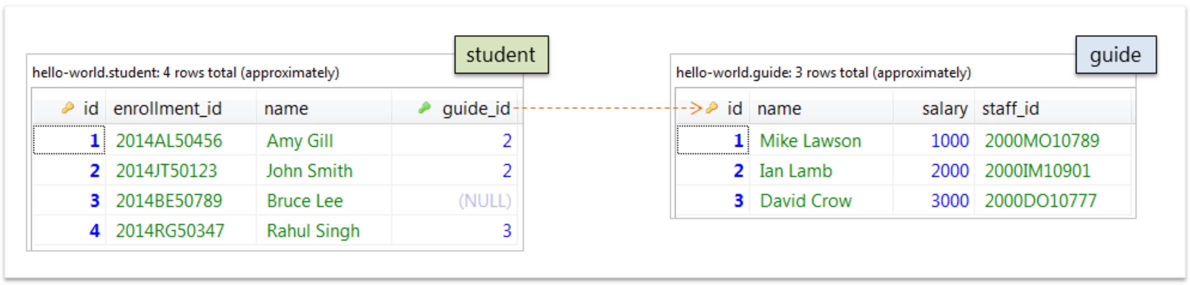
Task 1: As explained in the lecture on Merging Detached Objects, simulate a user's interaction with an application in which the user is doing the following:

1) Loading Guide with id=2
2) Modifying the salary of the Guide[id=2] to 2500 and the name of its associated Student[id=1] to Amy Jade Gill
3) Persist the changes made to the Guide[id=2] and Student Student[id=1] to hello-world database

Implement the given user interaction using detached objects and extended persistence context.

Consider the following to be the starting state of the student and guide tables for each implementation.



student

hello-world.student: 4 rows total (approximately)

id	enrollment_id	name	guide_id
1	2014AL50456	Amy Gill	2
2	2014JT50123	John Smith	2
3	2014BE50789	Bruce Lee	(NULL)
4	2014RG50347	Rahul Singh	3

guide

hello-world.guide: 3 rows total (approximately)

id	name	salary	staff_id
1	Mike Lawson	1000	2000MO10789
2	Ian Lamb	2000	2000IM10901
3	David Crow	3000	2000DO10777

Task 2: In the lecture, we used CascadeType.*MERGE* to merge the detached Guide, which merged not just the detached Guide but also the Student objects associated with it. Do you think the same could have been done using the CascadeType.*PERSIST* instead?

Task 3: If the line int numOfStudents = students.size(); is removed from the HelloWorldClient to initialize the students collection proxy, what are the exceptions it would cause to throw? If the answer is none, please explain why?

The source code files for the lecture on "Merging Detached Objects" are available to be downloaded with this lab exercise. You could use them to complete the given tasks successfully.

You could also look at the source code inside all the downloadable files below.

Source Code: Merging Detached Objects

Schema for hello-world database

```sql
CREATE DATABASE `hello-world`;
```

persistence.xml

(to be placed inside META-INF folder; the META-INF folder should be in the root of the classpath of your application)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
   version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">

   <persistence-unit name="hello-world" transaction-type="RESOURCE_LOCAL">
      <properties>

         <!-- Database connection settings -->
         <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
         <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/hello-
world" />
         <property name="javax.persistence.jdbc.user" value="root" />
         <property name="javax.persistence.jdbc.password" value="password" />

         <!-- SQL dialect -->
         <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />

         <!-- Create/update tables automatically using mapping metadata -->
         <property name="hibernate.hbm2ddl.auto" value="update" />

         <!-- Pretty print the SQL in the log file and console -->
         <property name="hibernate.format_sql" value="true" />
      </properties>

   </persistence-unit>
</persistence>
```

log4j.properties

```
# Direct to file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=hello-world.log
log4j.appender.file.MaxFileSize=2MB
log4j.appender.file.MaxBackupIndex=1
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

# Direct to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

# Root logger option
log4j.rootLogger=OFF, stdout, file

#Log everything (this will also include the logging information configured by
"log4j.logger.org.hibernate.SQL=ALL" and
"log4j.logger.org.hibernate.type.descriptor.sql.BasicBinder=TRACE")
#log4j.logger.org.hibernate=ALL

