Phase 2

Spring JDBC **21-02-2025**

```
package com.wipro.config;


import javax.sql.DataSource;


import org.springframework.context.annotation.Bean;
 import org.springframework.context.annotation.ComponentScan;
 import org.springframework.context.annotation.Configuration;
 import org.springframework.jdbc.core.JdbcTemplate;
 import org.springframework.jdbc.datasource.DriverManagerDataSource;


@Configuration
 @ComponentScan(basePackages = "com.wipro")
 public class MyConfiguration {


      @Bean
        public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/wipro");
        dataSource.setUsername("root");
        dataSource.setPassword("root");
        return dataSource;
        }


        @Bean
        public JdbcTemplate jdbcTemplate(DataSource dataSource) {
        return new JdbcTemplate(dataSource);
        }
```

```
 }
```

package com.wipro.dao; import com.wipro... by RK (Unverified)RK (Unverified)12:39 PM

```
package com.wipro.dao;


import com.wipro.model.Student;


public interface StudentDAO {


	int saveStudent(Student student);


}
```

package com.wipro.daoimpl; import org.spr... by RK (Unverified)RK (Unverified)12:39 PM

```
package com.wipro.daoimpl;


import org.springframework.beans.factory.annotation.Autowired;
 import org.springframework.jdbc.core.JdbcTemplate;
 import org.springframework.stereotype.Repository;


import com.wipro.dao.StudentDAO;
 import com.wipro.model.Student;


@Repository(value = "student")
 public class StudentDAOImpl implements StudentDAO {
```

```
	@Autowired
	 private  JdbcTemplate template;

	 public int saveStudent(Student student) {


		String sql = "insert into student values(?,?,?)";

		return template.update(sql,
student.getStid(),student.getStname(),student.getStaddress());

	 }


}
```

package com.wipro.model; public class ... by RK (Unverified)RK (Unverified)12:40 PM

```
package com.wipro.model;


public class Student {


	private Integer stid;

	private String stname;

	private String staddress;



	public Integer getStid() {

		return stid;

	}

	public void setStid(Integer stid) {

		this.stid = stid;
```

```
}
public String getStname() {
	return stname;
}
public void setStname(String stname) {
	this.stname = stname;
}
public String getStaddress() {
	return staddress;
}
public void setStaddress(String staddress) {
	this.staddress = staddress;
}
public Student(Integer stid, String stname, String staddress) {
	super();
	this.stid = stid;
	this.stname = stname;
	this.staddress = staddress;
}

@Override
public String toString()
{
	return stid + " " + stname + " " + staddress;
```

```
    }
    public Student()
    {

    }

}
```

<dependency> <groupId>org.springframewo... by RK (Unverified)RK (Unverified)12:40 PM

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>6.1.14</version>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.33</version>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
```

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>6.1.14</version>
</dependency>
```

What is the difference between the jdbc and spring jdbc

JDBC (Java Database Connectivity)

- **What It Is**: A low-level API provided by Java to connect to databases and execute SQL queries.
- **How It Works**: You **manually** write all the code for:
    - Opening/closing database connections.
    - Creating `PreparedStatement`/`Statement` objects.
    - Handling exceptions.
    - Managing resource cleanup (connections, statements, result sets).

Features of JDBC:

1. **Manual Effort**: You handle every step manually.
2. **Error-Prone**: If you forget to close a connection, it can lead to resource leaks.
3. **Boilerplate Code**: You write repetitive code for handling exceptions, connections, etc.

Spring JDBC

- **What It Is**: A higher-level abstraction provided by the Spring Framework on top of JDBC.
- **How It Works**: Spring manages:
    - Database connections.
    - Exception handling.
    - Resource cleanup.
    - Replacing boilerplate code with simpler APIs.

Features of Spring JDBC:

1. **Less Code**: Removes the need for repetitive boilerplate code (e.g., connection management).
2. **Exception Translation**: Converts low-level `SQLException` into Spring's `DataAccessException`, making it easier to handle.
3. **Built-In Templates**: Uses `JdbcTemplate` to simplify queries (e.g., `insert`, `update`, `delete`).
4. **Integration Ready**: Works well with Spring's Dependency Injection (DI) and other features

What is JdbcTemplate?

