AIM: SPRING JDBC

THEORY:

Java Database Connectivity(JDBC):

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver, o Native Driver,
- Network Protocol Driver, and
- Thin Driver

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.

The problems of JDBC API are as follows:

o We need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc. o We need to perform exception handling code on the database logic. o We need to handle transaction. o Repetition of all these codes from one to another database logic is a time-consuming task.

Spring JdbcTemplate is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

Advantage of Spring JdbcTemplate:

Spring JdbcTemplate eliminates all the above-mentioned problems of JDBC API. It provides you methods to write the queries directly, so it saves a lot of work and time.

JdbcTemplate class:

It is the central class in the Spring JDBC support classes. It takes care of creation and release of resources such as creating and closing of connection object etc. So it will not lead to any problem if you forget to close the connection. It handles the exception and provides the informative exception messages by the help of excepion classes defined in the **org.springframework.dao** package. We can perform all the database operations by the help of JdbcTemplate class such as insertion, updation, deletion and retrieval of the data from the database. Let's see the methods of spring JdbcTemplate class.

Method	Description
public int update(String query)	is used to insert, update and delete records.
public int update(String query,Object... args)	is used to insert, update and delete records using PreparedStatement using given arguments.
public void execute(String query)	is used to execute DDL query.
public T execute(String sql, PreparedStatementCallback action)	executes the query by using PreparedStatement callback.
public T query(String sql, ResultSetExtractor rse)	is used to fetch records using ResultSetExtractor.
public List query(String sql, RowMapper rse)	is used to fetch records using RowMapper.

Creating database and creating table in mysql:

```
mysql> create database springjdbc;
Query OK, 1 row affected (0.00 sec)

mysql> use springjdbc;
Database changed
mysql> CREATE TABLE employee(
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     name VARCHAR(255) NOT NULL,
    ->     salary INT NOT NULL,
    ->     PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> desc employee;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| name   | varchar(255) | NO   |     | NULL    |                |
| salary | int(11)      | NO   |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
3 rows in set (0.02 sec)
```

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>jdbc</groupId>
 <artifactId>demo</artifactId>
 <version>0.0.1-SNAPSHOT</version>

 <name>demo</name>
 <!-- FIXME change it to the project's website -->
 <url>http://www.example.com</url>

 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
 </properties>

 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.11</version>
   <scope>test</scope>
  </dependency>

     <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.9.8</version>
</dependency>
  <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.3.16</version>
</dependency>

  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.39</version>
```

```
</dependency>
   <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-jdbc</artifactId>
   <version>5.3.14</version>
</dependency>

 </dependencies>

 <build>
   <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be
moved to parent pom) -->
     <plugins>
      <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-
core/lifecycles.html#clean_Lifecycle -->
      <plugin>
       <artifactId>maven-clean-plugin</artifactId>
       <version>3.1.0</version>
      </plugin>
      <!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-
core/default-bindings.html#Plugin_bindings_for_jar_packaging -->
      <plugin>
       <artifactId>maven-resources-plugin</artifactId>
       <version>3.0.2</version>
      </plugin>
      <plugin>
       <artifactId>maven-compiler-plugin</artifactId>
       <version>3.8.0</version>
      </plugin>
      <plugin>
       <artifactId>maven-surefire-plugin</artifactId>
       <version>2.22.1</version>
      </plugin>
      <plugin>
       <artifactId>maven-jar-plugin</artifactId>
       <version>3.0.2</version>
      </plugin>
      <plugin>
       <artifactId>maven-install-plugin</artifactId>
       <version>2.5.2</version>
      </plugin>
      <plugin>
       <artifactId>maven-deploy-plugin</artifactId>
       <version>2.8.2</version>
      </plugin>
      <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-
core/lifecycles.html#site_Lifecycle -->
      <plugin>
       <artifactId>maven-site-plugin</artifactId>
       <version>3.7.1</version>
```

```
        </plugin>
        <plugin>
          <artifactId>maven-project-info-reports-plugin</artifactId>
          <version>3.0.0</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

dbConfig.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
   xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName" value="com.mysql.jdbc.Driver" />
<property name="url" value="jdbc:mysql://localhost:3308/springjdbc" />
<property name="username" value="root" />
<property name="password" value="Narend-10" />
</bean>

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="dataSource"></property>
</bean>

<bean id="edao" class="jdbc.demo.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>

<bean id="pstdao" class="jdbc.demo.preparedst.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>

<bean id="readdao" class="jdbc.demo.ResultSetExtractor.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>

<bean id="rowdao" class="jdbc.demo.RowMapper.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>

