Hibernate - One-to-One Mapping

Last Updated : 25 Aug, 2025

One-to-one mapping represents a relationship where a single instance of an entity is associated with a single instance of another entity.

Real-world examples:

- A person has one passport; a passport belongs to one person.
- Leopards have unique spot patterns.
- A college ID is uniquely associated with a student.

In database terms, one-to-one mapping can be of two types:

1. **Unidirectional:** Only one entity knows about the relationship.
2. **Bidirectional:** Both entities are aware of the relationship.

One-to-one unidirectional

In this type of mapping one entity has a property or a column that references to a property or a column in the target entity. Let us see this with the help of an example:

ER Diagram

Step-by-Step Implementation

Step 1: Database Setup

We will use MySQL for this example.

```
DROP SCHEMA IF EXISTS `hb_one_to_one_mapping`;
CREATE SCHEMA `hb_one_to_one_mapping`;
USE `hb_one_to_one_mapping`;
SET FOREIGN_KEY_CHECKS = 0;

-- Student detail table
CREATE TABLE `student_gfg_detail` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `college` VARCHAR(128) DEFAULT NULL,
  `no_of_problems_solved` INT DEFAULT 0,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Student table
CREATE TABLE `student` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(45) DEFAULT NULL,
  `last_name` VARCHAR(45) DEFAULT NULL,
  `email` VARCHAR(45) DEFAULT NULL,
  `student_gfg_detail_id` INT UNIQUE,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`student_gfg_detail_id`) REFERENCES
`student_gfg_detail`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

SET FOREIGN_KEY_CHECKS = 1;
```

Step 2: Hibernate Configuration

Create hibernate.cfg.xml in src/main/resources:

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost:3306/hb_one_to_one_mapping?
```

```
useSSL=false&amp;allowPublicKeyRetrieval=true</property>
        <property name="connection.username">your_username</property>
        <property name="connection.password">your_password</property>
        <property name="connection.pool_size">1</property>
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">true</property>
        <property name="current_session_context_class">thread</property>
    </session-factory>
</hibernate-configuration>
```

Step 3: Maven Dependencies

Add the following dependencies in pom.xml:

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-vibur</artifactId>
        <version>5.6.5.Final</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.28</version>
    </dependency>
</dependencies>
```

Step 4: Define Entity Classes

Now that we have added the related dependencies let's add start with defining our entities.

StudentGfgDetail.java

```
package com.geeksforgeeks.entity;

import javax.persistence.*;

@Entity
@Table(name = "student_gfg_detail")
public class StudentGfgDetail {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;
```

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

    @Column(name = "no_of_problems_solved")
    private int noOfProblemsSolved;

    @OneToOne(mappedBy = "studentGfgDetail", cascade = CascadeType.ALL)
    private Student student;

    public StudentGfgDetail() {}

    public StudentGfgDetail(String college, int noOfProblemsSolved) {
        this.college = college;
        this.noOfProblemsSolved = noOfProblemsSolved;
    }

    // Getters and setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getCollege() { return college; }
    public void setCollege(String college) { this.college = college; }
    public int getNoOfProblemsSolved() { return noOfProblemsSolved; }
    public void setNoOfProblemsSolved(int noOfProblemsSolved) {
this.noOfProblemsSolved = noOfProblemsSolved; }
    public Student getStudent() { return student; }
    public void setStudent(Student student) { this.student = student; }

    @Override
    public String toString() {
        return "StudentGfgDetail{" +
                "id=" + id +
                ", college='" + college + '\'' +
                ", noOfProblemsSolved=" + noOfProblemsSolved +
                '}';
    }
}
```

Student.java

```
package com.geeksforgeeks.entity;

import javax.persistence.*;

@Entity
@Table(name = "student")
public class Student {

    @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;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "student_gfg_detail_id")
    private StudentGfgDetail studentGfgDetail;

    public Student() {}
    public Student(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    // Getters and setters
    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; }
    public StudentGfgDetail getStudentGfgDetail() { return studentGfgDetail;
}
    public void setStudentGfgDetail(StudentGfgDetail studentGfgDetail) {
this.studentGfgDetail = studentGfgDetail; }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", email='" + email + '\'' +
                ", studentGfgDetail=" + studentGfgDetail +
                '}';
    }
}
```

Step 5: Hibernate CRUD Operations

1. Adding Entry (Unidirectional)