JdbcTemplate is a class in Spring Framework that simplifies interacting with a relational database. It is part of Spring's **Spring JDBC module** and provides a convenient way to work with databases by reducing the boilerplate code required for JDBC.

Why Use JdbcTemplate?

1. **Simplifies JDBC Code**: It handles common tasks like connection management, SQL execution, and result set processing for you.
2. **Automatic Resource Management**: It ensures that connections, statements, and result sets are properly closed after use.
3. **Less Code**: Removes the need for repetitive and error-prone code like `try-catch-finally` blocks.
4. **Integrated with Spring**: Works seamlessly with Spring's dependency injection.

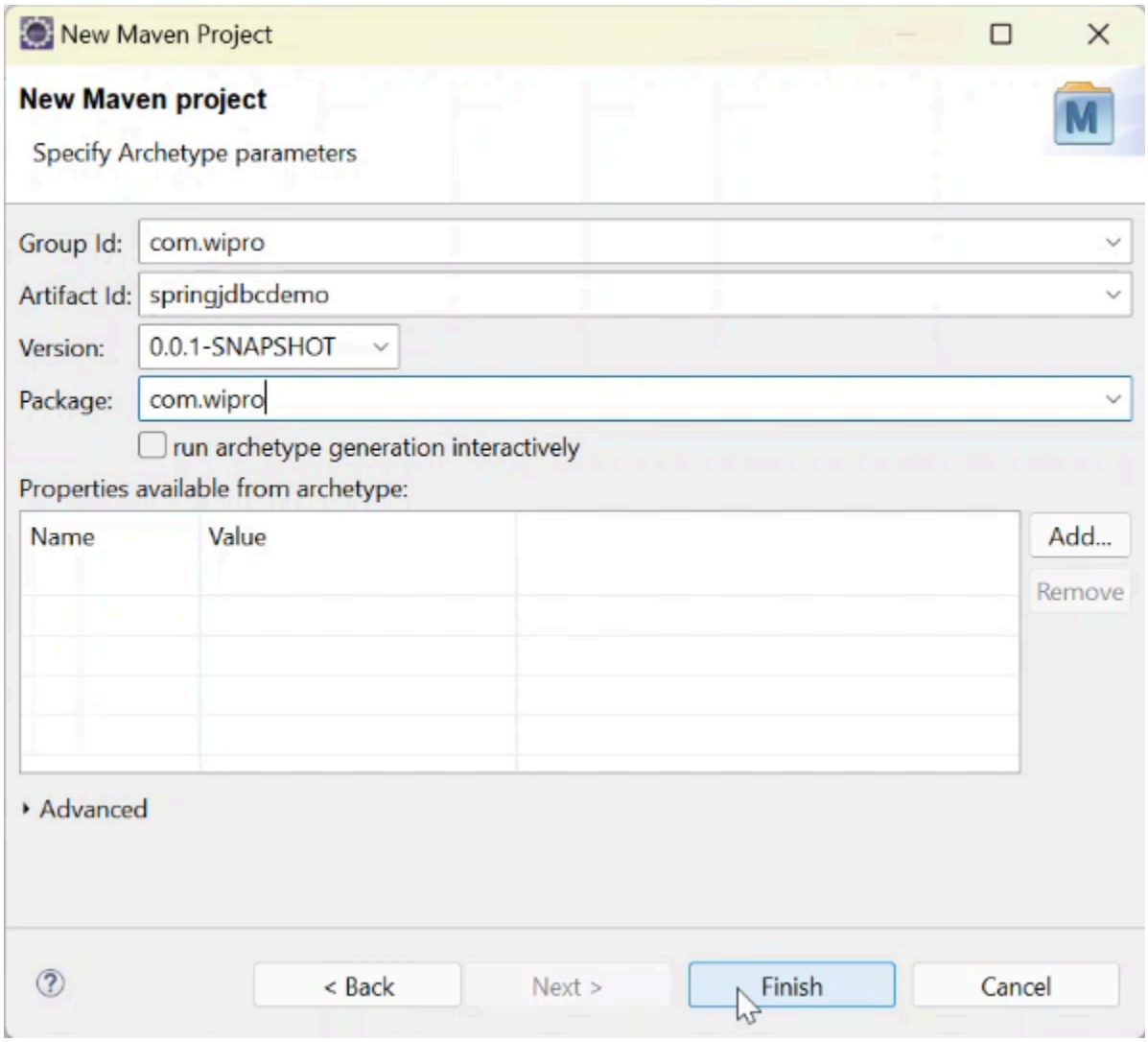
BOILER PLATE CODE

Boilerplate code refers to the repetitive, standard, or template-like code that is necessary in many programming tasks but does not directly contribute to the core logic or functionality of the program. It often involves code that is required for setup, configuration, or routine tasks.

