Hibernate (Operations on Records)

Operations on database records

- CRUD operations
  - Save (Create)
  - Select (Read)
  - Update (Update)
  - Delete (Delete)

Student class

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

    @Id
    @GeneratedValue
    private int student_id;
    private String student_name;

    public int getStudent_id() {
        return student_id;
    }
```

```
    public String getStudent_name()
     {
        return student_name;
    }

    public void
     setStudent_name(String
     student_name) {
        this.student_name =
     student_name;
    }
}
```

Main class

```
public class Main {
    public static void main(String[] args) {
     SessionFactory sessionFactory= new
    AnnotationConfiguration().configure().buildSessionFactory();
     Session session=sessionFactory.openSession();
     session.beginTransaction();
    // Perform operations
     session.getTransaction().commit();
     session.close();
     sessionFactory.close();
  }
}
```

Add show_sql property in hibernate configuration file

- We can use the following property to see what queries are fired on database by hibernate for various operations

```
<property name="hibernate.show_sql">true</property>
```

Save/Insert operation

- Add following code in Main class

```
Student student = new Student();
student.setStudent_name("Kamal");

SessionFactory sessionFactory= new
AnnotationConfiguration().configure().buildSessionFactory();
Session session=sessionFactory.openSession();
session.beginTransaction();
// Perform operations
session.save(student);

session.getTransaction().commit();
```

Table structure and records

localhost » ddu » student

Browse | Structure | SQL | Search | Insert | Export | Import

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	student_id	int(11)			No	None	AUTO_INCREMENT
2	student_name	varchar(255)	latin1_swedish_ci		Yes	NULL	

```
SELECT *
FROM `student`
LIMIT 0 , 30
```

Show : Start row: 0 Number of rows: 30 Headers every 100 rows

+ Options

←T→	student_id	student_name
Edit Copy Delete	1	Kamal

Select operation

- Code in Main class

```
Student student;
...
 session.beginTransaction();
 // Perform operations
student=(Student) session.get(Student.class, 1);
System.out.println("Student ID="+student.getStudent_id()+"
    Name="+student.getStudent_name());
session.getTransaction().commit();
...
```

- Two arguments to get() method are as follows
  - Student.class: name of Entity class
  - 1: Primary key id
- O/P on console

```
Student ID=1 Name=Kamal
```

But if there is no record for indicated key value

- get() may return null value

```
 // Perform operations
student=(Student) session.get(Student.class, 10);
If(student!=null){
    System.out.println("Student ID="+student.getStudent_id()+"
    Name="+student.getStudent_name());
}
```

Update existing record

- Code in Main class

```
//Perform operation
student=(Student) session.get(Student.class, 1);
student.setStudent_name("Kamalkumar");
session.update(student);
```

- Updated record in database (name, Kamal -> Kamalkumar)

```
SELECT *
FROM `student`
LIMIT 0 , 30
```

Show : Start row: 0 Number of rows: 30 Headers every 100 rows

+ Options

←T→	student_id	student_name
Edit Copy Delete	1	Kamalkumar

Delete an existing record

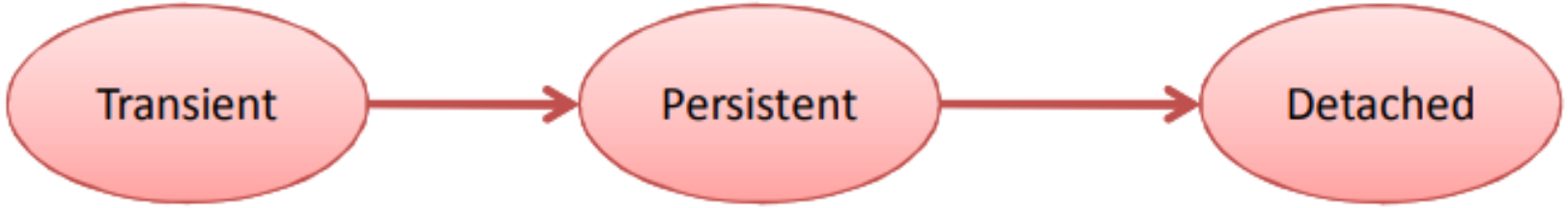
- Code in Main class

