DAY 3
Problems on Hibernate

Now that we understood the basics of Hibernate, let us try some scenario-based questions to deepen our understanding of CRUD operations.

Scenario 1:

Fetch one row from the table

Consider the below table,

Student Table:

roll	name	Mail id
1	alex	alex@gmail.com
2	bob	bob@gmail.com

- To retrieve data from the database there is get() in hibernate.
- The get() accepts two parameters-
  The 1st parameter is the class name from which object the data to be fetched & the id(primary key) is the 2nd parameter.

Logic to fetch the data from the database:

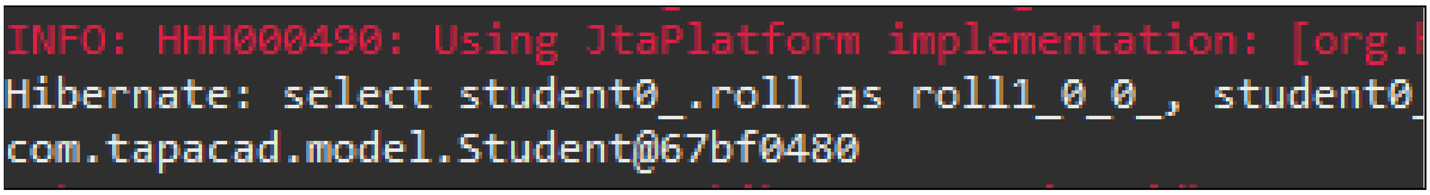
```
Student s = session.get(Student.class, 1); // Here get method
is helping to fetch the Student object with primary key 1, in
this case it is alex object
System.out.println(s); // Here alex student object with the
details is printed
```

Program2.java

```
package com.tapacad.application;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.tapacad.model.Student;
```

```java
public class Program2 {

	public static void main(String[] args) {
		SessionFactory sessionFactory = null;
		Session session = null;
		try {
//			Create Session Factory
			sessionFactory = new Configuration()
					.configure()
					.addAnnotatedClass(Student.class)
					.buildSessionFactory();
//			Create Session
			session = sessionFactory.openSession();

//			Create Transaction
			Transaction transaction = session.beginTransaction();

//			CRUD Operations
			Student s = session.get(Student.class, 1);
			System.out.println(s);

			transaction.commit();
		} finally {
//			Closing Resources
			session.close();
			sessionFactory.close();
		}

	}
}
```

Output:



```
INFO: HHH000490: Using JtaPlatform implementation: [org.
Hibernate: select student0_.roll as roll1_0_0_, student0_
com.tapacad.model.Student@67bf0480
```

As we can see from the code, using session.get() we can fetch one row from the table. Whenever we try to fetch the data which is not present in the database that time it will return null.

Scenario 2:

Fetch all rows from the table

Consider the below table,

roll	name	Mail id
1	alex	alex@gmail.com
2	bob	bob@gmail.com

- To fetch all the data present in the table it is required to make use of **Hibernate Query Language (HQL)** which is similar to SQL query language.

HQL syntax to fetch all data from the table

```
String hql = "FROM Student"; // Here you need to write the query that you want to perform operation
Query query = session.createQuery(hql); // A Query is an Object Orientation representation of Hibernate query. We can get the Query instance, by calling the session.createQuery();
List results = query.list(); // query.list() will return the query results
```

Logic:

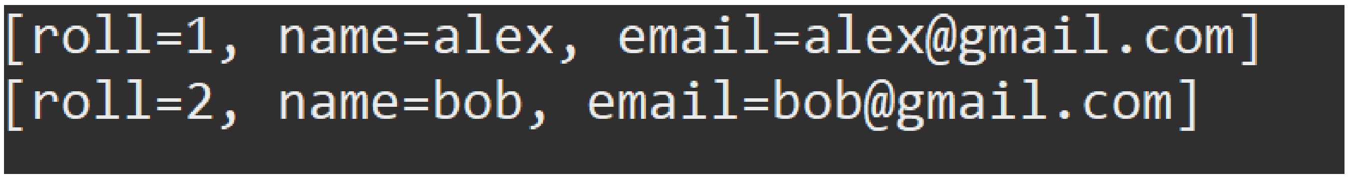
```
String hql = "From Student"; //Here first we need to tell from which class we are fetching the data
Query createQuery = session.createQuery(hql); //createQuery() method generate query object
List students = createQuery.list(); //This will help you to fetch all Student objects
```