In simple terms, **it's the "must-have" code that you need to write over and over again, even if it's not solving the main problem**.

Now we r moving into the eclipse

1.create the maven project>>choose quickstart 1.1 >>uncheck the runartitype>



Once the project is created we need to add the spring jdbc related dependencies and also we need to add the data base dependencies bcoz we r implementing the data base operations so we need to add them

Now open the pom.xml file

Add the spring mvc maven dependencies >>choose 6.1.14

<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->

<dependency>

<groupId>org.springframework</groupId>

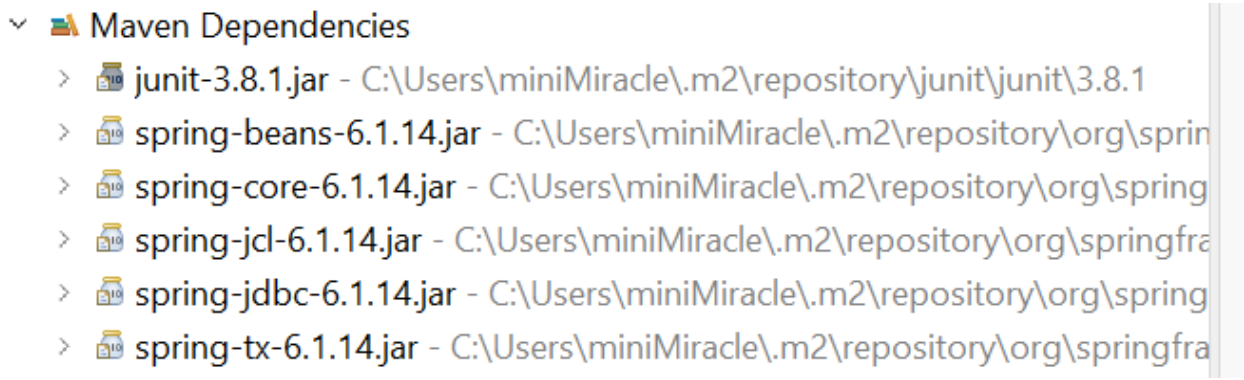
<artifactId>spring-jdbc</artifactId>

<version>6.1.14</version>

</dependency>

Into the xml file

As soon as we added the dependency into the xml file then automatically it is pulling the related jar files