```
// Perform operations
student=(Student) session.get(Student.class, 1);
session.delete(student);
```

Hibernate (Object State)

Hibernate objects

- Object state
  - Transient
  - Persistent
  - Detached



Saving object multiple times

- Executing code multiple times by running the program multiple times?
  - E.g., session.save(student);
- E.g., if we run two times, we get following

```
SELECT *
FROM `student`
LIMIT 0 , 30
```

Show : Start row: 0 Number of rows: 30 Headers every 100 rows

Sort by key: None

+ Options

←T→	student_id	student_name
Edit Copy Delete	2	Kamal
Edit Copy Delete	3	Kamal

Modification in object after save

- If we have following code

```
student.setStudent_name("Kamalkumar");
session.save(student);
student.setStudent_name("Kamalkumar Kumar");
```

- Records in the table

+ Options

			student_id	student_name
Edit	Copy	Delete	2	Kamal
Edit	Copy	Delete	3	Kamal
Edit	Copy	Delete	4	Kamalkumar Kumar

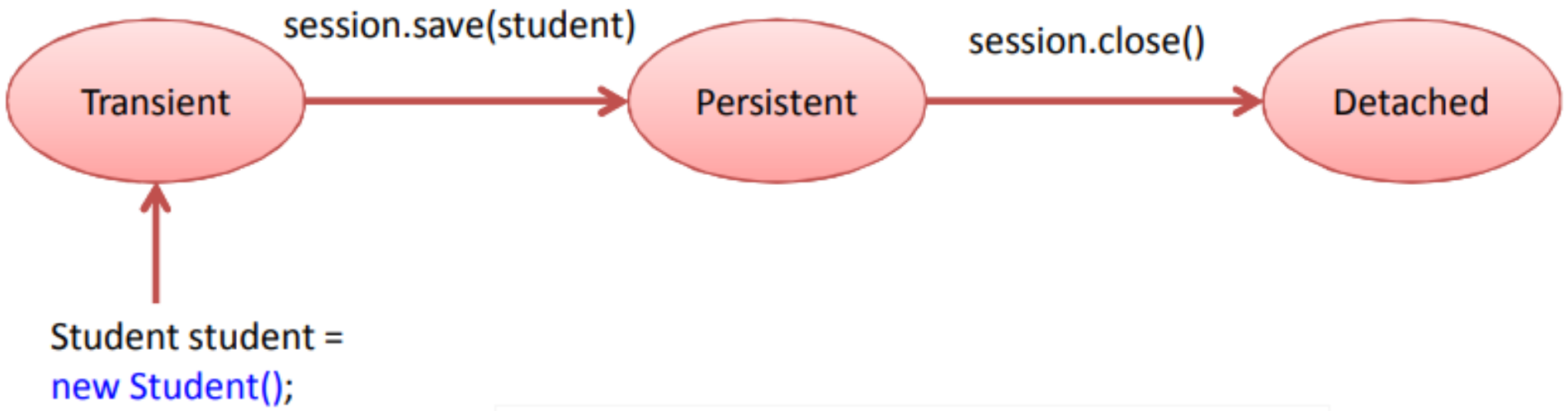
New record, but hibernate automatically modified name of student

State transition

- From creating a student object (using new) until saving it (using session.save()), the object remains in transient state
- After writing the object until we close the session, the object's state becomes persistent state.
  - If object is in persistent state, and if we do any modification in the object, the modifications will be reflected into the database by hibernate
- After session is closed, the object goes into detached state
  - I.e., no effect on db record for any change after session.close()

