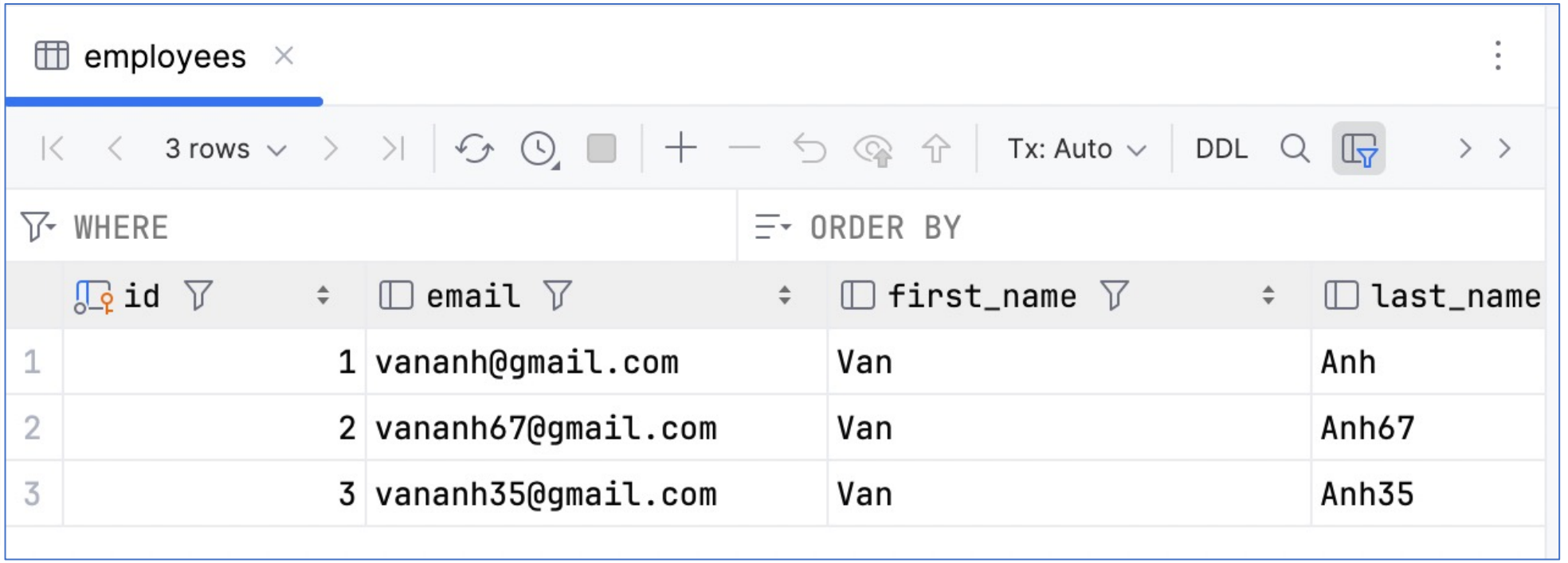
1. Introduction
2. Creating a New Spring Boot Project
3. Creating an Employee Entity
4. Creating a Repository Interface
5. Configuring your Database
6. Writing from your Application to the Database
7. Creating a Data Source
8. Calling a Custom Query
9. Summary



# Creating a Data Source (cont.)



```
Hibernate: insert into employees (email,first_name,last_name) values (?,?,?)
Employee List:
Hibernate: select e1_0.id,e1_0.email,e1_0.first_name,e1_0.last_name from employees e1_0
Employee(id=1, firstName=Van, lastName=Anh, email=vananh@gmail.com)
Employee(id=2, firstName=Van, lastName=Anh67, email=vananh67@gmail.com)
Employee(id=3, firstName=Van, lastName=Anh35, email=vananh35@gmail.com)
```




| | id | email | first_name | last_name |
|---|---|---|---|---|
| 1 | 1 | vananh@gmail.com | Van | Anh |
| 2 | 2 | vananh67@gmail.com | Van | Anh67 |
| 3 | 3 | vananh35@gmail.com | Van | Anh35 |

# Spring Data JPA - Create

1. Create
2. Read
3. Update
4. Delete
5. Search

```
private void insertEmployee(EmployeeRepository employeeRepository) {
    int number = new Random().nextInt( bound: 100) + 1;
    Employee employee = Employee.builder()
            .firstName("Van")
            .lastName("Anh" + number)
            .email("vananh" + number + "@gmail.com")
            .build();
    employeeRepository.save(employee);
}
```

# Spring Data JPA - Read

1. Create
2. Read
3. Update
4. Delete
5. Search

```java
// JPQL - Java Persistence Query Language
@Query("SELECT e FROM Employee e WHERE e.id = :id")  1 usage
Employee findEmployeeById(@Param("id") Long id);

// Native Query
@Query(value = "SELECT * FROM employees", nativeQuery = true)
List<Employee> findAllEmployee();
```

```java
// Find by Id
Employee employee = employeeRepository.findEmployeeById(Long.valueOf( l: 1));
System.out.println("Employee: " + employee);

// Find All Employee
employeeRepository.findAllEmployee().forEach(System.out::println);
```

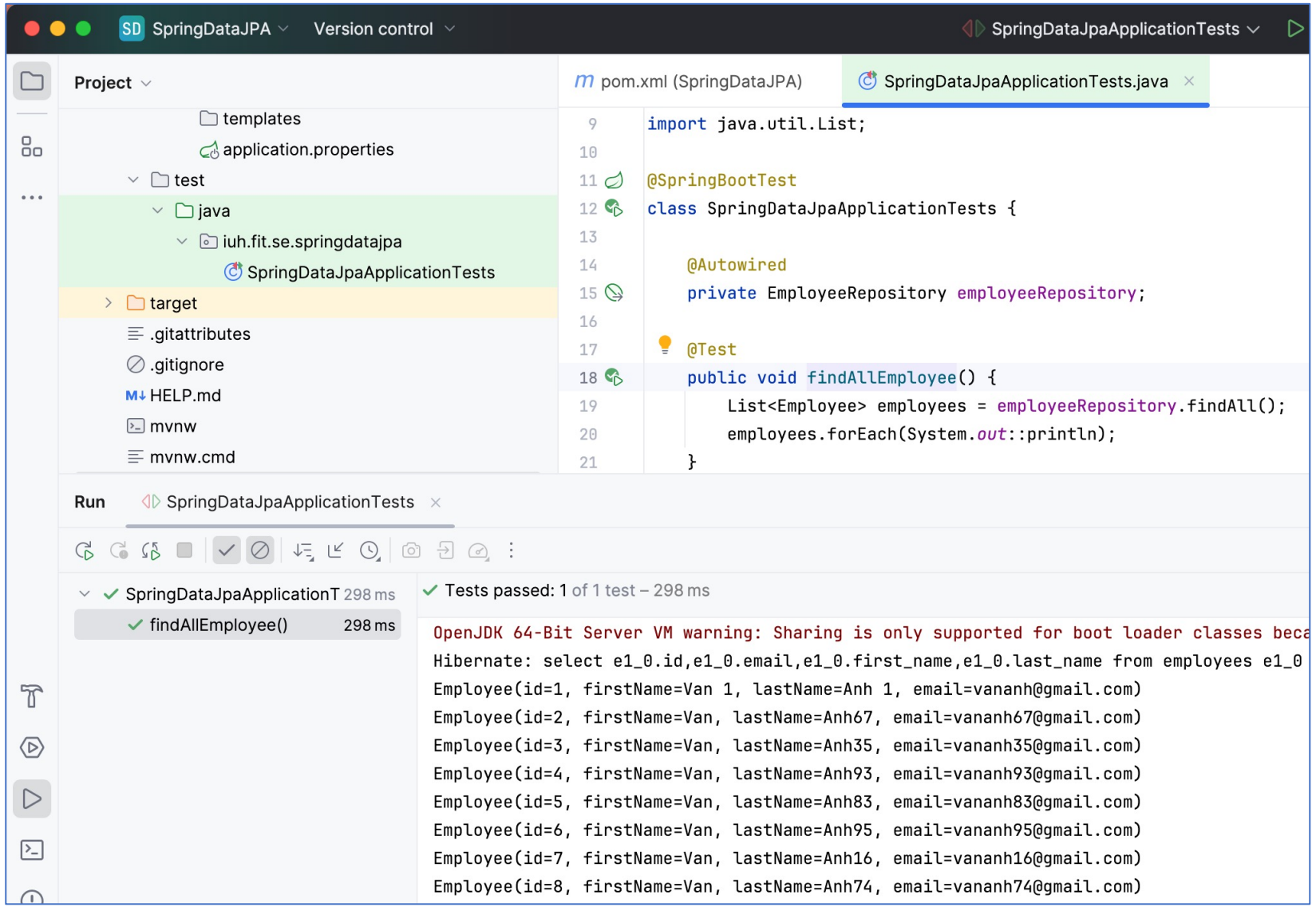
# Spring Data JPA – Read – Unit Test

1. Create
2. Read
3. Update
4. Delete
5. Search

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

# Spring Data JPA – Read – Unit Test (cont.)

1. Create
2. Read
3. Update
4. Delete
5. Search



# Spring Data JPA - Update

1. Create
2. Read
3. Update
4. Delete
5. Search

```
@Transactional  no usages
@Modifying
@Query("UPDATE Employee e SET e.firstName = ?2, e.lastName = ?3 WHERE e.id = ?1")
void updateById(Long id, String firstName, String lastName);
```

```
// Update
employeeRepository.updateById(Long.valueOf( l: 1),
        firstName: "Van 1",  lastName: "Anh 1");
```

```
Hibernate: update employees e1_0 set first_name=?,last_name=? where e1_0.id=?
Hibernate: select e1_0.id,e1_0.email,e1_0.first_name,e1_0.last_name from employees e1_0 where e1_0.id=?
Employee: Employee(id=1, firstName=Van 1, lastName=Anh 1, email=vananh@gmail.com)
```