Now we need to add the mysql maven dependency >8.0

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

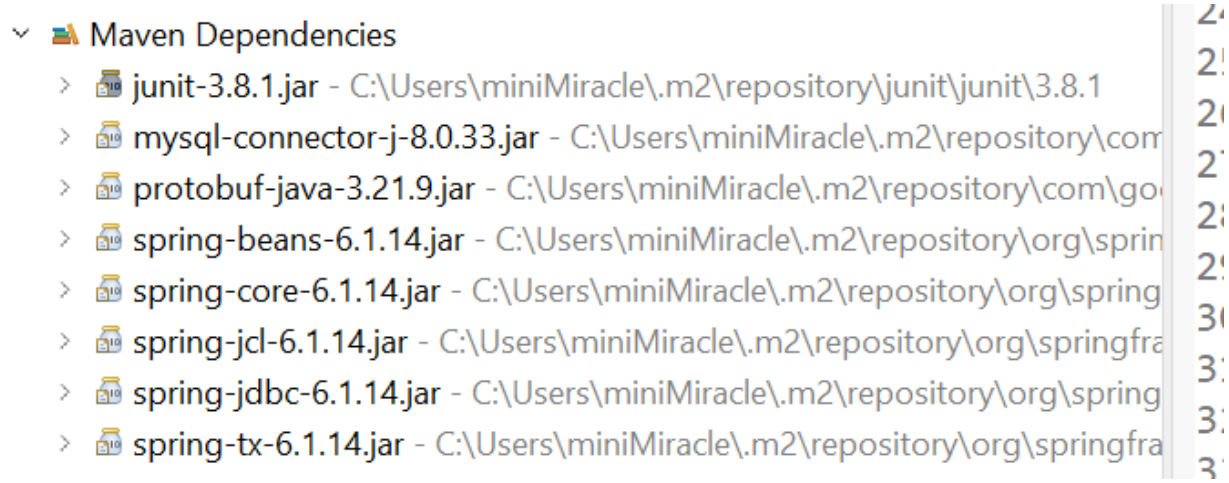
<artifactId>mysql-connector-java</artifactId>

<version>8.0.33</version>

</dependency>

Into the xml file

Now my sql related jars are automatically pulled



Now create a package under src/main/java named as **com.wipro.dao**

Where we r writing the logic

And create another package named as **com.wipro.model**

Now we need to define the class in the model package ie Student

```
1 package com.wipro.model;
2
3 public class Student {
4
5     private Integer stid;
6     private String stname;
7     private String staddress;
8 }
9
```

Generate the getters and setters and constructors too

And also generate the toString method

```
26
27      @Override
28      public String toString()
29      {
30          return stid + " " +stname +" "+ staddress;
31      }
32
33  }
```

And also give the defaut Constructr

Now we need to create the student DaO Interface

```
1  package com.wipro.dao;
2
3  import com.wipro.model.Student;
4
5  public interface StudentDAO
6  {
7      int saveStudent(Student student);
8
9  }
10
```

As it is a interface so we need to implement it in another package and we need to create one class where it contains the implementation classes

```java
package com.wipro.daoimpl;

import com.wipro.dao.StudentDAO;

public class StudentDAOImple implements StudentDAO {

    @Override
    public int saveStudent(Student student) {
        // TODO Auto-generated method stub
        return 0;
    }

}

```

Now we need to write the logic by using the jdbc template

For that we need to add the spring context maven depenecey

```xml
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.1.14</version>
</dependency>
```

Where it contains spring core maven dependencies

Now we need to add the @Repository in the above StudentDAOImpl

```java
package com.wipro.daoimpl;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import com.wipro.dao.StudentDAO;
import com.wipro.model.Student;
@Repository
public class StudentDAOImple implements StudentDAO {

    private JdbcTemplate template;
    public int saveStudent(Student student) {

        String sql="insert into student values(?,??)";
        return template.update(sql, student.getStid(),student.getStname(),student.getStaddress());

    }

}

```

Now we need to config this..for this we need to create one more package and a config class inside of it

