

In **MySQL**, a **one-to-one relationship** is implemented using **foreign keys** with a **unique constraint** to ensure one-to-one mapping between two tables.

---

## ✅ MySQL One-to-One Example

👉 Tables: **person** and **passport**

```
CREATE TABLE person (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(100)  
);  
  
CREATE TABLE passport (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    passport_number VARCHAR(50),  
    person_id INT UNIQUE,                -- UNIQUE to ensure 1-to-1  
    FOREIGN KEY (person_id) REFERENCES person(id)  
);
```

---

## 🔍 Sample MySQL Queries

- Insert into person

```
INSERT INTO person (name) VALUES ('John');
```

-- Insert into passport for that person


```
INSERT INTO passport (passport_number, person_id) VALUES ('P123456', 1);
```

-- Fetch person with passport

```
SELECT p.name, pa.passport_number  
FROM person p-  
JOIN passport pa ON p.id = pa.person_id;
```

Great! Let's build the full flow for **One-to-One Mapping** in **Spring Boot**, including:

1. ✅ Entities (**Person**, **Passport**)
2. ✅ Repositories

3.  REST Controller with POST, GET, and GET with Pagination
  4.  Postman Examples
- 

## Step 1: Entities

### Person.java

```
@Entity
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToOne(mappedBy = "person", cascade = CascadeType.ALL)
    private Passport passport;
}
```

### Passport.java

```
@Entity
public class Passport {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String passportNumber;

    @OneToOne
    @JoinColumn(name = "person_id", unique = true)
    private Person person;
}
```

---

## Step 2: Repositories

```
public interface PersonRepository extends JpaRepository<Person, Long> {}
```

```
public interface PassportRepository extends JpaRepository<Passport, Long> {}
```

---

## ✓ Step 3: REST Controller

### PersonController.java

```
@RestController
@RequestMapping("/api/persons")
public class PersonController {

    @Autowired
    private PersonRepository personRepository;

    @PostMapping
    public Person createPersonWithPassport(@RequestBody Person person) {
        // set both sides of the relationship
        if (person.getPassport() != null) {
            person.getPassport().setPerson(person);
        }
        return personRepository.save(person);
    }

    @GetMapping
    public List<Person> getAllPersons() {
        return personRepository.findAll();
    }

    @GetMapping("/page")
    public Page<Person> getPersonsPaged(@RequestParam int page, @RequestParam int
size) {
        Pageable pageable = PageRequest.of(page, size);
        return personRepository.findAll(pageable);
    }
}
```

---

## ✓ Step 4: Sample JSON for Postman

### ♦ POST /api/persons

Body (raw JSON):

```
{
  "name": "John Doe",
  "passport": {
    "passportNumber": "P123456"
  }
}
```

---

### ♦ GET /api/persons

Returns list of all persons with their passport info.

---

### ♦ GET /api/persons/page?page=0&size=2

Pagination example: returns page 0 with 2 persons.

---

Would you like me to generate the full `application.properties` and Spring Boot starter setup too?

Perfect! Let's build the **One-to-Many** mapping in **Spring Boot with JPA**, including:

1. ☒ Entities: `Customer` (one) → `Orders` (many)
  2. ☒ Repositories
  3. ☒ REST Controller: `POST`, `GET`, `Pagination`
  4. ☒ Postman request examples
- 

## ☒ Step 1: Entities

### `Customer.java`

```
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    private String name;

    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
    private List<Order> orders;

    // getters and setters
}
```

---

## Order.java

```
@Entity
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String product;

    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;

    // getters and setters
}
```

---

## ✓ Step 2: Repositories

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {}
public interface OrderRepository extends JpaRepository<Order, Long> {}
```

---

## ✓ Step 3: REST Controller

### CustomerController.java

```
@RestController
@RequestMapping("/api/customers")
```

```

public class CustomerController {

    @Autowired
    private CustomerRepository customerRepository;

    @PostMapping
    public Customer createCustomer(@RequestBody Customer customer) {
        if (customer.getOrders() != null) {
            for (Order order : customer.getOrders()) {
                order.setCustomer(customer);
            }
        }
        return customerRepository.save(customer);
    }

    @GetMapping
    public List<Customer> getAllCustomers() {
        return customerRepository.findAll();
    }

    @GetMapping("/page")
    public Page<Customer> getCustomersPaged(@RequestParam int page,
    @RequestParam int size) {
        Pageable pageable = PageRequest.of(page, size);
        return customerRepository.findAll(pageable);
    }
}

```

---

## Step 4: Postman Examples

### POST /api/customers

**Body:**

```

{
  "name": "Alice",
  "orders": [
    { "product": "Laptop" },
    { "product": "Mouse" }
  ]
}

```