# Spring Data JPA - Delete

1. Create
2. Read
3. Update
4. Delete
5. Search

```java
@Modifying  no usages
@Query("DELETE FROM Employee e WHERE e.email = :email")
void deleteByEmail(@Param("email") String email);
```

```java
// Delete
employeeRepository.deleteByEmail("vananh@gmail.com");
```

# Spring Data JPA - Search

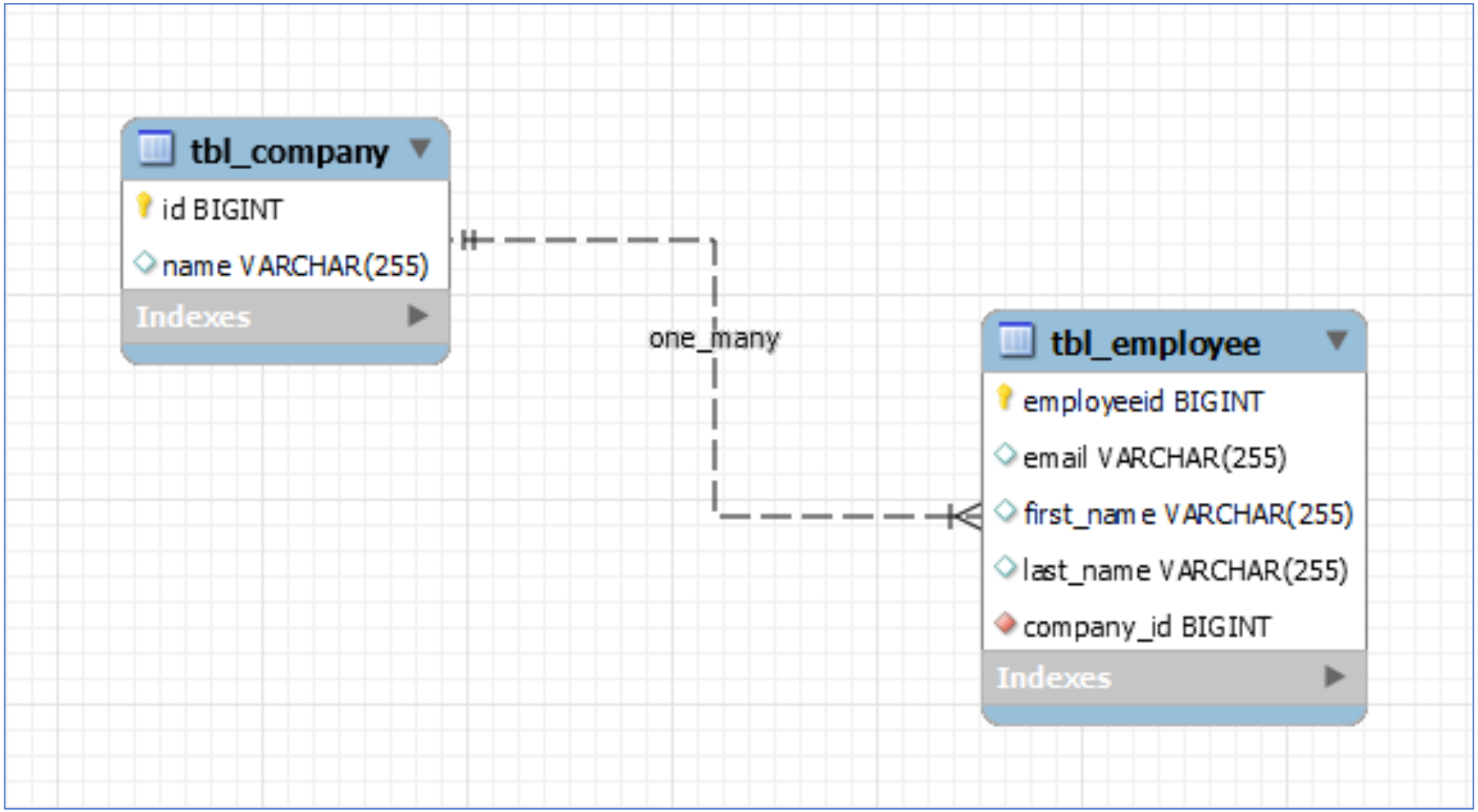
1. Create
2. Read
3. Update
4. Delete
5. Search

```
List<Employee> findByFirstNameLikeOrLastNameLike(String firstName,
                                                 String lastName);
```

```
// Search
employeeRepository.
        findByFirstNameLikeOrLastNameLike("van", "anh35").
        forEach(System.out::println);
```

# One To Many – Many To One

# OneToMany - ManyToOne



# OneToMany – ManyToOne: Generate table

```java
@Data
@Entity
@Table(name = "tbl_company")
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Company {
    @Id
    @Column(name = "id")
    @GeneratedValue
    private Long id;

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

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "company")
    private Set<Employee> listEmployee = new HashSet<>();
}
```

```java
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Table(name = "tbl_employee")
@ToString
public class Employee {
    @Id
    @GeneratedValue
    private long EmployeeID;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email")
    private String email;

    @ManyToOne
    @JoinColumn(name ="company_id", nullable = false )
    private Company company;
}
```

# OneToMany – ManyToOne: Create data

```
Company company = Company.builder()
        .name("TNHH")
        .build();
companyRepository.save(company);

Employee employee = Employee.builder()
        .firstName("Thuc")
        .lastName("Doan")
        .email("nttd@gmail.com")
        .company(company)
        .build();
employeeRepository.save(employee);

Employee employee1 = Employee.builder()
        .firstName("Trong")
        .lastName("Khoi")
        .email("ntk@gmail.com")
        .company(company)
        .build();
employeeRepository.save(employee1);

System.out.println("just get log!");
```

```
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Table(name = "tbl_employee")
@ToString(exclude = "company")
```

```
@Data
@Entity
@Table(name = "tbl_company")
@NoArgsConstructor
@AllArgsConstructor
@Builder
@ToString(exclude = "listEmployee")
public class Company {
```

# OneToMany – ManyToOne: Get data

```
//Native
@Query( value = "select * from tbl_employee where company_id=?1",
        nativeQuery = true
)
public List<Employee> findEmployeeByCompanyId(Long id);
```

```
@Test
public void findEmployeeByCompanyId(){
        List<Employee> employees =
                employeeRepository.findEmployeeByCompanyId(Long.valueOf(5));

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

# OneToMany – ManyToOne: Query Result

Output | schooldb.tbl_employee

2 rows | Tx: Auto | DDL

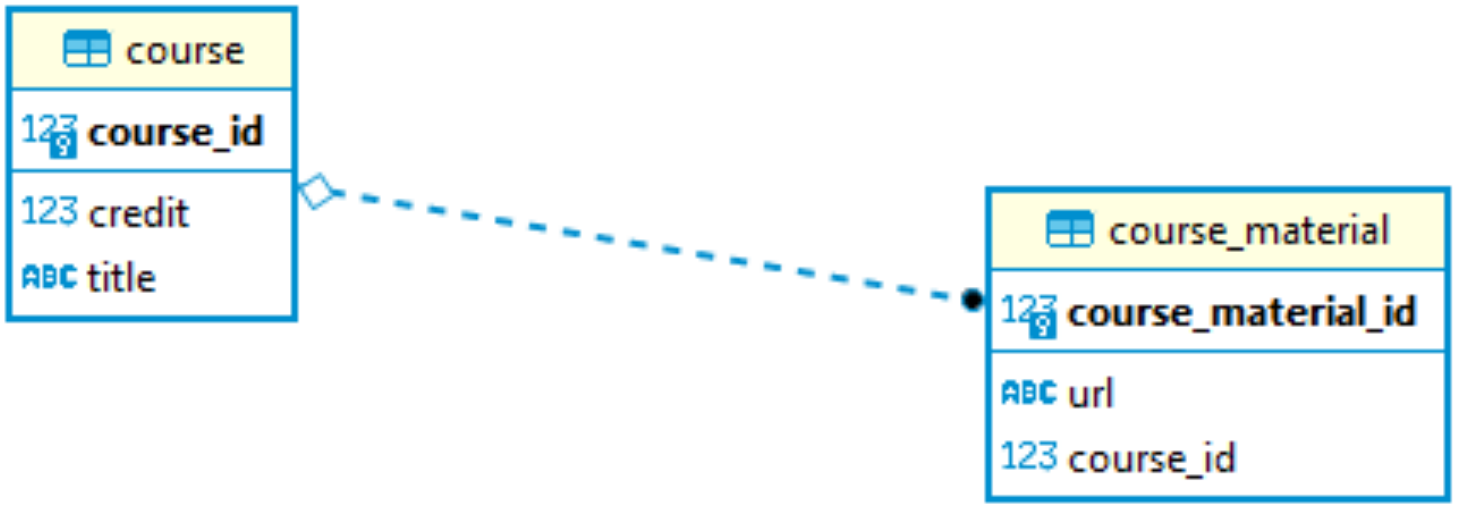
| | employeeid | email | first_name | last_name | company_id |
|---|---|---|---|---|---|
| 1 | 6 | nttd@gmail.com | Thuc | Doan | 5 |
| 2 | 7 | ntk@gmail.com | Trong | Khoi | 5 |

Tests passed: 1 of 1 test – 795 ms

Test Results 795 ms
EmployeeRepositoryTest 795 ms
findEmployeeByCompanyId() 795 ms