</beans>
```

A) Write a program to insert, update and delete records from the given table. (Employee Table-Id ,Name, Age)

SOURCE CODE:

dbConfig.xml:

```
<bean id="edao" class="jdbc.demo.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
```

Employee.java:

```
package jdbc.demo;

public class Employee {
	private int id;
	private String name;
	private float salary;

	public Employee() {

	}

	public Employee(String name, float salary) {
		super();
		this.name = name;
		this.salary = salary;
	}

	public Employee(int id, String name, float salary) {
		super();
		this.id = id;
		this.name = name;
		this.salary = salary;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public float getSalary() {
		return salary;
	}
```

```
        public void setSalary(float salary) {
                this.salary = salary;
        }
  }
```

EmployeeDao.java:

```
package jdbc.demo;

import java.util.List;

import org.springframework.jdbc.core.JdbcTemplate;

public class EmployeeDao {

private JdbcTemplate jdbcTemplate;

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
   this.jdbcTemplate = jdbcTemplate;
}

public int saveEmployee(Employee e){
   String query="insert into employee(name, salary)
values('"+e.getName()+"','"+e.getSalary()+"')";
   return jdbcTemplate.update(query);
}

public int updateEmployee(Employee e){
   String query="update employee set name='"+e.getName()+"',salary='"+e.getSalary()+"'
where id='"+e.getId()+"' ";
   return jdbcTemplate.update(query);
}

public int deleteEmployee(Employee e){
   String query="delete from employee where id='"+e.getId()+"' ";
   return jdbcTemplate.update(query);
}

public List<Employee> listEmployees() {
     String SQL = "select * from employee";
     List <Employee> emps = jdbcTemplate.query(SQL, new EmployeeMapper());
     return emps;
  }

}
```

EmployeeMapper.java:

```
package jdbc.demo;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class EmployeeMapper implements RowMapper<Employee> {
  public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {
            Employee e = new Employee();
            e.setId(rs.getInt("id"));
            e.setName(rs.getString("name"));
            e.setSalary(rs.getInt("salary"));
            return e;
          }
}
```

App.java:

```
package jdbc.demo;

import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * Narender Keswani
 *
 */
public class App {
public static void main(String[] args) {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("dbConfig.xml");

        EmployeeDao dao = (EmployeeDao) ctx.getBean("edao");

        // Insert Operation
        dao.saveEmployee(new Employee("Narender Keswani", 50000));
        dao.saveEmployee(new Employee("Neel Deshmukh", 60000));
        dao.saveEmployee(new Employee("Ronak Karia", 40000));
        dao.saveEmployee(new Employee("Hassan Haque", 70000));
        dao.saveEmployee(new Employee("Ritesh Yadav", 80000));

        // Read Operation (R)
        System.out.println("------Listing Employee Records After Insert--------");
        List<Employee> emps = dao.listEmployees();

        for (Employee record : emps) {
                System.out.print("ID : " + record.getId());
                System.out.print(", Name : " + record.getName());
```

```
                System.out.println(", Salary : " + record.getSalary());
        }
        // Update Operation (U)
        dao.updateEmployee(new Employee(16, "Ritesh Yadav", 60000));

        // Read Operation (R)
        System.out.println("------Listing Employee Records After Update--------");
        List<Employee> emps1 = dao.listEmployees();

        for (Employee record : emps1) {
                System.out.print("ID : " + record.getId());
                System.out.print(", Name : " + record.getName());
                System.out.println(", Salary : " + record.getSalary());
        }
        // Delete Operation (D)
        Employee e = new Employee();
        e.setId(14);
        dao.deleteEmployee(e);

        // Read Operation (R)
        System.out.println("------Listing Employee Records After Delete--------");
        List<Employee> emps2 = dao.listEmployees();

        for (Employee record : emps2) {
                System.out.print("ID : " + record.getId());
                System.out.print(", Name : " + record.getName());
                System.out.println(", Salary : " + record.getSalary());
        }
}
}
```

OUTPUT:

```
<terminated> App (4) [Java Application] C:\Users\NARENDER KESWANI\.p2\poo
------Listing Employee Records After Insert--------
ID : 12, Name : Narender Keswani, Salary : 50000.0
ID : 13, Name : Neel Deshmukh, Salary : 60000.0
ID : 14, Name : Ronak Karia, Salary : 40000.0
ID : 15, Name : Hassan Haque, Salary : 70000.0
ID : 16, Name : Ritesh Yadav, Salary : 80000.0
------Listing Employee Records After Update--------
ID : 12, Name : Narender Keswani, Salary : 50000.0
ID : 13, Name : Neel Deshmukh, Salary : 60000.0
ID : 14, Name : Ronak Karia, Salary : 40000.0
ID : 15, Name : Hassan Haque, Salary : 70000.0
ID : 16, Name : Ritesh Yadav, Salary : 60000.0
------Listing Employee Records After Delete--------
ID : 12, Name : Narender Keswani, Salary : 50000.0
ID : 13, Name : Neel Deshmukh, Salary : 60000.0
ID : 15, Name : Hassan Haque, Salary : 70000.0
ID : 16, Name : Ritesh Yadav, Salary : 60000.0
```

B) Write a program to demonstrate PreparedStatement in Spring JdbcTemplate

SOURCE CODE:

dbConfig.xml:

```
<bean id="pstdao" class="jdbc.demo.preparedst.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
```

Employee.java:

```
package jdbc.demo.preparedst;

public class Employee {
private int id;
private String name;
private float salary;

public Employee() {

}

public Employee(String name, float salary) {
	super();
	this.name = name;
	this.salary = salary;
}

public Employee(int id, String name, float salary) {
	super();
	this.id = id;
	this.name = name;
	this.salary = salary;
}
public int getId() {
	return id;
}
public void setId(int id) {
	this.id = id;
}
public String getName() {
	return name;
}
public void setName(String name) {
	this.name = name;
}
public float getSalary() {
	return salary;
}
public void setSalary(float salary) {
	this.salary = salary;
```

```
}
}
```

EmployeeMapper.java:

```
package jdbc.demo.preparedst;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class EmployeeMapper implements RowMapper<Employee> {
  public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {
            Employee e = new Employee();
            e.setId(rs.getInt("id"));
            e.setName(rs.getString("name"));
            e.setSalary(rs.getInt("salary"));
            return e;
          }
}
```

EmployeeDao.java:

```
package jdbc.demo.preparedst;

import java.sql.SQLException;
import java.util.List;

import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCallback;

public class EmployeeDao {

private JdbcTemplate jdbcTemplate;

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
  this.jdbcTemplate = jdbcTemplate;
}


public Boolean saveEmployee(final Employee e){
        String query="insert into employee(name, salary) values(?,?)";
  return jdbcTemplate.execute(query,new PreparedStatementCallback<Boolean>(){

                @Override
                public Boolean doInPreparedStatement(java.sql.PreparedStatement ps)
                                throws SQLException, DataAccessException {
                        // TODO Auto-generated method stub
                    ps.setString(1,e.getName());
```

```
                ps.setFloat(2,e.getSalary());

                return ps.execute();
            }
    });

}

public Boolean updateEmployee(final Employee e){
        String query="update employee set name=?,salary=? where id=?";
    return jdbcTemplate.execute(query,new PreparedStatementCallback<Boolean>(){

            @Override
            public Boolean doInPreparedStatement(java.sql.PreparedStatement ps)
                    throws SQLException, DataAccessException {
                // TODO Auto-generated method stub
                ps.setString(1,e.getName());
                ps.setFloat(2,e.getSalary());
                ps.setInt(3,e.getId());

                return ps.execute();
            }
    });

}

public Boolean deleteEmployee(final Employee e){
        String query="delete from employee where id=?";
    return jdbcTemplate.execute(query,new PreparedStatementCallback<Boolean>(){

            @Override
            public Boolean doInPreparedStatement(java.sql.PreparedStatement ps)
                    throws SQLException, DataAccessException {
                // TODO Auto-generated method stub
                ps.setInt(1,e.getId());

                return ps.execute();
            }
    });

}

public List<Employee> listEmployees() {
    String SQL = "select * from employee";
    List <Employee> emps = jdbcTemplate.query(SQL, new EmployeeMapper());
    return emps;
  }

}
```

App.java:

```
package jdbc.demo.preparedst;

import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

public static void main(String[] args) {
	// TODO Auto-generated method stub
	ApplicationContext ctx = new ClassPathXmlApplicationContext("dbConfig.xml");

	EmployeeDao dao = (EmployeeDao) ctx.getBean("pstdao");

	// Insert Operation
	dao.saveEmployee(new Employee("Wilson Rao", 50000));
	dao.saveEmployee(new Employee("Utkarsh Bafana", 60000));
	dao.saveEmployee(new Employee("Abhishek Dere", 40000));
	dao.saveEmployee(new Employee("Kalpesh Khandewal", 70000));
	dao.saveEmployee(new Employee("Ujjal Kumar Ray", 80000));

	// Read Operation (R)
	System.out.println("------Listing Employee Records After Insert--------");
	List<Employee> emps = dao.listEmployees();

	for (Employee record : emps) {
		System.out.print("ID : " + record.getId());
		System.out.print(", Name : " + record.getName());
		System.out.println(", Salary : " + record.getSalary());
	}