```
}
```

---

#### ♦ GET /api/customers

Returns all customers with their orders.

---

#### ♦ GET /api/customers/page?page=0&size=2

Returns page 0 with 2 customers and their orders.

---

Would you like the same logic implemented inside the `main` method or a full Spring Boot project with `application.properties`, DTOs, and validations?

Here's how the **One-to-Many relationship** looks in **MySQL** with `Customer` and `Order` tables:

---

## ✓ Step 1: Create Tables in MySQL

#### ♦ customer Table

```
CREATE TABLE customer (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100) NOT NULL  
);
```

---

#### ♦ orders Table (with customer\_id as FK)

```
CREATE TABLE orders (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    product VARCHAR(100) NOT NULL,  
    customer_id INT,  
    FOREIGN KEY (customer_id) REFERENCES customer(id)  
);
```

This allows each customer to have multiple orders, but each order belongs to only one customer.

---

## ✓ Step 2: Sample Data Insertion

```
-- Insert customer
INSERT INTO customer (name) VALUES ('Alice');

-- Insert orders for Alice (id = 1)
INSERT INTO orders (product, customer_id) VALUES ('Laptop', 1);
INSERT INTO orders (product, customer_id) VALUES ('Mouse', 1);
```

---

## ✓ Step 3: Fetch Customers with Their Orders

### ♦ Using JOIN

```
SELECT c.name AS customer_name, o.product
FROM customer c
JOIN orders o ON c.id = o.customer_id;
```

This will return:

customer_name	product
Alice	Laptop
Alice	Mouse

---

Would you like a more advanced query using `GROUP_CONCAT()` to return all orders in one row per customer?

## ✓ Goal

We'll add a method like:

http



CopyEdit

POST /enrollments/add

Request body:

json

CopyEdit

```
{
  "studentName": "Shashank",
  "courseTitle": "Java Bootcamp"
}
```

And it will:

1. Create a new student
2. Create a new course
3. Create an enrollment record between them



## Updated Code

---

### EnrollmentRequestDTO.java (to receive input data)

java

CopyEdit

```
public class EnrollmentRequestDTO {
    private String studentName;
    private String courseTitle;
}
```

### Student.java

java

CopyEdit

@Entity

```
public class Student {
    @Id
```

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

private String name;

@OneToMany(mappedBy = "student", cascade = CascadeType.ALL)
private List<Enrollment> enrollments = new ArrayList<>();
}
```

---

## Course.java

```
@Entity
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @OneToMany(mappedBy = "course", cascade = CascadeType.ALL)
    private List<Enrollment> enrollments = new ArrayList<>();
}
```

---

## Enrollment.java

```
java
CopyEdit
@Entity
public class Enrollment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private LocalDate enrollmentDate;

    @ManyToOne
    @JoinColumn(name = "student_id")
    private Student student;

    @ManyToOne
```

```
        @JoinColumn(name = "course_id")
        private Course course;
    }
}
```

---

## Repositories

```
java
CopyEdit
public interface StudentRepository extends JpaRepository<Student, Long> {}
public interface CourseRepository extends JpaRepository<Course, Long> {}
public interface EnrollmentRepository extends JpaRepository<Enrollment, Long> {}
{}
```

---

## EnrollmentService.java

```
@Service
public class EnrollmentService {

    @Autowired
    private StudentRepository studentRepo;

    @Autowired
    private CourseRepository courseRepo;

    @Autowired
    private EnrollmentRepository enrollmentRepo;

    // ✅ ENROLL existing student and course
    public Enrollment enrollStudentToCourse(Long studentId, Long courseId) {
        Student student = studentRepo.findById(studentId)
            .orElseThrow(() -> new RuntimeException("Student not found"));
        Course course = courseRepo.findById(courseId)
            .orElseThrow(() -> new RuntimeException("Course not found"));

        Enrollment enrollment = new Enrollment();
        enrollment.setStudent(student);
        enrollment.setCourse(course);
        enrollment.setEnrollmentDate(LocalDate.now());

        return enrollmentRepo.save(enrollment);
    }
}
```

```

    }

    // ✅ ENROLL by creating student and course
    public Enrollment createAndEnroll(EnrollmentRequestDTO request) {
        Student student = new Student();
        student.setName(request.getStudentName());
        student = studentRepo.save(student);

        Course course = new Course();
        course.setTitle(request.getCourseTitle());
        course = courseRepo.save(course);

        Enrollment enrollment = new Enrollment();
        enrollment.setStudent(student);
        enrollment.setCourse(course);
        enrollment.setEnrollmentDate(LocalDate.now());

        return enrollmentRepo.save(enrollment);
    }

    public List<Enrollment> getAllEnrollments() {
        return enrollmentRepo.findAll();
    }

    public List<Course> getCoursesForStudent(Long studentId) {
        Student student = studentRepo.findById(studentId)
            .orElseThrow(() -> new RuntimeException("Student not found"));
        return student.getEnrollments().stream()
            .map(Enrollment::getCourse)
            .collect(Collectors.toList());
    }

    public List<Student> getStudentsForCourse(Long courseId) {
        Course course = courseRepo.findById(courseId)
            .orElseThrow(() -> new RuntimeException("Course not found"));
        return course.getEnrollments().stream()
            .map(Enrollment::getStudent)
            .collect(Collectors.toList());
    }
}