```
        company0_.id as id1_1_0_,
        company0_.name as name2_1_0_,
        listemploy1_.company_id as company_5_2_1_,
        listemploy1_.employeeid as employee1_2_1_,
        listemploy1_.employeeid as employee1_2_2_,
        listemploy1_.company_id as company_5_2_2_,
        listemploy1_.email as email2_2_2_,
        listemploy1_.first_name as first_na3_2_2_,
        listemploy1_.last_name as last_nam4_2_2_
    from
        tbl_company company0_
    left outer join
        tbl_employee listemploy1_
            on company0_.id=listemploy1_.company_id
    where
        company0_.id=?
[Employee(EmployeeID=6, firstName=Thuc, lastName=Doan, email=nttd@gmail.com), Employee(EmployeeID=7, firstName=Trong, lastName=Khoi, email=ntk@gmail.com)]
```

# Relationship @OneToOne



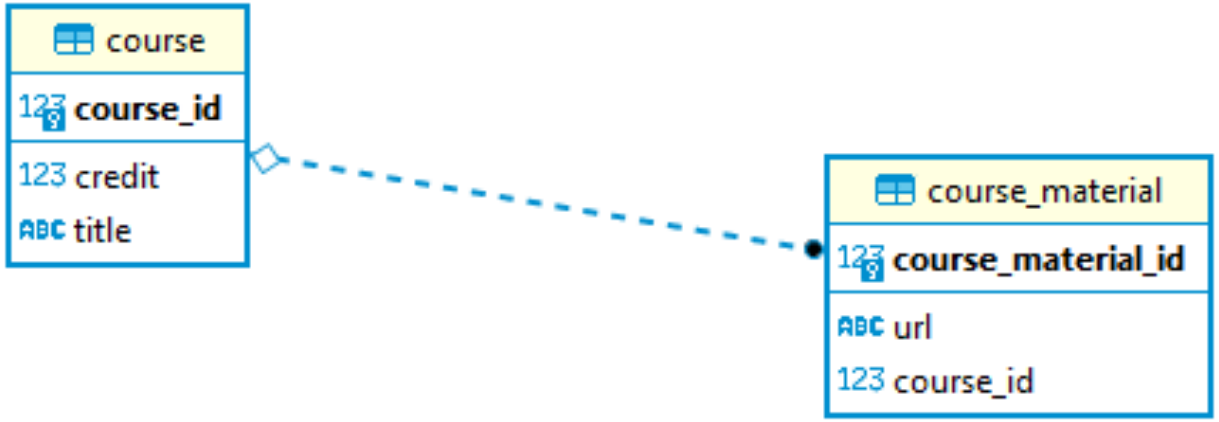
# Relationship @OneToOne (cont.)

```java
@Entity
@Builder
public class Course {

    @Id
    @SequenceGenerator(
            name = "course_sequence",
            sequenceName = "course_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "course_sequence"
    )
    private Long courseId;
    private String title;
    private Integer credit;
}
```

```java
@Entity
@Builder
public class CourseMaterial {
    @Id
    @SequenceGenerator(
            name = "course_material_sequence",
            sequenceName = "course_material_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "course_material_sequence"
    )
    private Long courseMaterialId;
    private String url;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(
            name = "course_id",
            referencedColumnName = "courseId"
    )
    private Course course;
}
```

# Relationship @OneToOne (cont.)

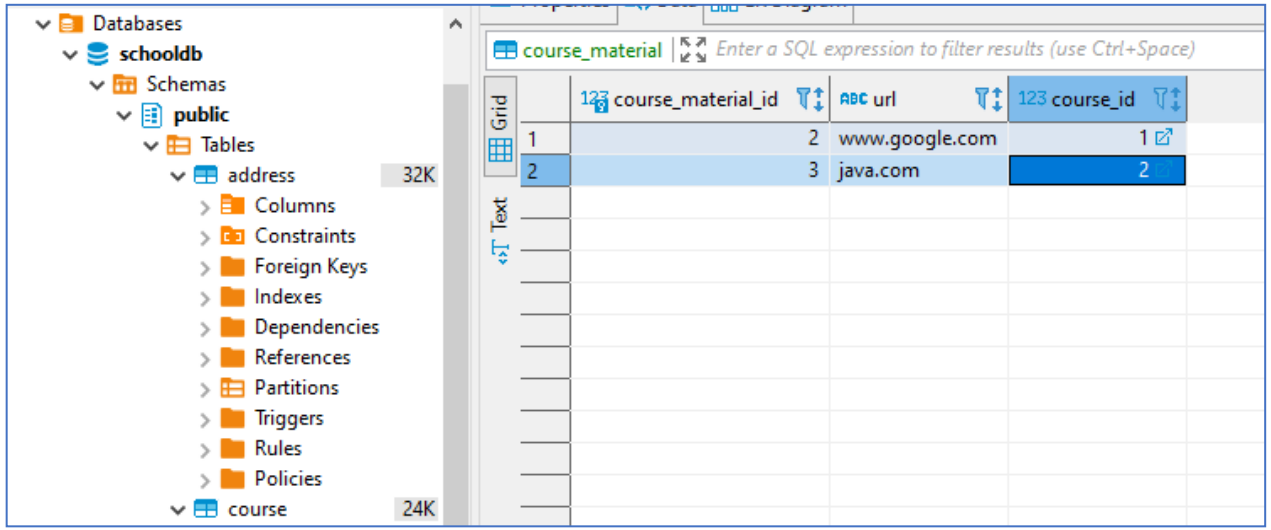
```java
@Entity
@Builder
public class CourseMaterial {
    @Id
    @SequenceGenerator(
            name = "course_material_sequence",
            sequenceName = "course_material_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "course_material_sequence"
    )
    private Long courseMaterialId;
    private String url;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(
            name = "course_id",
            referencedColumnName = "courseId"
    )
    private Course course;
}
```

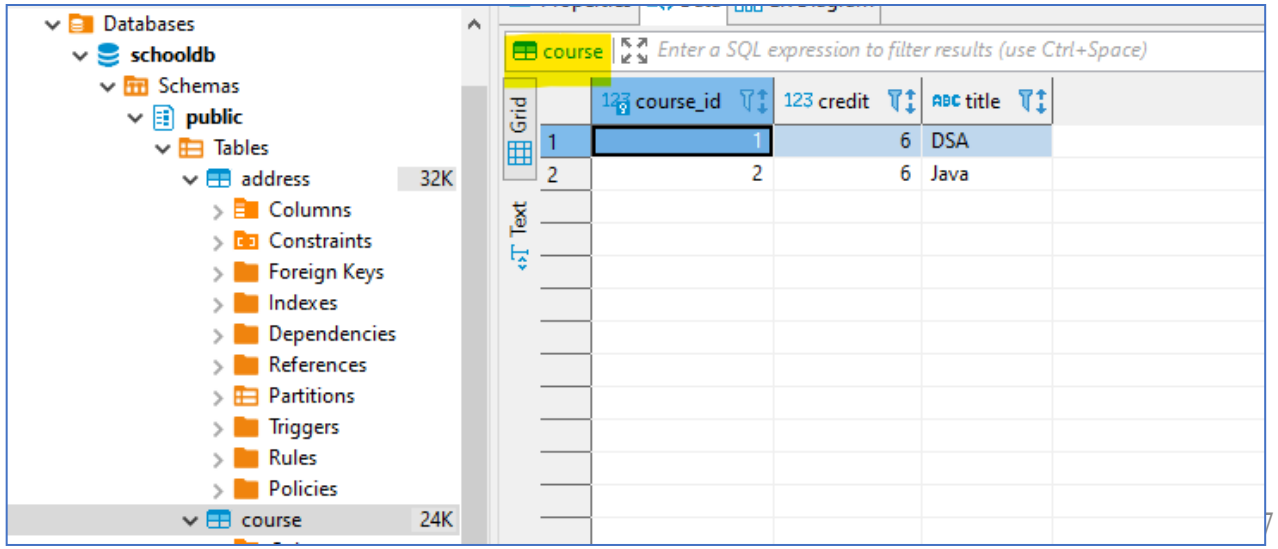


# Relationship @OneToOne (cont.)

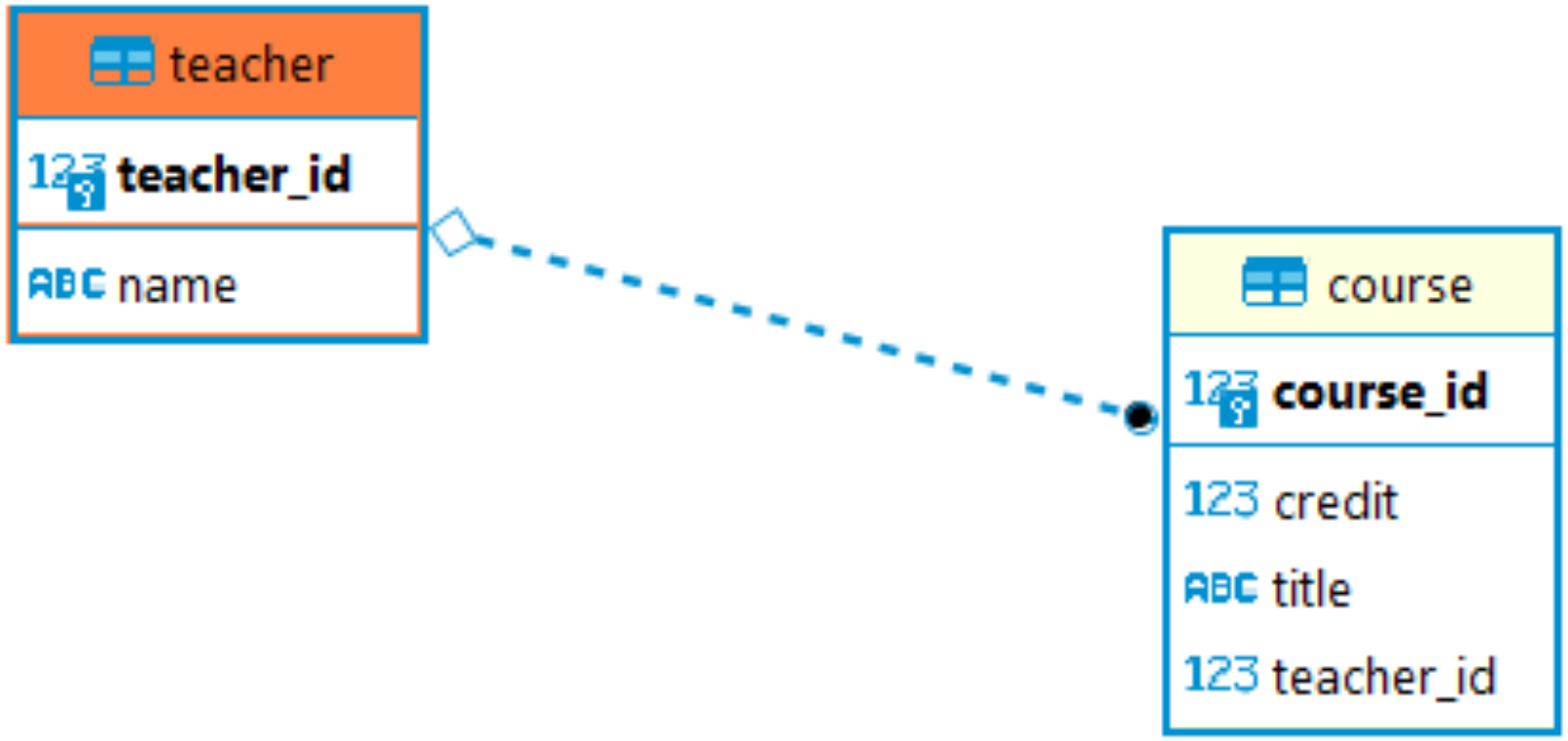
```
Course course = Course.builder()
        .title("Java")
        .credit(6)
        .build();