```
session.close();
sessionFactory.close();
student.setStudent_name("Kamal Kumar");
```

State transition for a new object

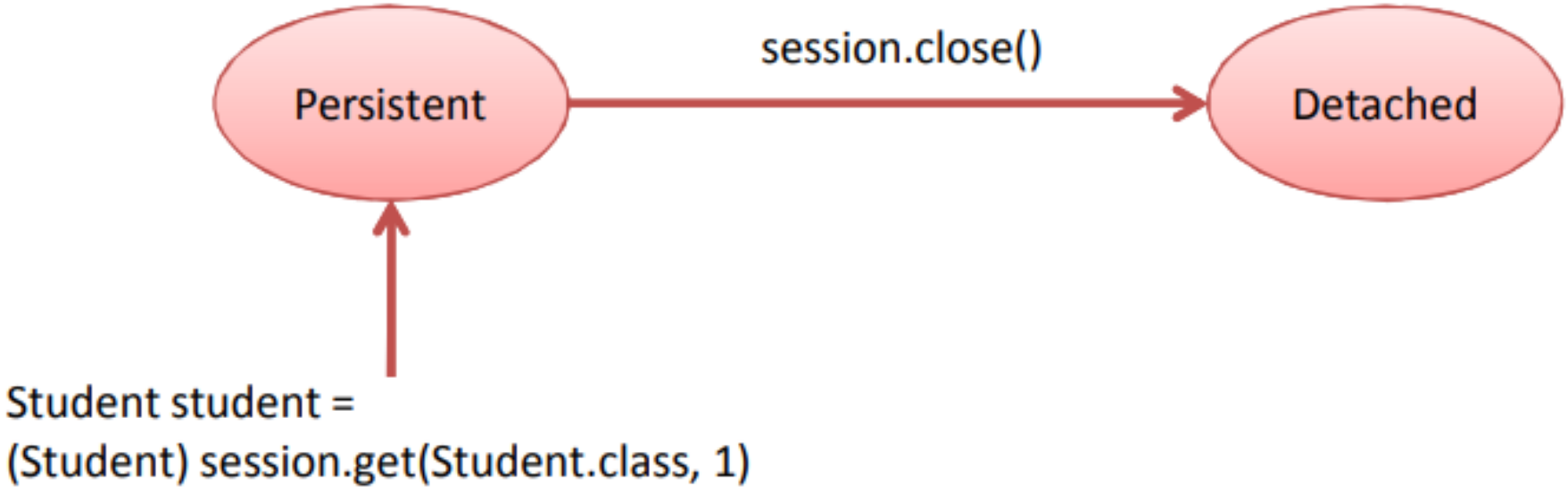


State transition

- Transient
  - Object has no corresponding record in the database
- Persistent
  - Object has corresponding record in the database and the object is associated with hibernate session
  - Any modifications done to object, will be reflected into database, when the session is closed
- Detached
  - Object has corresponding record in the database, but the object is not associated with any hibernate session.

State transition for a retrieved object

- The student object directly goes in persistent state.



Modification to Retrieved object

- The state of a retrieved object is also persistent
- Therefore, the following code will also do modification in the database

Class of Entity object Key value

```
...
session.beginTransaction();
Student student=(Student)session.get(Student.class, 2);
student.setStudent_name("Radha");
session.getTransaction().commit();
session.close();
```

Modification to Retrieved object

- Before modification

+ Options

				student_id	student_name
Edit	Copy	Delete		2	Kamal
Edit	Copy	Delete		3	Kamal
Edit	Copy	Delete		4	Kamalkumar Kumar
Edit	Copy	Delete		5	Kamalkumar Kumar

- After modification

+ Options

				student_id	student_name
Edit	Copy	Delete		2	Radha
Edit	Copy	Delete		3	Kamal
Edit	Copy	Delete		4	Kamalkumar Kumar
Edit	Copy	Delete		5	Kamalkumar Kumar

Deleting persistent object

- If a object in persistent state is deleted, the state of the object changes from persistent to transient.
  - Code to delete an object

```
session.delete(student);
```

State transition for a deleted object

Convert an object from detached state to persistent state

- After session is closed, the object goes to detached state; therefore, any modification will not be reflected to the database.
- Why to create a new session?
  - If there is a long gap between data (record/object) is retrieved and data (record/object) is update.
  - E.g., data update using a webpage
- Solution
  - Open a new session
  - Update the modified object

State transition: detached state to persistent state

Write object that is outside session

- Using the following code, we can write back the detached object:

```
...
Student student=(Student)session.get(Student.class, 2);
session.getTransaction().commit();
session.close();

student.setStudent_name("Radhika");
Session session2=sessionFactory.openSession();
session2.beginTransaction();
session2.update(student);
session2.getTransaction().commit();
...
```

LECT
Activ

Database records

- Get object in first session

+ Options

←T→	student_id	student_name
Edit Copy Delete	2	Radha
Edit Copy Delete	3	Kamal
Edit Copy Delete	4	Kamalkumar Kumar
Edit Copy Delete	5	Kamalkumar Kumar

- Write object in second session

+ Options

←T→	student_id	student_name
Edit Copy Delete	2	Radhika
Edit Copy Delete	3	Kamal
Edit Copy Delete	4	Kamalkumar Kumar
Edit Copy Delete	5	Kamalkumar Kumar