Configuration class must be annotated with the @configuration

Overall codes

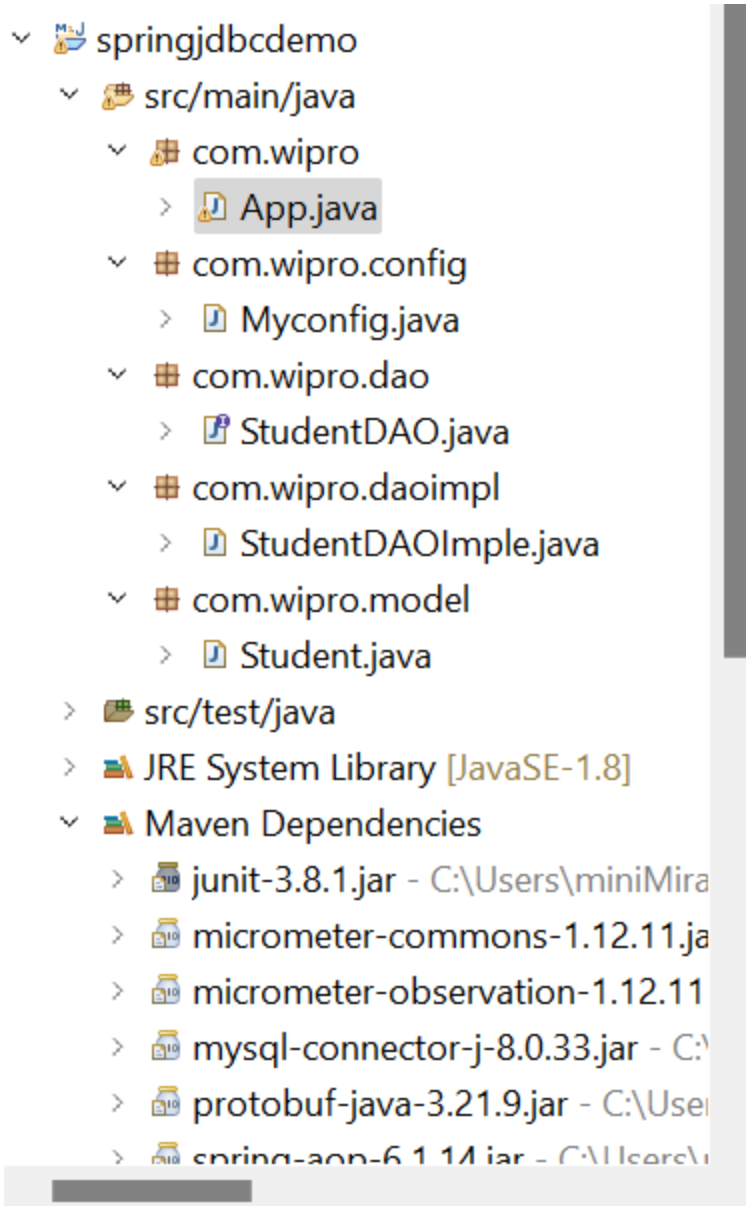
```java
package com.wipro.model;

public class Student {

    private Integer stid;
    private String stname;
    private String staddress;
    public Integer getStid() {
        return stid;
    }
    public void setStid(Integer stid) {
        this.stid = stid;
    }
    public String getStname() {
        return stname;
    }
    public void setStname(String stname) {
        this.stname = stname;
    }
    public String getStaddress() {
        return staddress;
    }
    public void setStaddress(String staddress) {
        this.staddress = staddress;
    }

    public Student(Integer stid, String stname, String staddress) {
        super();
        this.stid = stid;
        this.stname = stname;
        this.staddress = staddress;
    }
    @Override
    public String toString()
    {
        return stid + " " +stname +" "+ staddress;
    }
    public Student()
    {

    }

}
```

```java
package com.wipro.dao;

import com.wipro.model.Student;

public interface StudentDAO
{
    int saveStudent(Student student);

}

```

```java
package com.wipro.daoimpl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import com.wipro.dao.StudentDAO;
import com.wipro.model.Student;
@Repository
public class StudentDAOImple implements StudentDAO {

    @Autowired
    private JdbcTemplate template;
    public int saveStudent(Student student) {

        String sql="insert into student(stid,stname,staddress) values(?,?,?)";
        return template.update(sql, student.getStid(),student.getStname(),student.getStaddress());

    }

}