CourseMaterial courseMaterial =
        CourseMaterial
                .builder()
                .url("java.com")
                .course(course)
                .build();
repository.save(courseMaterial);
```

```
  @OneToOne(cascade = CascadeType.ALL)
  @JoinColumn(
          name = "course_id",
          referencedColumnName = "courseId"
  )
  private Course course;
```



# Relationship @OneToMany



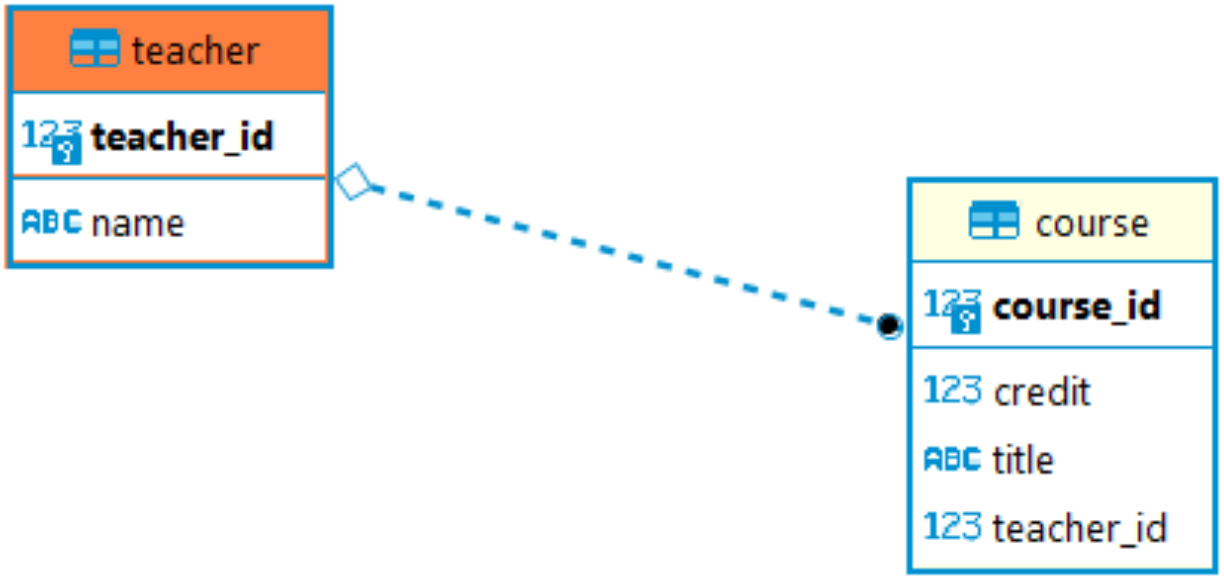
# Relationship @OneToMany (cont.)

```java
@ToString(exclude = "courses")
public class Teacher {
    @Id
    @SequenceGenerator(
            name = "teacher_sequence",
            sequenceName = "teacher_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "teacher_sequence"
    )
    private Long teacherId;
    private String name;

    @OneToMany(
            cascade = CascadeType.ALL
    )
    @JoinColumn(
            name = "teacher_id",
            referencedColumnName = "teacherId"
    )
    private List<Course> courses;
}
```

```java
@ToString
public class Course {

    @Id
    @SequenceGenerator(
            name = "course_sequence",
            sequenceName = "course_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "course_sequence"
    )
    private Long courseId;
    private String title;
    private Integer credit;

    @OneToOne(mappedBy = "course")
    private CourseMaterial courseMaterial;
}
```

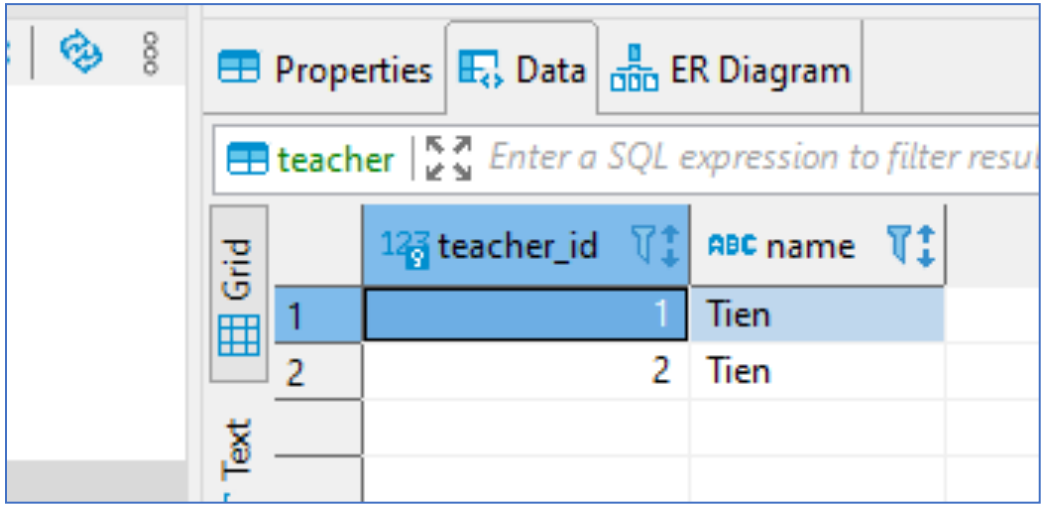
# Relationship @OneToMany (cont.)

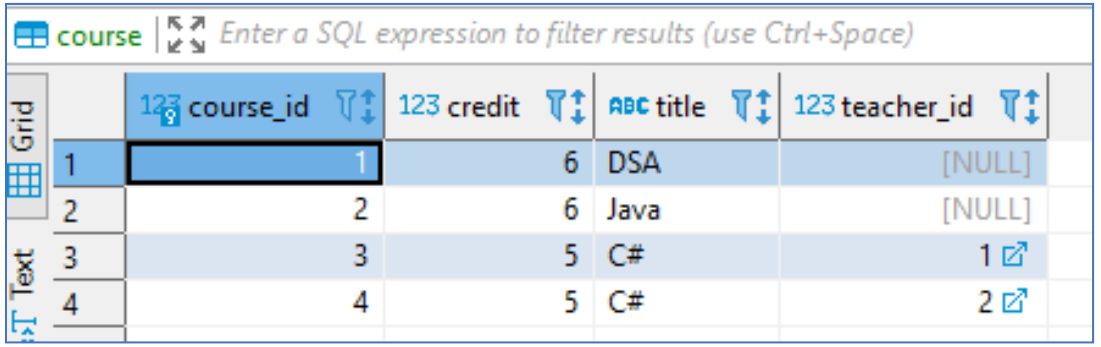
```java
@ToString(exclude = "courses")
public class Teacher {
    @Id
    @SequenceGenerator(
            name = "teacher_sequence",
            sequenceName = "teacher_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "teacher_sequence"
    )
    private Long teacherId;
    private String name;

    @OneToMany(
            cascade = CascadeType.ALL
    )
    @JoinColumn(
            name = "teacher_id",
            referencedColumnName = "teacherId"
    )
    private List<Course> courses;
}
```



# Relationship @OneToMany (cont.)