Program3.java

```
package com.tap.app;
import java.io.Serializable;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;
import com.tap.model.Student;

public class Program1 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Configuration config = new Configuration();

		config.configure();

		config.addAnnotatedClass(Student.class);

		SessionFactory sessionFactory = config.buildSessionFactory();

		Session session = sessionFactory.openSession();

		Transaction transaction = session.beginTransaction();

//		Logics
		String hql = "From Student";
		Query createQuery = session.createQuery(hql);
		List students = createQuery.list();
		display(students);

		transaction.commit();

	}

	private static void display(List students) {
		for (Object student : students) {
```

```
                              System.out.println(student);
                    }

          }

}
```

Output:



```
[roll=1, name=alex, email=alex@gmail.com]
[roll=2, name=bob, email=bob@gmail.com]
```

As you can see from the above output, using hql query you can fetch all the data present in the table and as overriding of toString() is done you are getting roll number, name, email id.

Scenario 3:

Fetch all rows from the table where the **marks of the student is greater than 90 or the name of the student starts with c**

Consider the below table,

roll	name	email	marks
1	bob	bob@gmail.com	97
3	charli	charli@gmail.com	40
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	92

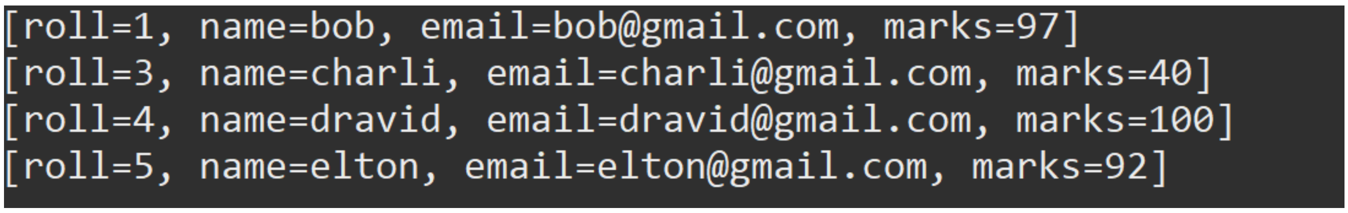
Logic:

```
String hql = "From Student s where s.marks > 90 or s.name LIKE
'c%' "; //Here query is written to fetch all the data whose
marks is greater than 90 or name starting with c
Query createQuery = session.createQuery(hql); // Here query
object is created
List students = createQuery.list(); // This will give list of
object who matches the criteria
```

Program4.java

```
package com.tap.app;
import java.io.Serializable;
import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;
import com.tap.model.Student;

public class Program1 {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Configuration config = new Configuration();

		config.configure();

		config.addAnnotatedClass(Student.class);

		SessionFactory sessionFactory = config.buildSessionFactory();

		Session session = sessionFactory.openSession();

		Transaction transaction = session.beginTransaction();

//		Logics
		String hql = "From Student s where s.marks > 90 or s.name LIKE 'c%' ";
		Query createQuery = session.createQuery(hql);
		List students = createQuery.list();

		display(students);

		transaction.commit();

	}

	private static void display(List students) {
		for (Object student : students) {
			System.out.println(student);
		}
	}

}
```

Output:



```
[roll=1, name=bob, email=bob@gmail.com, marks=97]
[roll=3, name=charli, email=charli@gmail.com, marks=40]
[roll=4, name=dravid, email=dravid@gmail.com, marks=100]
[roll=5, name=elton, email=elton@gmail.com, marks=92]
```

As you can see from the above these are the students who have **marks greater than 90 or name starts with c**

Scenario 4:

Given a table update the marks of the **particular student**
Consider the below table,

roll	name	email	marks
1	bob	bob@gmail.com	97
3	charli	charli@gmail.com	40
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	92

Logic:

```
Student student = session.get(Student.class, 1); // Here
student object we are getting whose marks we want to update or
change
student.setMarks(25); // update the marks of the student object
using setter
```

