# NOTES ON ONE TO MANY RELATION IN DB:

Handling a one-to-many relationship in Hibernate is straightforward and involves using the `@OneToMany` annotation in the parent entity and the `@ManyToOne` annotation in the child entity. Let's create an example of a one-to-many relationship between a "Department" and "Employee" in a company where one department can have multiple employees.

1. Database Tables:

```sql
CREATE TABLE Department (
    department_id INT PRIMARY KEY,
    name VARCHAR(100)
);

CREATE TABLE Employee (
    employee_id INT PRIMARY KEY,
    name VARCHAR(50),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES Department(department_id)
);
```

2. Entity Classes using Hibernate Annotations:

Department Entity:
```java
import javax.persistence.*;
import java.util.List;

@Entity
@Table(name = "departments")
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long departmentId;

    private String name;

    @OneToMany(mappedBy = "department", cascade = CascadeType.ALL)
    private List<Employee> employees;
```

```
    public Department() {
        // Default constructor required by Hibernate
    }

    public Department(String name) {
        this.name = name;
    }

    // Getters, setters, etc.
}
```

Employee Entity:
```java
import javax.persistence.*;

@Entity
@Table(name = "employees")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long employeeId;

    private String name;

    @ManyToOne
    @JoinColumn(name = "department_id")
    private Department department;

    public Employee() {
        // Default constructor required by Hibernate
    }

    public Employee(String name, Department department) {
        this.name = name;
        this.department = department;
    }

    // Getters, setters, etc.
}
```

3. Using Hibernate to Save and Retrieve Data:

```java
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class Main {
    public static void main(String[] args) {
        // Create Hibernate SessionFactory
        SessionFactory sessionFactory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(Department.class)
                .addAnnotatedClass(Employee.class)
                .buildSessionFactory();

        // Create a department and some employees
        Department department = new Department("IT");
        Employee employee1 = new Employee("John", department);
        Employee employee2 = new Employee("Alice", department);

        // Add employees to the department (One-to-Many)
        department.getEmployees().add(employee1);
        department.getEmployees().add(employee2);

        // Save the data into the database
        try (Session session = sessionFactory.getCurrentSession()) {
            Transaction transaction = session.beginTransaction();

            session.save(department);

            transaction.commit();
        }

        // Retrieve data from the database
        try (Session session = sessionFactory.getCurrentSession()) {
            Transaction transaction = session.beginTransaction();

            Department retrievedDepartment = session.get(Department.class,
1L); // Assuming ID 1 exists
```

```
            System.out.println("Retrieved Department: " +
retrievedDepartment.getName());
            System.out.println("Employees: " +
retrievedDepartment.getEmployees());

            transaction.commit();
        }

        // Close the session factory when done
        sessionFactory.close();
    }
}
```

In this example, we have a one-to-many relationship between "Department" and "Employee" entities. The `Department` entity uses the `@OneToMany` annotation to represent the collection of employees, and the `Employee` entity uses the `@ManyToOne` annotation to represent the association with the department. Hibernate handles the relationship automatically when persisting data, and it also supports lazy loading to efficiently manage the relationship between entities.

In the code snippet,the data in the `Department` object and its associated `Employee` objects will be saved to their respective tables (`departments` and `employees`). However, only the data for the department itself will be saved to the `departments` table, not the data for the employees. Instead, the data for the employees will be saved to the `employees` table.

Here's what happens step-by-step:

1. We create a `Department` object and two `Employee` objects (`employee1` and `employee2`).
2. We associate the `Employee` objects with the `Department` by setting their `department` property to the `Department` object.
3. We add the `Employee` objects to the list of employees in the `Department` using `department.getEmployees().add(employee1)` and `department.getEmployees().add(employee2)`.
4. When we save the `Department` object using `session.save(department)`, Hibernate will automatically recognize that the `Department` object has a one-to-many relationship with the `Employee` objects and will save the department data to the `departments` table and the employee data to the

`employees` table. Additionally, the `department_id` will be set in the `employees` table to establish the relationship between the employees and the department.

So, when you retrieve the data from the database later, you will get the department and its associated employees.

Again, I apologize for any confusion. I hope this clears up how the data is saved to the database. If you have any further questions or need additional clarification, please feel free to ask!