```java
@Test
public void saveTeacher(){
    Course course = Course.builder()
            .title("C#")
            .credit(5)
            .build();

    Teacher teacher =
            Teacher.builder()
                    .name("Tien")
                    .courses(List.of(course))
                    .build();

    repository.save(teacher);

}
```

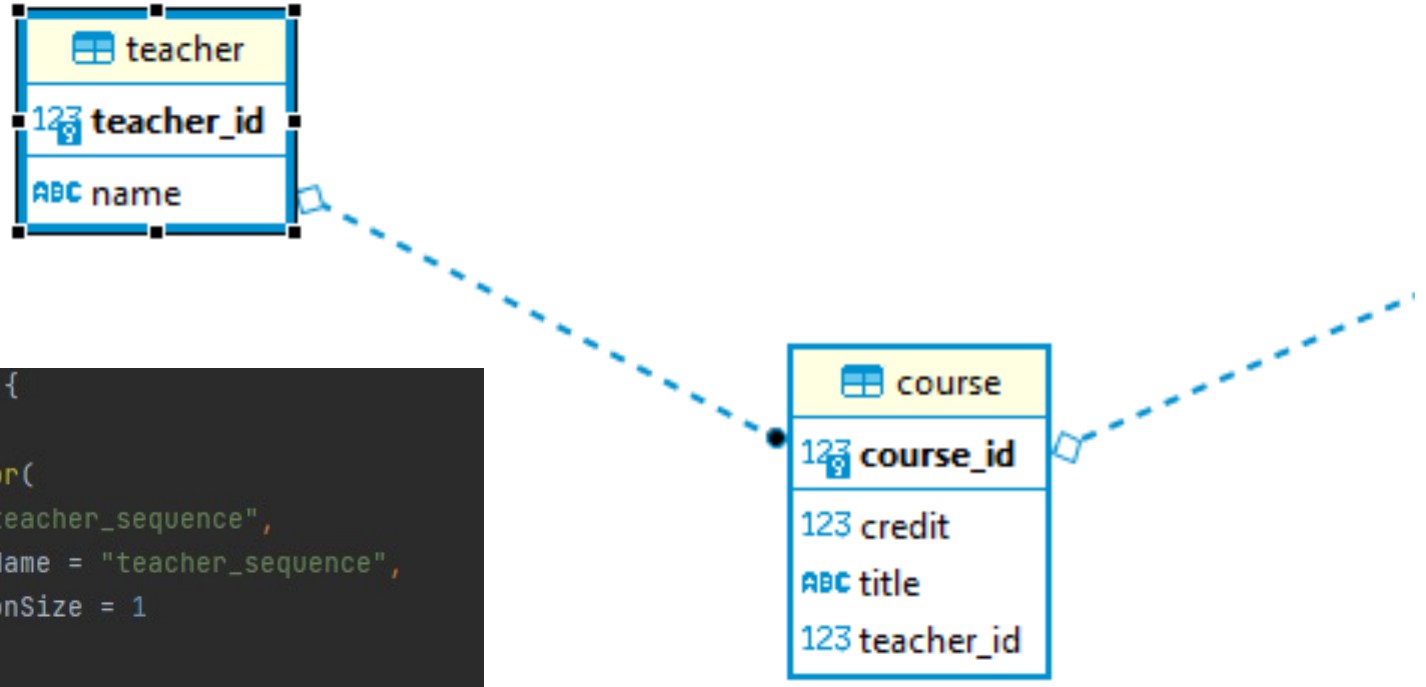


# Relationship @ManyToOne (cont.)

```java
@ToString
public class Course {

    @Id
    @SequenceGenerator(
            name = "course_sequence",
            sequenceName = "course_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "course_sequence"
    )
    private Long courseId;
    private String title;
    private Integer credit;

    @OneToOne(mappedBy = "course")
    private CourseMaterial courseMaterial;

    @ManyToOne(
            cascade = CascadeType.ALL
    )
    @JoinColumn(
            name = "teacher_id",
            referencedColumnName = "teacherId"
    )
    private Teacher teacher;
}
```

```java
public class Teacher {
    @Id
    @SequenceGenerator(
            name = "teacher_sequence",
            sequenceName = "teacher_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "teacher_sequence"
    )
    private Long teacherId;
    private String name;
```



# Relationship @ManyToOne (cont.)

```java
@Test
public void saveCourseWithTeacher(){

    Teacher teacher =
            Teacher.builder()
                    .name("Tien")
                    //.courses(List.of(course))
                    .build();
    Course course = Course.builder()
            .title("Javascript")
            .credit(5)
            .teacher(teacher)
            .build();

    repository.save(course);
}
```

course | Enter a SQL expression to filter results (use Ctrl+Space)

| | course_id | credit | title | teacher_id |
|---|---|---|---|---|
| 1 | 1 | 6 | DSA | [NULL] |
| 2 | 2 | 6 | Java | [NULL] |
| 3 | 3 | 5 | C# | 1 |
| 4 | 4 | 5 | C# | 2 |
| 5 | 5 | 5 | C# | 3 |
| 6 | 6 | 5 | Javascript | 4 |

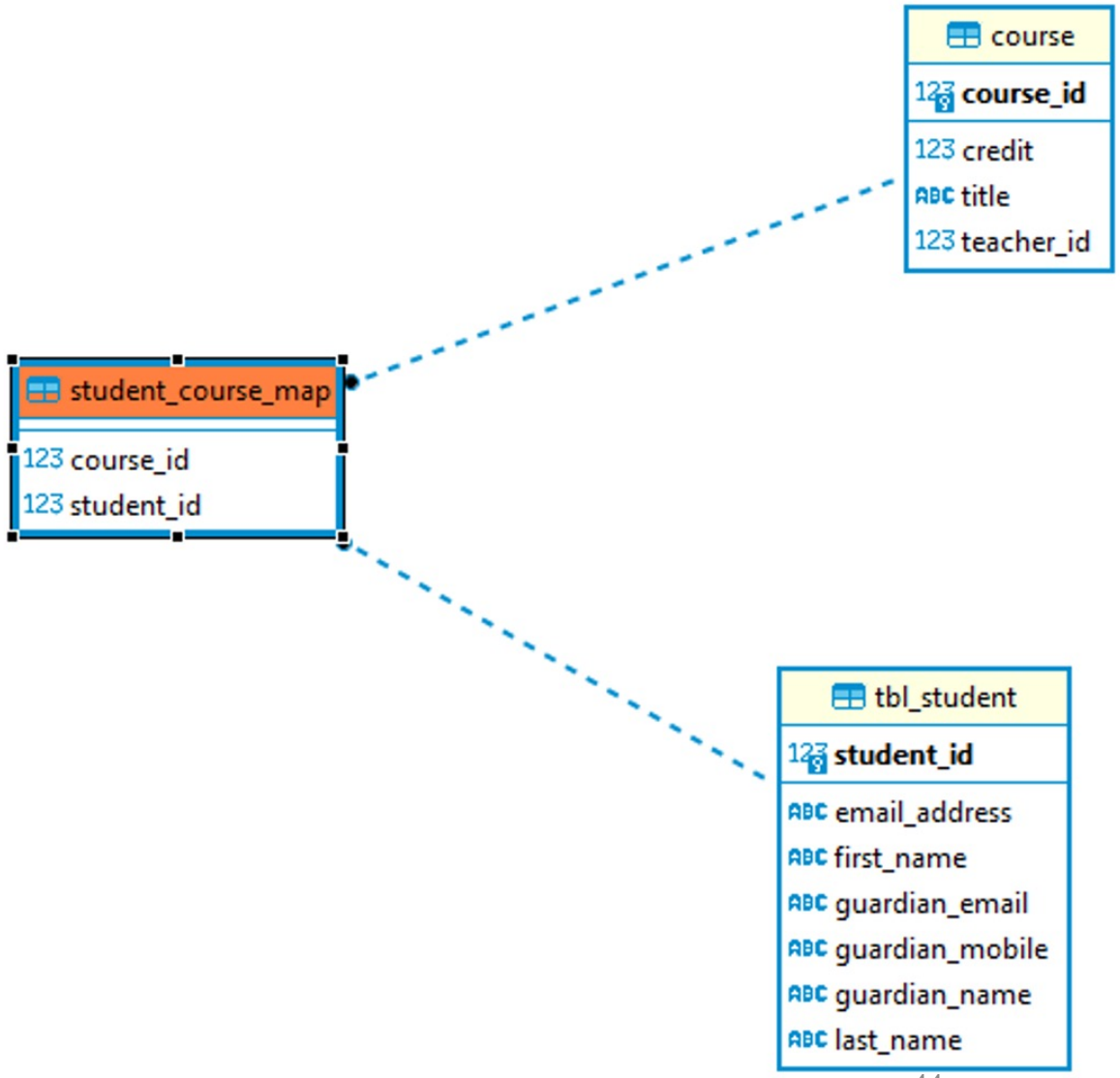
Properties | Data | ER Diagram

teacher | Enter a SQL expression to filter results (use

| | teacher_id | name |
|---|---|---|
| 1 | 1 | Tien |
| 2 | 2 | Tien |
| 3 | 3 | Tien |
| 4 | 4 | Tien |

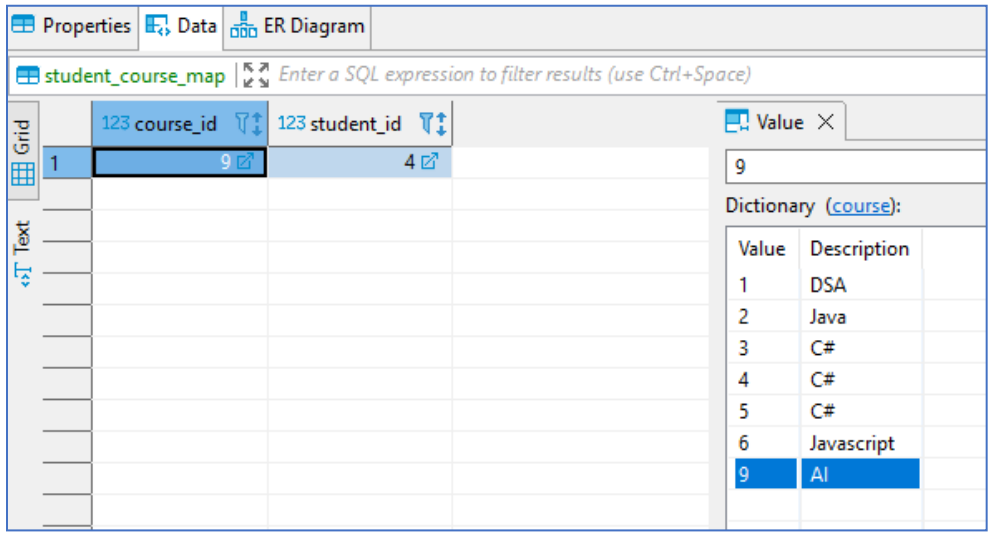
# Relationship @ManyToMany (cont.)

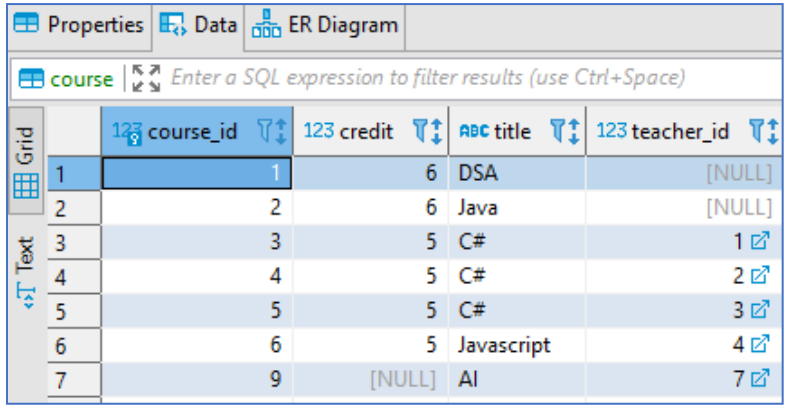
```java
@ManyToMany()
@JoinTable(
        name = "student_course_map",
        joinColumns = @JoinColumn(
                name = "course_id",
                referencedColumnName = "courseId"
        ),
        inverseJoinColumns = @JoinColumn(
         name = "student_id",
        referencedColumnName = "studentId"
        )
)
private List<Student> students;
```

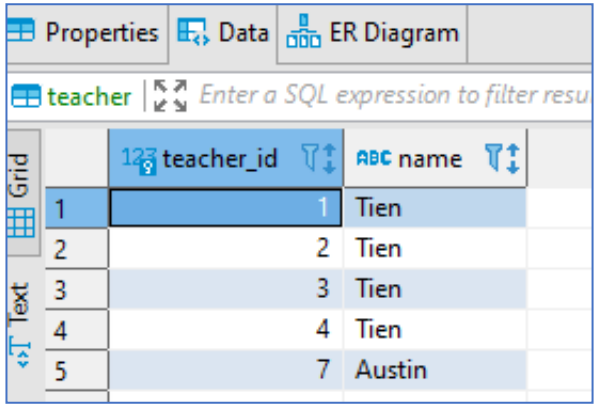


# Relationship @ManyToMany (cont.)