Program5.java

```
package com.tap.app;

import java.io.Serializable;
import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

import com.tap.model.Student;

public class Program1 {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Configuration config = new Configuration();

                config.configure();

                config.addAnnotatedClass(Student.class);

                SessionFactory sessionFactory = config.buildSessionFactory();

                Session session = sessionFactory.openSession();

                Transaction transaction = session.beginTransaction();

//              Logics
                Student student = session.get(Student.class, 1);
                student.setMarks(25);

                transaction.commit();

        }

        private static void display(List students) {
                for (Object student : students) {
                        System.out.println(student);
                }

        }

}
```

Output:

roll	**name**	**email**	**marks**
1	bob	bob@gmail.com	25
3	charli	charli@gmail.com	40
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	92

If you can see from the above table, bob marks have been updated.

One more way of updating marks of the student is by making use of update()

```
Student student = new Student(6, "zakir", "zakir@gmail.com",
75); // Here you are creating student object whose marks or
whose data you want to change but there should be match in
primary key at least if there no match in that it will give you
exception.

session.update(student); // using update() you are updating
student object
```

Program6.java

```
package com.tap.app;

import java.io.Serializable;
import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

import com.tap.model.Student;

public class Program1 {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Configuration config = new Configuration();

                config.configure();

                config.addAnnotatedClass(Student.class);

                SessionFactory sessionFactory = config.buildSessionFactory();
```

```java
                Session session = sessionFactory.openSession();

                Transaction transaction = session.beginTransaction();

//              Logics
//              Student student = session.get(Student.class, 1);
//              student.setMarks(25);

                Student student = new Student(6, "zakir", "zakir@gmail.com", 75);
                session.update(student);

                transaction.commit();

        }

        private static void display(List students) {
                for (Object student : students) {
                        System.out.println(student);
                }

        }

}
```

The above code gives an exception because when you are updating the value there should be a match in the primary key of the table.

Scenario 5:

Given the student object, update the data of the student if the student exists in the table. If student is new data insert the student into the table

Logic:

```
Student student = new Student(6, "zakir", "zakir@gmail.com", 75); // Create a object that you want to insert or update in database

session.saveOrUpdate(student); // Here using saveOrUpdate() if student object exists it will update the data else it will insert the new object
```

Program7.java

```
package com.tap.app;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.tap.model.Student;

public class Program1 {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Configuration config = new Configuration();

                config.configure();

                config.addAnnotatedClass(Student.class);

                SessionFactory sessionFactory = config.buildSessionFactory();

                Session session = sessionFactory.openSession();
                Transaction transaction = session.beginTransaction();

//              Logics

                Student student = new Student(6, "zakir", "zakir@gmail.com", 75);
                session.saveOrUpdate(student);

                transaction.commit();

        }

        private static void display(List students) {
                for (Object student : students) {
                        System.out.println(student);
                }
        }

}
```

Output:

```
Hibernate: insert into student (email, marks, name, roll) values (?, ?, ?, ?)
```

Data base

roll	name	email	marks
1	bob	bob@gmail.com	25
3	charli	charli@gmail.com	40
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	92
6	zakir	zakir@gmail.com	75

If you observe the table zakir data was not there so using **saveOrUpdate()** you can update the data if the user exists else insert new data if the user doesn’t exist

Now we will try to update the **marks of the elton by making use of saveOrUpdate()**

Program8.java

```
package com.tap.app;

import java.util.List;

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

import com.tap.model.Student;

public class Program1 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Configuration config = new Configuration();

		config.configure();

		config.addAnnotatedClass(Student.class);

		SessionFactory sessionFactory = config.buildSessionFactory();

		Session session = sessionFactory.openSession();

		Transaction transaction = session.beginTransaction();

//		Logics
//		Student student = session.get(Student.class, 1);
//		student.setMarks(25);

		Student student = new Student(5, "elton", "elton@gmail.com", 75);
		session.saveOrUpdate(student);

		transaction.commit();

	}
```

```
        private static void display(List students) {
                for (Object student : students) {
                        System.out.println(student);
                }

        }

}
```