	// Update Operation (U)
	dao.updateEmployee(new Employee(16, "Abhishek Dere [Mazgaon]", 30000));

	// Read Operation (R)
	System.out.println("------Listing Employee Records After Update--------");
	List<Employee> emps1 = dao.listEmployees();

	for (Employee record : emps1) {
		System.out.print("ID : " + record.getId());
		System.out.print(", Name : " + record.getName());
		System.out.println(", Salary : " + record.getSalary());
	}

	// Delete Operation (D)
	Employee e = new Employee();
	e.setId(21);
	dao.deleteEmployee(e);
```

```
		// Read Operation (R)
		System.out.println("------Listing Employee Records After Delete--------");
		List<Employee> emps2 = dao.listEmployees();

		for (Employee record : emps2) {
			System.out.print("ID : " + record.getId());
			System.out.print(", Name : " + record.getName());
			System.out.println(", Salary : " + record.getSalary());
		}
}
}
```

OUTPUT:

```
------Listing Employee Records After Insert--------
ID : 12, Name : Narender Keswani, Salary : 50000.0
ID : 13, Name : Neel Deshmukh, Salary : 60000.0
ID : 15, Name : Hassan Haque, Salary : 70000.0
ID : 16, Name : Ritesh Yadav, Salary : 60000.0
ID : 17, Name : Wilson Rao, Salary : 50000.0
ID : 18, Name : Utkarsh Bafana, Salary : 60000.0
ID : 19, Name : Abhishek Dere, Salary : 40000.0
ID : 20, Name : Kalpesh Khandewal, Salary : 70000.0
ID : 21, Name : Ujjal Kumar Ray, Salary : 80000.0
------Listing Employee Records After Update--------
ID : 12, Name : Narender Keswani, Salary : 50000.0
ID : 13, Name : Neel Deshmukh, Salary : 60000.0
ID : 15, Name : Hassan Haque, Salary : 70000.0
ID : 16, Name : Abhishek Dere [Mazgaon], Salary : 30000.0
ID : 17, Name : Wilson Rao, Salary : 50000.0
ID : 18, Name : Utkarsh Bafana, Salary : 60000.0
ID : 19, Name : Abhishek Dere, Salary : 40000.0
ID : 20, Name : Kalpesh Khandewal, Salary : 70000.0
ID : 21, Name : Ujjal Kumar Ray, Salary : 80000.0
------Listing Employee Records After Delete--------
ID : 12, Name : Narender Keswani, Salary : 50000.0
ID : 13, Name : Neel Deshmukh, Salary : 60000.0
ID : 15, Name : Hassan Haque, Salary : 70000.0
ID : 16, Name : Abhishek Dere [Mazgaon], Salary : 30000.0
ID : 17, Name : Wilson Rao, Salary : 50000.0
ID : 18, Name : Utkarsh Bafana, Salary : 60000.0
ID : 19, Name : Abhishek Dere, Salary : 40000.0
ID : 20, Name : Kalpesh Khandewal, Salary : 70000.0
```

C) Write a program in Spring JDBC to demonstrate ResultSetExtractor Interface

SOURCE CODE:

dbConfig.xml:

```
<bean id="readdao" class="jdbc.demo.ResultSetExtractor.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
```

Employee.java:

```
package jdbc.demo.ResultSetExtractor;

public class Employee {
  private int id;
  private String name;
  private float salary;

  public Employee() {

  }

  public Employee(String name, float salary) {
        super();
        this.name = name;
        this.salary = salary;
  }

  public Employee(int id, String name, float salary) {
        super();
        this.id = id;
        this.name = name;
        this.salary = salary;
  }
  public int getId() {
        return id;
  }
  public void setId(int id) {
        this.id = id;
  }
  public String getName() {
        return name;
  }
  public void setName(String name) {
        this.name = name;
  }
  public float getSalary() {
        return salary;
  }
  public void setSalary(float salary) {
```

```
            this.salary = salary;
    }
  }
```

Employee.Dao.java:

```
package jdbc.demo.ResultSetExtractor;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.ResultSetExtractor;

public class EmployeeDao {

    private JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
       this.jdbcTemplate = jdbcTemplate;
    }



    public List<Employee> listEmployees() {
            return jdbcTemplate.query("select * from employee",new
ResultSetExtractor<List<Employee>>(){
                    public List<Employee> extractData(ResultSet rs) throws
SQLException,
                            DataAccessException {