```java
public void saveCourseWithStudentAndTeacher(){
    Teacher teacher = Teacher
            .builder()
            .name("Austin")
            .build();

    Student student = Student
            .builder()
            .firstName("Nguyen")
            .lastName("Khoi")
            .emailId("ntk@gmail.com")
            .build();

    Course course = Course
            .builder()
            .title("AI")
            .teacher(teacher)
            .build();
    course.addStudents(student);
    repository.save(course);
}
```



# FetchType - LAZY

```java
@ToString(exclude = "course")
public class CourseMaterial {
    @Id
    @SequenceGenerator(
            name = "course_material_sequence",
            sequenceName = "course_material_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "course_material_sequence"
    )
    private Long courseMaterialId;
    private String url;

    @OneToOne(cascade = CascadeType.ALL,
            fetch = FetchType.LAZY
    )
    @JoinColumn(
            name = "course_id",
            referencedColumnName = "courseId"
    )
    private Course course;
```

```java
@Test
public void printAllCourseMaterial(){
    List<CourseMaterial> materialList = repository.findAll();
    System.out.println("materialList = " + materialList);
}
```

```
Hibernate:
    select
        coursemate0_.course_material_id as course_m1_2_,
        coursemate0_.course_id as course_i3_2_,
        coursemate0_.url as url2_2_
    from
        course_material coursemate0_
materialList = [CourseMaterial(courseMaterialId=2, url=www.google.com), CourseMaterial(courseMaterialId=3, url=java.com)]
```

# FetchType - EAGER

e.java × CourseMaterial.java × UserRepository.java × StudentRepository.java ×

```
    )
    private Long courseMaterialId;
    private String url;

    @OneToOne(cascade = CascadeType.ALL,
            fetch = FetchType.EAGER
    )
    @JoinColumn(
            name = "course_id",
            referencedColumnName = "courseId"
    )
    private Course course;
}
```

```
@Test
public void printAllCourseMaterial(){
    List<CourseMaterial> materialList = repository.findAll();
    System.out.println("materialList = " + materialList);
}
```

```
Hibernate:
    select
        course0_.course_id as course_i1_1_0_,
        course0_.credit as credit2_1_0_,
        course0_.title as title3_1_0_
    from
        course course0_
    where
        course0_.course_id=?
materialList = [CourseMaterial(courseMaterialId=2, url=www.google.com, course=com.example.education.entity.Course@36ae17ef), CourseMaterial(courseMaterialId=3,
```

# Uni & Bidirectional Relationship - mappedBy

```
@NoArgsConstructor
@ToString
public class Course {

    @Id
    @SequenceGenerator(
            name = "course_sequence",
            sequenceName = "course_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "course_sequence"
    )
    private Long courseId;
    private String title;
    private Integer credit;

    @OneToOne(mappedBy = "course")
    private CourseMaterial courseMaterial;
}
```

```
Hibernate:
    select
        coursemate0_.course_material_id as course_m1_2_0_,
        coursemate0_.course_id as course_i3_2_0_,
        coursemate0_.url as url2_2_0_
    from
        course_material coursemate0_
    where
        coursemate0_.course_id=?
courses = [Course(courseId=1, title=DSA, credit=6, courseMaterial=CourseMaterial(courseMaterialId=2, url=www.google.com)), Course(courseId=2, title=Java, credit=6, cour
```

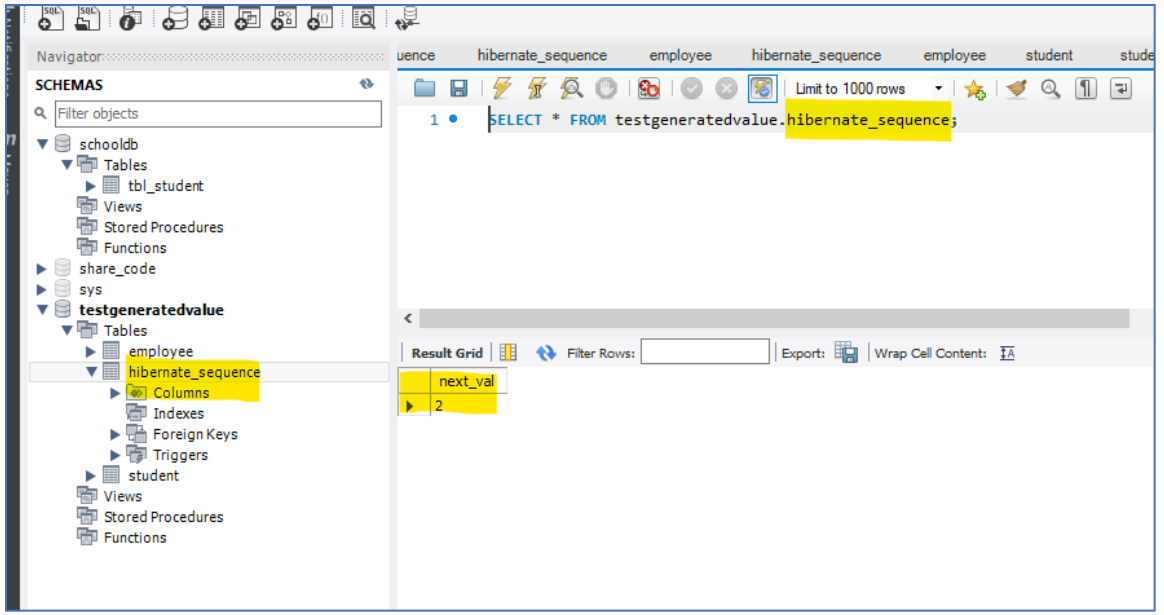
# @GeneratedValue - GenerationType.INDENTITY

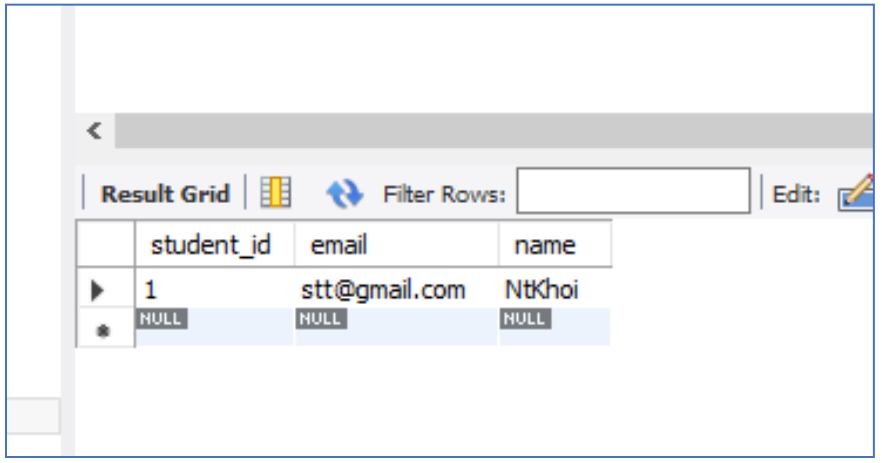
```java
public class Employee {
    @Id
    @Column(name = "employee_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long employeeId;
    private String name;
    private String email;
}
```

# @GeneratedValue - GenerationType.AUTO