```
package com.geeksforgeeks.application;
```

```
package com.geeksforgeeks.geeksforgeeksapplication;

import com.geeksforgeeks.entity.Student;
import com.geeksforgeeks.entity.StudentGfgDetail;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class AddingEntryDemo {

    public static void main(String[] args) {
        SessionFactory factory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(Student.class)
                .addAnnotatedClass(StudentGfgDetail.class)
                .buildSessionFactory();

        try (factory; Session session = factory.getCurrentSession()) {

            Student student = new Student("Vyom", "Yadav",
"vyom@gmail.com");
            StudentGfgDetail studentGfgDetail = new StudentGfgDetail("GFG
College", 20);

            student.setStudentGfgDetail(studentGfgDetail);

            session.beginTransaction();
            session.save(student); // CascadeType.ALL saves both
            session.getTransaction().commit();

            System.out.println("Transaction Successfully Completed!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Explanation:

- We create the Session factory with the help of the configuration file as we had set up the details in the configuration file. Then we add annotated classes, basically classes we annotated in the previous steps and then just build the session factory.
- Get the current session from the session factory.
- Create relevant objects and call setter methods.
- Begin the transaction.
- Save the **Student** object, this also saves the associated **StudentGfgDetail** as we used **CascadeType.ALL** in the previous steps.

- Get the transaction and commit the changes.
- Close the current session.
- Close the session factory as we don't want to create more sessions.

Run and Varify

```
AddingEntryDemo
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java ...
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Feb 12, 2022 2:31:19 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate ORM core version 5.6.5.Final
Feb 12, 2022 2:31:20 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
INFO: HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Feb 12, 2022 2:31:21 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
Hibernate: insert into student_gfg_detail (college, no_of_problems_solved) values (?, ?)
Hibernate: insert into student (email, first_name, last_name, student_gfg_detail_id) values (?, ?, ?, ?)
Transaction Successfully Completed!

Process finished with exit code 0
```

Output

Let's verify whether these values have been successfully inserted into the database.

#	id	first_name	last_name	email	student_gfg_detail_id
1	1	Vyom	Yadav	vyom@gmail.com	1
*	NULL	NULL	NULL	NULL	NULL

student Table

#	id	college	no_of_problems_solved
1	1	GFG College	20
*	NULL	NULL	NULL

student_gfg_detail Table

Updating Entry

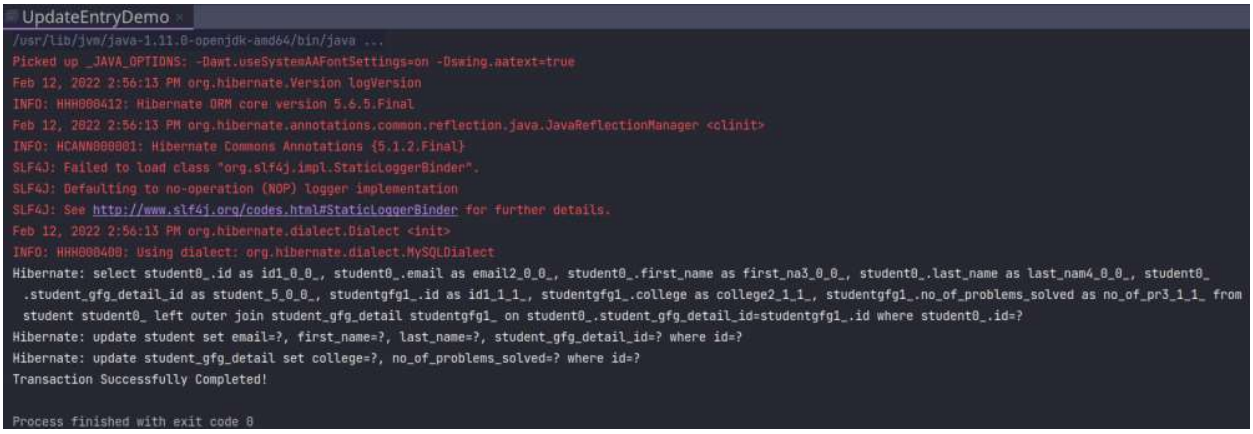
Now that we have verified that we were able to add an entry to the database and our operations were working let's try to update these