```

```java
package com.wipro.config;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

@Configuration
@ComponentScan(basePackages = "com.wipro")

public class Myconfig
{

	@Bean
	public DataSource dataSource() {
	    DriverManagerDataSource dataSource = new DriverManagerDataSource();
	    dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
	    dataSource.setUrl("jdbc:mysql://localhost:3306/wipro");
	    dataSource.setUsername("root");
	    dataSource.setPassword("#Mahadev7");
	    return dataSource;
	}

	@Bean
	public JdbcTemplate jdbcTemplate(DataSource dataSource) {
	    return new JdbcTemplate(dataSource);
	}


}
```

```java
package com.wipro;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.wipro.config.Myconfig;
import com.wipro.daoimpl.StudentDAOImple;
import com.wipro.model.Student;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context=new AnnotationConfigApplicationContext(Myconfig.class);
        StudentDAOImple dao=context.getBean(StudentDAOImple.class);

       int x= dao.saveStudent(new Student(24,"pavan","kadiri"));
       System.out.println(x +"rows inserted");
    }
}
```


springjdbcdemo
src/main/java
com.wipro
App.java
com.wipro.config
Myconfig.java
com.wipro.dao
StudentDAO.java
com.wipro.daoimpl
StudentDAOImple.java
com.wipro.model
Student.java
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
junit-3.8.1.jar - C:\Users\miniMira
micrometer-commons-1.12.11.ja
micrometer-observation-1.12.11
mysql-connector-j-8.0.33.jar - C:
protobuf-java-3.21.9.jar - C:\Use

```
http://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation with catalog)
<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>com.wipro</groupId>
  <artifactId>springjdbcdemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>springjdbcdemo</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>


    <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>6.1.14</version>
</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
</dependency>


<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.1.14</version>
</dependency>

```

```
40
41  <dependency>
42      <groupId>org.springframework</groupId>
43      <artifactId>spring-context</artifactId>
44      <version>6.1.14</version>
45  </dependency>
46
47
48  <dependency>
49      <groupId>org.springframework</groupId>
50      <artifactId>spring-beans</artifactId>
51      <version>6.1.14</version>
52  </dependency>
53  <dependency>
54      <groupId>org.springframework</groupId>
55      <artifactId>spring-core</artifactId>
56      <version>6.1.14</version>
57  </dependency>
58
59    </dependencies>
60  </project>
61
```

Run the app.java file

```
1rows inserted
```

As soon as we run the program it reflects in the sql

```
17 •  use wipro;
18 •  create table student(
19    stid int,
20    stname varchar(25),
21    staddress varchar(45)
22    );
23 •  select * from student;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

stid	stname	staddress
24	pavan	kadiri

student 6

Output

Action Output

#	Time	Action
17	13:56:00	use wipro
18	13:56:40	create table student(stid int, stname varchar(25), staddress varchar(45))
19	14:21:19	select * from student LIMIT 0, 1000

Now we r performing another operations like update and delete for the same code by changing few logics

```java
package com.wipro.dao;

import com.wipro.model.Student;

public interface StudentDAO
{
    int saveStudent(Student student);

    int updateStudent(Student student);

    int deleteStudent(Student student);

}

```

```java
package com.wipro.daoimpl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import com.wipro.dao.StudentDAO;
import com.wipro.model.Student;
@Repository
public abstract class StudentDAOImple implements StudentDAO {

    @Autowired
    private JdbcTemplate template;
    public int saveStudent(Student student) {

        String sql="insert into student(stid,stname,staddress) values(?,?,?)";
        return template.update(sql, student.getStid(),student.getStname(),student.getStaddress());

    }

        public int updateStudent(Student student) {

        String sql="update student set stname=?, staddress=? where stid=?";
        return template.update(sql, student.getStname(),student.getStaddress(),student.getStid());

    }
        public int deleteStudent(int stid) {
            String sql = "delete from student where stid=?";
            return template.update(sql, stid);
        }


}
```

```java
package com.wipro;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.wipro.config.Myconfig;
import com.wipro.daoimpl.StudentDAOImple;
import com.wipro.model.Student;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context=new AnnotationConfigApplicationContext(Myconfig.class);
        StudentDAOImple dao=context.getBean(StudentDAOImple.class);

       int x= dao.saveStudent(new Student(24,"pavan","kadiri"));
       System.out.println(x +"rows inserted");


       int y= dao.updateStudent(new Student(100,"kalyan","kdr"));
       System.out.println(y +"rows updated");

       int z = dao.deleteStudent(100);
       System.out.println(z + " rows deleted");