```java
public class Student {
    @Id
    @Column(name = "student_id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long Id;
    private String name;
    private String email;
}
```

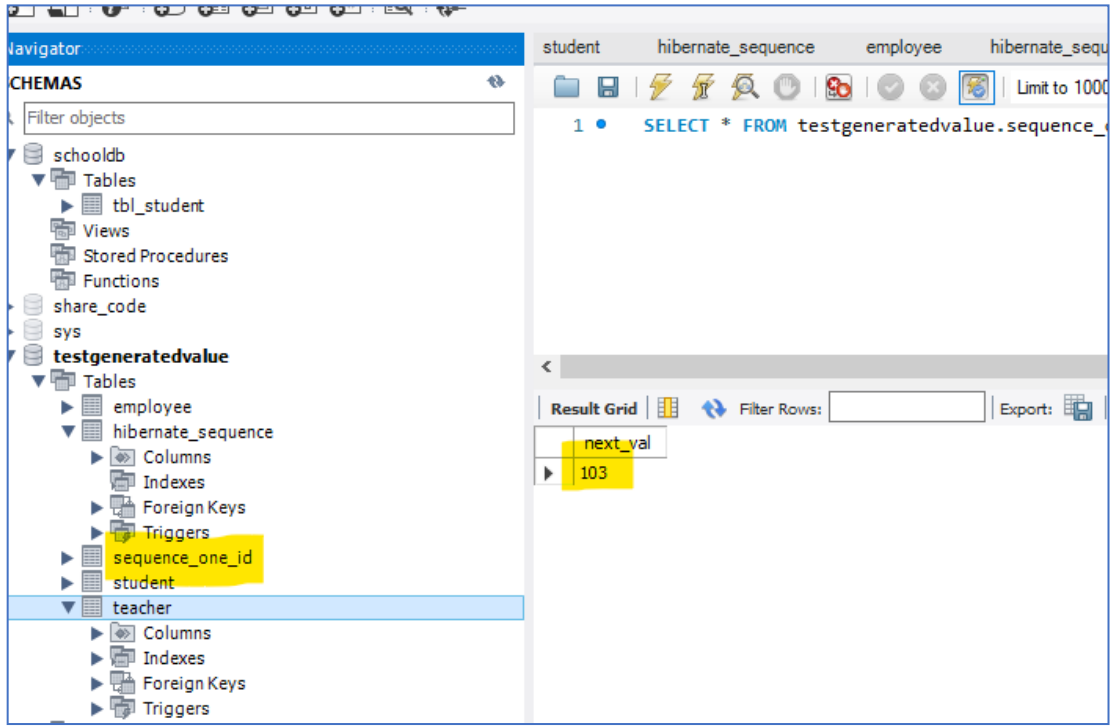
```java
@Test
public void saveStudent(){
    Student student = Student
            .builder()
            .name("NtKhoi")
            .email("stt@gmail.com")
            .build();

    repository.save(student);
}
```

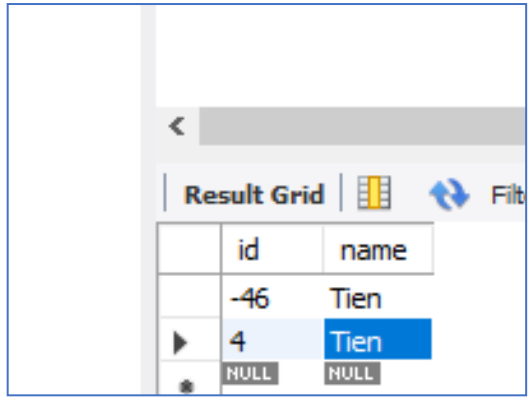


# @GeneratedValue - GenerationType.SEQUENCE

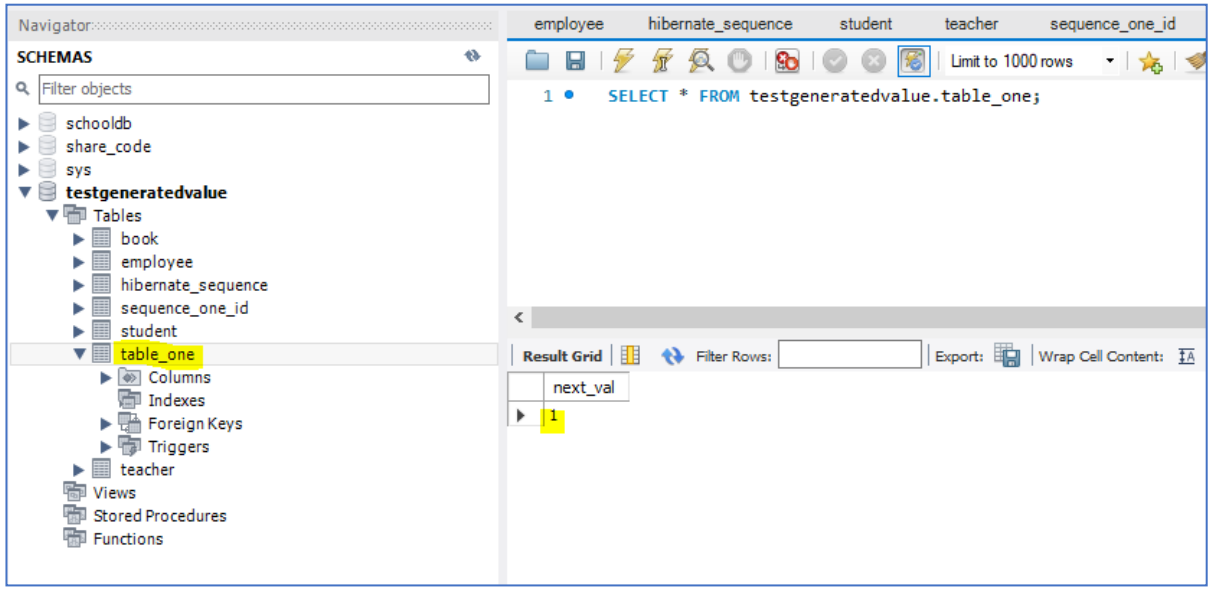
```java
@Builder
public class Teacher {
    @Id
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "teacher_sequence"
    )
    @SequenceGenerator(
            name = "teacher_sequence",
            sequenceName = "sequence_one_id",
            allocationSize = 50
    )
    private Long id;
```

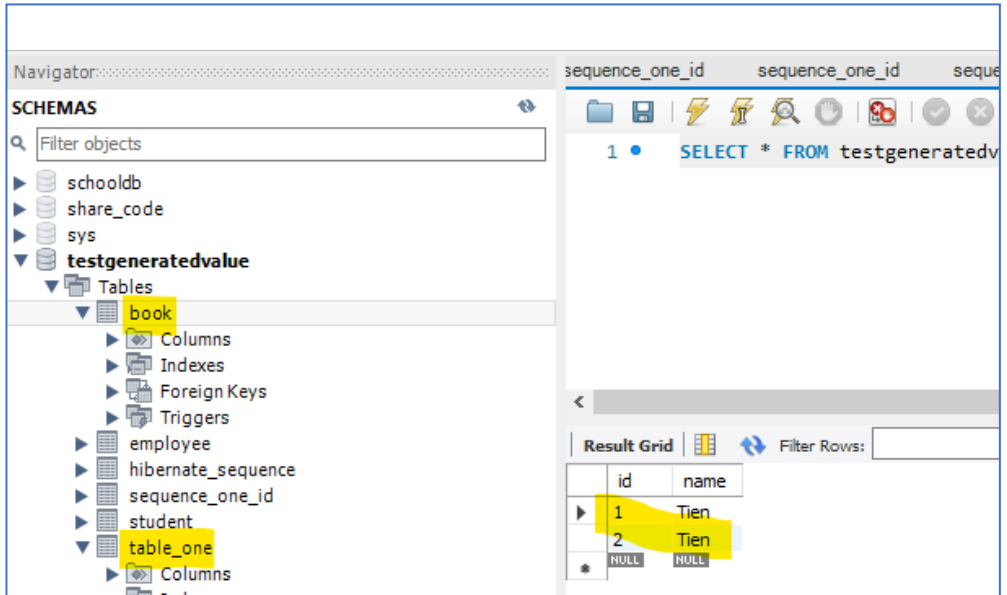


# @GeneratedValue - GenerationType.TABLE