Data base

roll	name	email	marks
1	bob	bob@gmail.com	25
3	charli	charli@gmail.com	40
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	75
6	zakir	zakir@gmail.com	75

Now, if you observe the above table elton marks has been updated with 75

Scenario 6:

Update the marks of all the students to 50 whose marks is less than 50

HQL Syntax to update multiple rows is shown below:

```
String hql = "UPDATE Student s SET s.marks = 50 WHERE s.marks < 50";//Here query is written to update all the marks of the student whose marks is less than 50

Query query = session.createQuery(hql); // This is an HQL query which helps to create query object

int i = query.executeUpdate(); //This will update the query and return how many object it is updated
```

Consider the table below:

roll	**name**	**email**	**marks**
1	bob	bob@gmail.com	25
3	charli	charli@gmail.com	40
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	75
6	zakir	zakir@gmail.com	75

program9.java

```
package com.tap.app;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;
import com.tap.model.Student;
```

```
public class Program1 {
	public static void main(String[] args) {
		Configuration config = new Configuration();
		config.configure();
		config.addAnnotatedClass(Student.class);
		SessionFactory sessionFactory = config.buildSessionFactory();
		Session session = sessionFactory.openSession();
		Transaction transaction = session.beginTransaction();

//		Logics

		String hql = "UPDATE Student s SET s.marks = 50 WHERE s.marks < 50";
		Query query = session.createQuery(hql);
		int i = query.executeUpdate();
		System.out.println(i);

		transaction.commit();
	}
	}
}
```

Database

roll	name	email	marks
1	bob	bob@gmail.com	50
3	charli	charli@gmail.com	50
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	75
6	zakir	zakir@gmail.com	75

Now if you can see from the above table bob and charli marks were less than 50 now it is updated with the value 50. In the above program executeUpdate() will return number of rows it has updated

Scenario 7:

Now let's see how to delete single row from the table

Logic:

```
Student student = session.get(Student.class, 6); // Get the student object that you want to delete using get()

session.delete(student); // Delete the student object using delete()
```

Consider the table below, here we will delete zakir data from the table

roll	name	email	marks
1	bob	bob@gmail.com	50
3	charli	charli@gmail.com	50
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	75
6	zakir	zakir@gmail.com	75

Program10.java

```
package com.tap.app;

import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;
import com.tap.model.Student;

public class Program1 {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Configuration config = new Configuration();

		config.configure();

		config.addAnnotatedClass(Student.class);

		SessionFactory sessionFactory = config.buildSessionFactory();

		Session session = sessionFactory.openSession();

		Transaction transaction = session.beginTransaction();

//		Logics
		Student student = session.get(Student.class, 6);
		session.delete(student);

		transaction.commit();

	}

	}
```

Data base:

roll	name	email	marks
1	bob	bob@gmail.com	50
3	charli	charli@gmail.com	50
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	75

You can clearly see from the above table using delete() now zakir data with primary key 6 is been deleted from the database

Scenario 8:

Delete multiple row from the table where marks of the student is 50

HQL Syntax to delete multiple rows from the table

```
String hql = "DELETE FROM Student s WHERE s.marks = 50"; //Here
query is written to delete all the student object whose marks
is equal to 50

Query query = session.createQuery(hql); // This is an HQL query
which helps to create query object

int i = query.executeUpdate(); // It is used to update the
query object
```

Consider the table below:

roll	name	email	marks
1	bob	bob@gmail.com	50
3	charli	charli@gmail.com	50
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	75

Program11.java

```
package com.tap.app;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;
import com.tap.model.Student;

public class Program1 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Configuration config = new Configuration();

		config.configure();

		config.addAnnotatedClass(Student.class);

		SessionFactory sessionFactory = config.buildSessionFactory();

		Session session = sessionFactory.openSession();

		Transaction transaction = session.beginTransaction();

//		Logics
		String hql = "DELETE FROM Student s WHERE s.marks = 50";
		Query query = session.createQuery(hql);
		int i = query.executeUpdate();
		System.out.println(i);

		transaction.commit();

	}

	}
```

Database:

roll	**name**	**email**	**marks**
4	dravid	dravid@gmail.com	100
5	elton	elton@gmail.com	75

From the above table you can clearly see the row with marks equal to 50 is been deleted