entries.

```
package com.geeksforgeeks.application;

import com.geeksforgeeks.entity.Student;
import com.geeksforgeeks.entity.StudentGfgDetail;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class UpdateEntryDemo {

    public static void main(String[] args) {
        SessionFactory factory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(Student.class)
                .addAnnotatedClass(StudentGfgDetail.class)
                .buildSessionFactory();

        try (factory; Session session = factory.getCurrentSession()) {
            session.beginTransaction();

            Student student = session.get(Student.class, 1);
            StudentGfgDetail detail = student.getStudentGfgDetail();

            student.setEmail("vyom@geeksforgeeks.com");
            detail.setNoOfProblemsSolved(40);

            session.update(student); // CascadeType.ALL updates both
            session.getTransaction().commit();

            System.out.println("Transaction Successfully Completed!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Run and Varify

Run UpdateEntryDemo class and varify the output with database



We can also verify these changes inside the database

#	id	first_name	last_name	email	student_gfg_detail_id
1	1	Vyom	Yadav	vyom@geeksforgeeks.com	1
*	NULL	NULL	NULL	NULL	NULL

Note that email changed

#	id	college	no_of_problems_solved
1	1	GFG College	40
*	NULL	NULL	NULL

Note that the no_of_problems_solved changed

3. Reading Entry

```
// Inside main transaction
Student student = session.get(Student.class, 1);
StudentGfgDetail detail = student.getStudentGfgDetail();
System.out.println(student);
System.out.println(detail);
session.getTransaction().commit();
```

4. Deleting Entry

```
Student student = session.get(Student.class, 1);
session.delete(student); // CascadeType.ALL deletes both
session.getTransaction().commit();
```

Bidirectional One-to-One Mapping

A unidirectional relationship (Student ->StudentGfgDetail) doesn't allow accessing the Student from StudentGfgDetail. This can cause dangling foreign keys if a detail is deleted. Make the relationship bidirectional using

mappedBy in StudentGfgDetail, allowing safe access both ways without changing the database structure.

Problem

> *If we delete a StudentGfgDetail entity and leave the Student entity as it is, the Student entity will have a foreign key referring to a non-existing object, which causes a dangling foreign key, a bad practice.*

Depending on the database design, we might want to:

- Keep the Student as a record of users who left the community
- Or delete it along with the StudentGfgDetail

With Hibernate, we can handle both situations without modifying the database.

Step 1: StudentGfgDetail Entity

```
package com.geeksforgeeks.entity;

import javax.persistence.*;

@Entity
@Table(name = "student_gfg_detail")
public class StudentGfgDetail {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

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

    @Column(name = "no_of_problems_solved")
    private int noOfProblemsSolved;

    @OneToOne(mappedBy = "studentGfgDetail", cascade = CascadeType.ALL)
    private Student student;

    public StudentGfgDetail() {}

    public StudentGfgDetail(String college, int noOfProblemsSolved) {
        this.college = college;
```

```
        this.noOfProblemsSolved = noOfProblemsSolved;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getCollege() { return college; }
    public void setCollege(String college) { this.college = college; }

    public int getNoOfProblemsSolved() { return noOfProblemsSolved; }
    public void setNoOfProblemsSolved(int noOfProblemsSolved) {
this.noOfProblemsSolved = noOfProblemsSolved; }

    public Student getStudent() { return student; }
    public void setStudent(Student student) { this.student = student; }