                        List<Employee> list=new ArrayList<Employee>();
                        while(rs.next()){
                        Employee e=new Employee();
                        e.setId(rs.getInt(1));
                        e.setName(rs.getString(2));
                        e.setSalary(rs.getInt(3));
                        list.add(e);
                        }
                        return list;
                        }
                    });
                  }
                }
```

App.java:

```
package jdbc.demo.ResultSetExtractor;

import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

    public static void main(String[] args) {
            // TODO Auto-generated method stub
            ApplicationContext ctx = new
ClassPathXmlApplicationContext("dbConfig.xml");

            EmployeeDao dao = (EmployeeDao) ctx.getBean("readdao");

            // Read Operation (R)
            System.out.println("------Listing Employee Records--------");
            List<Employee> emps2 = dao.listEmployees();

            for (Employee record : emps2) {
                    System.out.print("ID : " + record.getId());
                    System.out.print(", Name : " + record.getName());
                    System.out.println(", Salary : " + record.getSalary());
            }
    }
    }
```

OUTPUT:

```
------Listing Employee Records--------
ID : 12, Name : Narender Keswani, Salary : 50000.0
ID : 13, Name : Neel Deshmukh, Salary : 60000.0
ID : 15, Name : Hassan Haque, Salary : 70000.0
ID : 16, Name : Abhishek Dere [Mazgaon], Salary : 30000.0
ID : 17, Name : Wilson Rao, Salary : 50000.0
ID : 18, Name : Utkarsh Bafana, Salary : 60000.0
ID : 19, Name : Abhishek Dere, Salary : 40000.0
ID : 20, Name : Kalpesh Khandewal, Salary : 70000.0
```

D) Write a program to demonstrate RowMapper interface to fetch the records from the database

SOURCE CODE:

dbConfig.xml:

```
<bean id="rowdao" class="jdbc.demo.RowMapper.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
```

Employee.java:

```
package jdbc.demo.RowMapper;

public class Employee {
private int id;
private String name;
private float salary;

public Employee() {

}

public Employee(String name, float salary) {
	super();
	this.name = name;
	this.salary = salary;
}

public Employee(int id, String name, float salary) {
	super();
	this.id = id;
	this.name = name;
	this.salary = salary;
}
public int getId() {
	return id;
}
public void setId(int id) {
	this.id = id;
}
public String getName() {
	return name;
}
public void setName(String name) {
	this.name = name;
}
public float getSalary() {
```

```
        return salary;
}
public void setSalary(float salary) {
        this.salary = salary;
}
}
```

EmployeeMapper.java:

```
package jdbc.demo.RowMapper;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class EmployeeMapper implements RowMapper<Employee> {
  public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {
            Employee e = new Employee();
            e.setId(rs.getInt("id"));
            e.setName(rs.getString("name"));
            e.setSalary(rs.getInt("salary"));
            return e;
          }
}
```

EmployeeDao.java:

```
package jdbc.demo.RowMapper;

import java.util.List;

import org.springframework.jdbc.core.JdbcTemplate;

public class EmployeeDao {

private JdbcTemplate jdbcTemplate;

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
   this.jdbcTemplate = jdbcTemplate;
}


public List<Employee> listEmployees() {
    String SQL = "select * from employee";
    List <Employee> emps = jdbcTemplate.query(SQL, new EmployeeMapper());
    return emps;
  }

}
```

App.java:

```
package jdbc.demo.RowMapper;

import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * Narender Keswani
 *
 */
public class App {
public static void main(String[] args) {
	ApplicationContext ctx = new ClassPathXmlApplicationContext("dbConfig.xml");

	EmployeeDao dao = (EmployeeDao) ctx.getBean("rowdao");

	// Read Operation (R)
	System.out.println("------Listing Employee Records--------");
	List<Employee> emps = dao.listEmployees();

	for (Employee record : emps) {
		System.out.print("ID : " + record.getId());
		System.out.print(", Name : " + record.getName());
		System.out.println(", Salary : " + record.getSalary());
	}


}
}
```

OUTPUT:

```
------Listing Employee Records--------
ID : 12, Name : Narender Keswani, Salary : 50000.0
ID : 13, Name : Neel Deshmukh, Salary : 60000.0
ID : 15, Name : Hassan Haque, Salary : 70000.0
ID : 16, Name : Abhishek Dere [Mazgaon], Salary : 30000.0
ID : 17, Name : Wilson Rao, Salary : 50000.0
ID : 18, Name : Utkarsh Bafana, Salary : 60000.0
ID : 19, Name : Abhishek Dere, Salary : 40000.0
ID : 20, Name : Kalpesh Khandewal, Salary : 70000.0
```

CONCLUSION:

From this practical, I have learned how to implement CRUD operations using jdbc template and Spring.