```

---

## EnrollmentController.java

```
@RestController
@RequestMapping("/enrollments")
public class EnrollmentController {

    @Autowired
    private EnrollmentService enrollmentService;

    // ✅ Enroll by IDs (existing student & course)
    @PostMapping("/{studentId}/{courseId}")
    public ResponseEntity<Enrollment> enrollByIds(
        @PathVariable Long studentId,
        @PathVariable Long courseId) {
        return
ResponseEntity.ok(enrollmentService.enrollStudentToCourse(studentId,
courseId));
    }

    // ✅ Enroll and create student + course
    @PostMapping("/add")
    public ResponseEntity<Enrollment> createAndEnroll(@RequestBody
EnrollmentRequestDTO request) {
        return ResponseEntity.ok(enrollmentService.createAndEnroll(request));
    }

    @GetMapping
    public ResponseEntity<List<Enrollment>> getAll() {
        return ResponseEntity.ok(enrollmentService.getAllEnrollments());
    }

    @GetMapping("/student/{studentId}")
    public ResponseEntity<List<Course>> getCoursesForStudent(@PathVariable
Long studentId) {
        return
ResponseEntity.ok(enrollmentService.getCoursesForStudent(studentId));
    }

    @GetMapping("/course/{courseId}")
    public ResponseEntity<List<Student>> getStudentsForCourse(@PathVariable
Long courseId) {
```

```
        return  
        ResponseEntity.ok(enrollmentService.getStudentsForCourse(courseId));  
    }  
}
```

---

## ✓ Postman Example: Add Enrollment with New Student + Course

### Request:

```
http  
CopyEdit  
POST /enrollments/add  
Content-Type: application/json  
  
{  
  "studentName": "Shashank",  
  "courseTitle": "Spring Boot Essentials"  
}
```

### Response:

```
json  
CopyEdit  
{  
  "id": 1,  
  "enrollmentDate": "2025-04-30",  
  "student": {  
    "id": 1,  
    "name": "Shashank"  
  },  
  "course": {  
    "id": 1,  
    "title": "Spring Boot Essentials"  
  }  
}
```

---

Would you like to add validations (e.g., prevent duplicate enrollments) or show how to update/delete enrollments too?

Ah! Got it. You want to **manually create MySQL tables and write SQL queries** to implement a **many-to-many relationship using a third (join) table**—without using Java code.

Let's take the example:

**Students ↔ Enrollments ↔ Courses**

---

## ✓ 1. MySQL Table Design

We'll use:

- `student` table
  - `course` table
  - `enrollment` table (join table)
- 

### ♦ `student` table

```
CREATE TABLE student (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100) NOT NULL  
);
```

---

### ♦ `course` table

```
CREATE TABLE course (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    title VARCHAR(100) NOT NULL  
);
```

---

### ♦ `enrollment` table (Join Table)

```
CREATE TABLE enrollment (  
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    student_id INT,  
    course_id INT,  
    enrollment_date DATE,  
    FOREIGN KEY (student_id) REFERENCES student(id),  
    FOREIGN KEY (course_id) REFERENCES course(id)  
);
```

---

## ✓ 2. Insert Sample Data

```
-- Insert students  
INSERT INTO student (name) VALUES ('Shashank'), ('Ravi');  
  
-- Insert courses  
INSERT INTO course (title) VALUES ('Java'), ('Spring Boot');  
  
-- Enroll students  
INSERT INTO enrollment (student_id, course_id, enrollment_date)  
VALUES  
    (1, 1, CURDATE()), -- Shashank → Java  
    (1, 2, CURDATE()), -- Shashank → Spring Boot  
    (2, 1, CURDATE()); -- Ravi → Java
```

---

## ✓ 3. SQL Queries

### ♦ Get all courses for a student

```
SELECT c.title  
FROM course c  
JOIN enrollment e ON c.id = e.course_id  
WHERE e.student_id = 1;
```

### ♦ Get all students in a course

```
SELECT s.name  
FROM student s  
JOIN enrollment e ON s.id = e.student_id  
WHERE e.course_id = 1;
```



♦ **Get all enrollments with student and course**

```
SELECT s.name AS student, c.title AS course,  
e.enrollment_date  
FROM enrollment e  
JOIN student s ON s.id = e.student_id  
JOIN course c ON c.id = e.course_id;
```

---

## Summary

Table	Purpose
<code>student</code>	Stores student records
<code>course</code>	Stores course records
<code>enrollment</code> <code>t</code>	Joins students & courses

The `enrollment` table is the **third (join) table** that forms the **many-to-many** relationship.

---

Would you like me to generate a MySQL script file (`.sql`) for full setup?