# Show SQL statements
log4j.logger.org.hibernate.SQL=ALL

# Show the bind parameter values
log4j.logger.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

HelloWorldClient.java

```
package client;

import java.util.Set;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import entity.Guide;
import entity.Student;

public class HelloWorldClient {
   public static void main(String[] args) {

      EntityManagerFactory emf = Persistence.createEntityManagerFactory("hello-world");

      //==============================
      //Detached Objects
      //==============================
      EntityManager em1 = emf.createEntityManager();
      em1.getTransaction().begin();

      Guide guide = em1.find(Guide.class, 2L);
      Set<Student> students = guide.getStudents();
      int numOfStudents = students.size();

      Student student = null;
      for(Student nextStudent: students) {
         if(nextStudent.getId() == 2L) {
            student = nextStudent;
         }
      }

      em1.getTransaction().commit();
      em1.close();

      guide.setSalary(2500);
      student.setName("Amy Jade Gill");

      EntityManager em2 = emf.createEntityManager();
      em2.getTransaction().begin();

      @SuppressWarnings("unused")
      Guide mergedGuide = em2.merge(guide);
```

```
        em2.getTransaction().commit();
        em2.close();

        //==============================
        //Extended Persistence Context
        //==============================
        /*
        EntityManager em = emf.createEntityManager();

        em.getTransaction().begin();

        Guide guide = em.find(Guide.class, 2L);
        Set<Student> students = guide.getStudents();
        int numOfStudents = students.size();

        Student student = null;
        for(Student nextStudent: students) {
            if(nextStudent.getId() == 1L) {
                student = nextStudent;
            }
        }

        em.getTransaction().commit();

        guide.setSalary(2500);
        student.setName("Amy Jade Gill");

        em.getTransaction().begin();

        //merging not needed

        em.getTransaction().commit();

        em.close();
        */
    }
}
```

Guide.java

```java
package entity;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

@Entity
public class Guide {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(name = "staff_id", nullable = false)
    private String staffId;

    private String name;
    private Integer salary;

    @OneToMany(mappedBy = "guide")
    private Set<Student> students = new HashSet<Student>();

    public Guide() { }
    public Guide(String staffId, String name, Integer salary) {
        this.staffId = staffId;
        this.name = name;
        this.salary = salary;
    }
    public void addStudent(Student student) {
        students.add(student);
        student.setGuide(this);
    }
    public void setSalary(Integer salary) {
        this.salary = salary;
    }
    public Set<Student> getStudents() {
        return students;
    }
    public String toString() {
        return "Guide [id=" + id + ", staffId=" + staffId + ", name=" + name
                + ", salary=" + salary + ", students=" + students + "]";
    }

}
```

Student.java

```java
package entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

import org.apache.commons.lang3.builder.EqualsBuilder;
import org.apache.commons.lang3.builder.HashCodeBuilder;

@Entity
public class Student {

   @Id @GeneratedValue(strategy=GenerationType.AUTO)
   private Long id;

   @Column(name="enrollment_id", nullable=false)
   private String enrollmentId;

   private String name;

   @ManyToOne(fetch=FetchType.LAZY)
   @JoinColumn(name="guide_id")
   private Guide guide;

   public Student() {}
   public Student(String enrollmentId, String name) {
      this.enrollmentId = enrollmentId;   this.name = name;
   }

   public void setGuide(Guide guide) {
      this.guide = guide;
   }
   public void setName(String name) {
      this.name = name;
   }
   public Long getId() {
      return id;
   }
```

```
    @Override
    public int hashCode() {
        return new HashCodeBuilder().append(enrollmentId).toHashCode();
    }

    @Override
    public boolean equals(Object obj) {
        if(!(obj instanceof Student)) return false;
        Student other = (Student) obj;
        return new EqualsBuilder().append(enrollmentId, other.enrollmentId).isEquals();
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", enrollmentId=" + enrollmentId
                + ", name=" + name + ", guide=" + guide + "]";
    }

}
```