```java
@AllArgsConstructor
public class Book {
    @Id
    @GeneratedValue(
            strategy = GenerationType.TABLE,
            generator = "book_table"
    )
    @SequenceGenerator(
            name = "book_table",
            sequenceName = "table_one",
            allocationSize = 1
    )
    private Long Id;
    private String name;
}
```

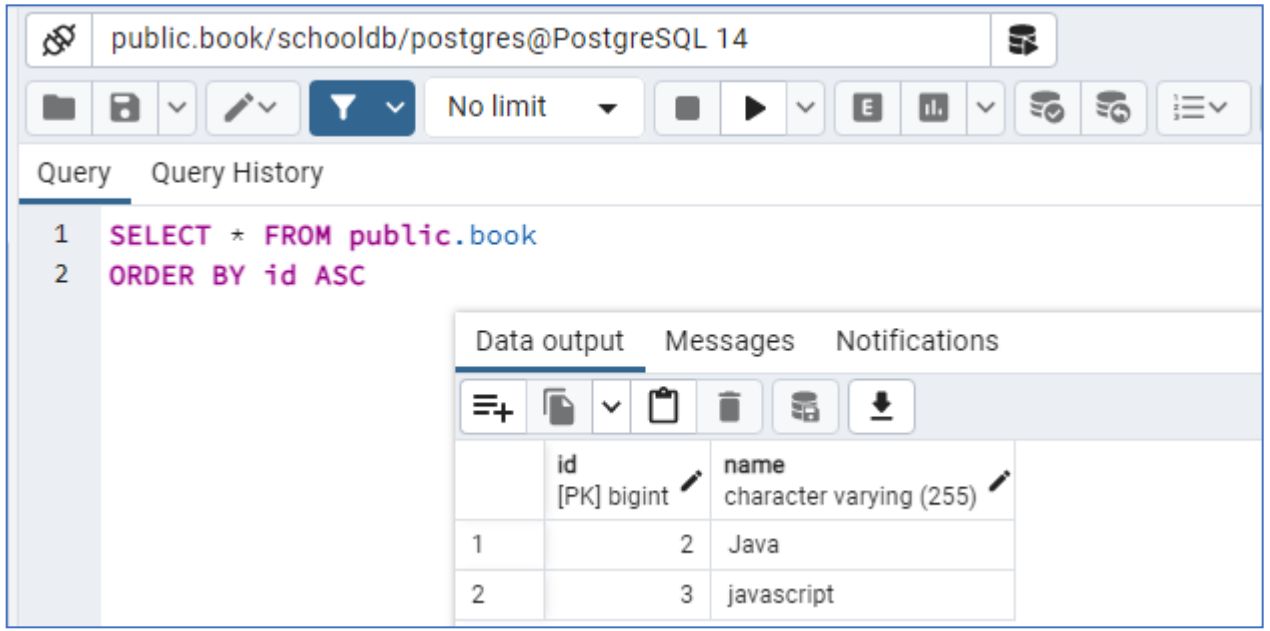


# Understanding Repository and their methods

# JpaRepository

```
1 usage
@Repository
public interface BookRepository extends JpaRepository<Book, Long> {

}
```




```
@Test
public void saveBook(){
    Book java = Book
            .builder()
            .name("Java")
            .build();

    repository.save(java);
    Book javascript = Book
            .builder()
            .name("javascript")
            .build();

    repository.save(javascript);
}
```

# JpaRepository (cont.)

```java
@Test
public void removeBookById(){

    repository.deleteById(1L);
}
```

```
Hibernate:
    select
        book0_.id as id1_0_,
        book0_.name as name2_0_
    from
        book book0_
books = [Book(Id=2, name=Java), Book(Id=3, name=javascript)]
```

```java
@Test
public void findAll(){
    List<Book> books = repository.findAll();
    System.out.println("books = " + books);
}
```

# JpaRepository (cont.)

```
public List<Book> findByNameAndPublishing(String name, String publishing);
```

```
@Test
public void findByNameAndPublishing(){
    List<Book> books = repository.findByNameAndPublishing( name: "ReactJS", publishing: "sharecode");
    System.out.println("books = " + books);
}
```

```
Hibernate:
    select
        book0_.id as id1_0_,
        book0_.name as name2_0_,
        book0_.publishing as publishi3_0_
    from
        book book0_
    where
        book0_.name=?
        and book0_.publishing=?
books = [Book(Id=5, name=ReactJS, publishing=sharecode)]
```

# JpaRepository (cont.)

```
public List<Book> findByNameContaining(String name);
```

```
@Test
public void findByNameContaining(){
    List<Book> books = repository.findByNameContaining("React");
    System.out.println("books = " + books);
}
```

```
Hibernate:
    select
        book0_.id as id1_0_,
        book0_.name as name2_0_,
        book0_.publishing as publishi3_0_
    from
        book book0_
    where
        book0_.name like ? escape ?
books = [Book(Id=4, name=React Native, publishing=sharecode), Book(Id=5, name=ReactJS, publishing=sharecode)]
```

# JpaRepository (cont.)

https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html

| Keyword | Sample | JPQL snippet |
|---|---|---|
| `Distinct` | `findDistinctByLastnameAndFirstname` | `select distinct … where x.lastname = ?1 and x.firstname = ?2` |
| `And` | `findByLastnameAndFirstname` | `… where x.lastname = ?1 and x.firstname = ?2` |
| `Or` | `findByLastnameOrFirstname` | `… where x.lastname = ?1 or x.firstname = ?2` |
| `Is`, `Equals` | `findByFirstname`, `findByFirstnameIs`, `findByFirstnameEquals` | `… where x.firstname = ?1` |
| `Between` | `findByStartDateBetween` | `… where x.startDate between ?1 and ?2` |
| `LessThan` | `findByAgeLessThan` | `… where x.age < ?1` |
| `LessThanEqual` | `findByAgeLessThanEqual` | `… where x.age <= ?1` |
| `GreaterThan` | `findByAgeGreaterThan` | `… where x.age > ?1` |
| `GreaterThanEqual` | `findByAgeGreaterThanEqual` | `… where x.age >= ?1` |
| `After` | `findByStartDateAfter` | `… where x.startDate > ?1` |
| `Before` | `findByStartDateBefore` | `… where x.startDate < ?1` |

# JpaRepository (cont.)

https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html

| | | |
|---|---|---|
| `IsNull`, `Null` | `findByAge(Is)Null` | `… where x.age is null` |
| `IsNotNull`, `NotNull` | `findByAge(Is)NotNull` | `… where x.age not null` |
| `Like` | `findByFirstnameLike` | `… where x.firstname like ?1` |
| `NotLike` | `findByFirstnameNotLike` | `… where x.firstname not like ?1` |
| `StartingWith` | `findByFirstnameStartingWith` | `… where x.firstname like ?1` (parameter bound with appended `%`) |
| `EndingWith` | `findByFirstnameEndingWith` | `… where x.firstname like ?1` (parameter bound with prepended `%`) |
| `Containing` | `findByFirstnameContaining` | `… where x.firstname like ?1` (parameter bound wrapped in `%`) |
| `OrderBy` | `findByAgeOrderByLastnameDesc` | `… where x.age = ?1 order by x.lastname desc` |

# JpaRepository (cont.)

https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html

| | | |
|---|---|---|
| `Not` | `findByLastnameNot` | `… where x.lastname <> ?1` |
| `In` | `findByAgeIn(Collection<Age> ages)` | `… where x.age in ?1` |
| `NotIn` | `findByAgeNotIn(Collection<Age> ages)` | `… where x.age not in ?1` |
| `True` | `findByActiveTrue()` | `… where x.active = true` |
| `False` | `findByActiveFalse()` | `… where x.active = false` |
| `IgnoreCase` | `findByFirstnameIgnoreCase` | `… where UPPER(x.firstname) = UPPER(?1)` |

# @Query - JPQL

# @Query

https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html#jpa.query-methods.at-query

```
1 usage
@Query("select b from Book b where b.name = ?1")
public List<Book> searchBookByName(String name);
```

```
@Test
public void searchBookByName(){
    List<Book> books = repository.searchBookByName("Java");
    System.out.println("books = " + books);
}
```

```
Hibernate:
    select
        book0_.id as id1_0_,
        book0_.name as name2_0_,
        book0_.publishing as publishi3_0_
    from
        book book0_
    where
        book0_.name=?
books = [Book(Id=2, name=Java, publishing=null)]
```

# @Query – Native Query

```
@Query(value = "select * from book where name = ?1", nativeQuery = true)
public Book selectBookByName(String name);
```

```
@Test
public void selectBookByName(){
    Book book = repository.selectBookByName("Java");
    System.out.println("books = " + book);
}
```

```
Hibernate:
    select
        *
    from
        book
    where
        name = ?
books = Book(Id=2, name=Java, publishing=null)
```

# @Query – Positional Parameters

```java
@Modifying
@Transactional
@Query(value = "update book set name= ?1 where id = ?2", nativeQuery = true)
public void updateBookByPublishing(String name, Long id);
```

```java
@Test
public void updateBookByPublishing(){
    repository.updateBookByPublishing( name: "java",  id: 2L);
}
```

```
SELECT * FROM public.book
ORDER BY id ASC
```

Data output | Messages | Notifications

| | id [PK] bigint | name character varying (255) | publishing character varying (255) |
|---|---|---|---|
| 1 | 2 | java | [null] |
| 2 | 3 | javascript | [null] |
| 3 | 4 | java | sharecode |
| 4 | 5 | java | sharecode |

# @Query – Named Parameters

```
1 usage
@Query(value = "select * from book b where b.publishing = :publishingName", nativeQuery = true)
public List<Book> selectBookByPublishingParam(@Param("publishingName") String name);
```

```
@Test
public void selectBookByPublishingParam(){
    List<Book> lb = repository.selectBookByPublishingParam( name: "sharecode");
    System.out.println("lb = " + lb);
}
```

```
Hibernate:
    select
        *
    from
        book b
    where
        b.publishing = ?
lb = [Book(Id=4, name=React Native, publishing=sharecode), Book(Id=5, name=ReactJS, publishing=sharecode)]
```

# Q&A