    @Override
    public String toString() {
        return "StudentGfgDetail{" +
                "id=" + id +
                ", college='" + college + '\'' +
                ", noOfProblemsSolved=" + noOfProblemsSolved +
                ", student=" + student +
                '}';
    }
}
```

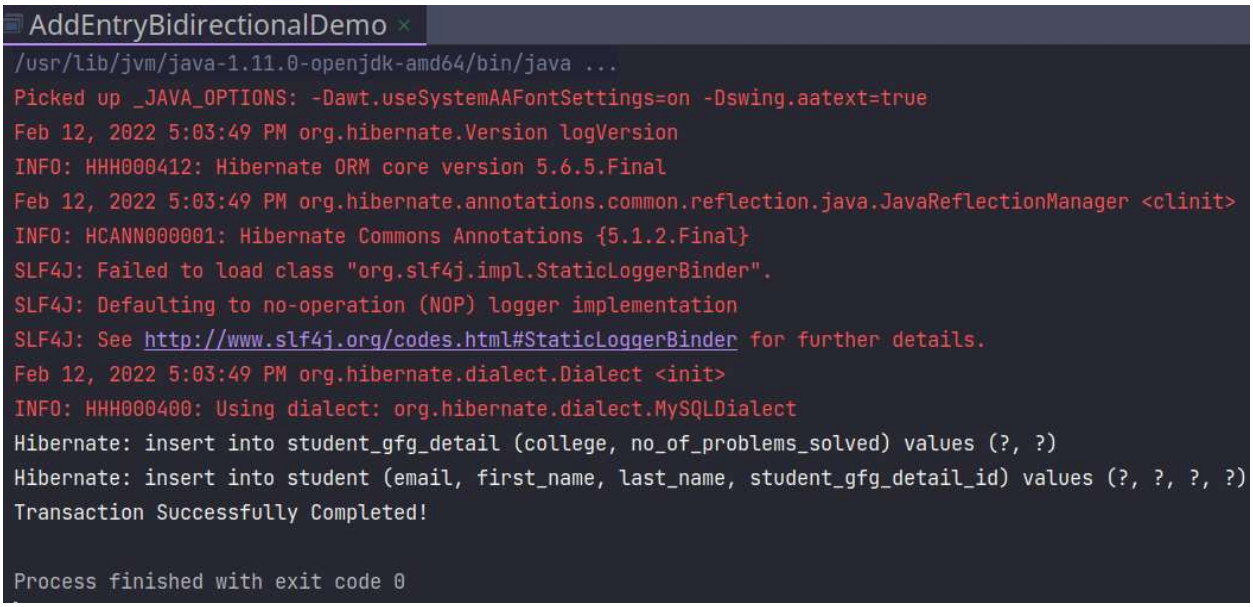
Step 2: Add Entry

```
package com.geeksforgeeks.application;

import com.geeksforgeeks.entity.Student;
import com.geeksforgeeks.entity.StudentGfgDetail;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class AddEntryBidirectionalDemo {

    public static void main(String[] args) {

        SessionFactory factory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(Student.class)
                .addAnnotatedClass(StudentGfgDetail.class)
                .buildSessionFactory();

        try (factory; Session session = factory.getCurrentSession()) {

            Student student = new Student("JJ", "Olatunji", "jj@gmail.com");
            StudentGfgDetail studentGfgDetail = new StudentGfgDetail("GFG
College", 0);
```

```
            student.setStudentGfgDetail(studentGfgDetail);
            studentGfgDetail.setStudent(student);

            session.beginTransaction();
            session.save(studentGfgDetail); // Saves both StudentGfgDetail
and Student
            session.getTransaction().commit();

            System.out.println("Transaction Successfully Completed!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Run and Varify

Note that we don't have any key referencing to the **student** in the **student_gfg_detail** table still, we can save it through the **StudentGfgDetail** object.



Step 3: Read Entry

```
package com.geeksforgeeks.application;

import com.geeksforgeeks.entity.StudentGfgDetail;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
```

```
public class ReadEntryBidirectionalDemo {

    public static void main(String[] args) {

        SessionFactory factory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(com.geeksforgeeks.entity.Student.class)

.addAnnotatedClass(com.geeksforgeeks.entity.StudentGfgDetail.class)
                .buildSessionFactory();

        try (factory; Session session = factory.getCurrentSession()) {

            session.beginTransaction();

            int theId = 5; // Change according to your DB
            StudentGfgDetail studentGfgDetail =
session.get(StudentGfgDetail.class, theId);

            System.out.println(studentGfgDetail.getStudent());
            System.out.println(studentGfgDetail);

            session.getTransaction().commit();

            System.out.println("Transaction Successfully Completed!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Run and Varify

As we can see, we just retrieved the **StudentGfgDetail** object and that object in turn implicitly retrieved the **Student** object.