    }
}
```

stid	stname	staddress
24	kalyan	kdr
24	kalyan	kdr
24	kalyan	kdr
24	pavan	kadiri
24	pavan	kadiri
24	pavan	kadiri

kadiri

Query and queryforObject and also usage of row mapper

Difference between `query()` and `queryForObject()` in Spring JDBC (`JdbcTemplate`)

Both `query()` and `queryForObject()` are methods of `JdbcTemplate` that help execute SQL queries and retrieve results, but they behave differently.

1️⃣ `queryForObject()`

- Used when you expect exactly one result (single row, single column).
- Throws an `IncorrectResultSizeDataAccessException` if the query returns more than one row.
- Throws an `EmptyResultDataAccessException` if no data is found.

Example: Fetch a single student by ID

```
String sql = "SELECT * FROM student WHERE id=?";
Student student = template.queryForObject(sql, new StudentRowMapper(), 100);
System.out.println(student.getName() + " - " + student.getAddress());
```

Scenarios:

Query Result	Behavior
1 row found	✅ Returns the `Student` object
0 rows found	❌ Throws `EmptyResultDataAccessException`
More than 1 row	❌ Throws `IncorrectResultSizeDataAccessException`

2️⃣ `query()`

- Used when you expect multiple rows from the database.
- Returns a `List<T>` of objects.

- If no results are found, it returns an empty list instead of throwing an exception.

Example: Fetch all students

```
String sql = "SELECT * FROM student";
List<Student> students = template.query(sql, new StudentRowMapper());

for (Student s : students) {
   System.out.println(s.getId() + " - " + s.getName() + " - " + s.getAddress());
}
```

Scenarios:

Query Result	Behavior
Multiple rows	✅ Returns a list of `Student` objects
1 row found	✅ Returns a list with one object
0 rows found	✅ Returns an empty list, no exception

🚀 Key Differences:

Feature	`queryForObject()`	`query()`
Returns	Single object (`T`)	List of objects (`List<T>`)
Expected Rows	Exactly one row	One or more rows
If No Rows Found	❌ Exception (`EmptyResultDataAccessException`)	✅ Returns empty list
If Multiple Rows Found	❌ Exception (`IncorrectResultSizeDataAccessException`)	✅ Returns all rows

When to Use Which?

✔ **Use `queryForObject()` when fetching a single row (e.g., finding a user by ID).**
✔ **Use `query()` when fetching multiple rows (e.g., retrieving all students).**

```
package com.wipro;


import java.util.List;


import org.springframework.context.ApplicationContext;
 import org.springframework.context.annotation.AnnotationConfigApplicationContext;


import com.wipro.config.MyConfiguration;
 import com.wipro.daoimpl.StudentDAOImpl;
 import com.wipro.model.Student;


/**
 * Hello world!
 *
 */
 public class App
 {
        public static void main( String[] args )
        {
        ApplicationContext context = new
AnnotationConfigApplicationContext(MyConfiguration.class);
        StudentDAOImpl dao =(StudentDAOImpl)context.getBean("student");

        /*int x =  dao.saveStudent(new Student(102,"ramesh","bangalore"));
        System.out.println(x + "row(s) inserted");

        int y = dao.updateStudent(new Student(100,"ram","hyd"));
        System.out.println(y + " row(s) updated");

        int z = dao.deleteStudent(300);
```

```
		System.out.println(z + "row(s) deleted");
		*/

		Student student =  dao.getStudentById(100);
		System.out.println(student);

		System.out.println("====================");

		List<Student> students = dao.getAllStudents();

		for(Student st:students)
		{
			System.out.println(st.getStid() + " " + st.getStname() + " " + st.getStaddress());
		}
		}
}
```

package com.wipro.dao; import java.util... by RK (Unverified)RK (Unverified)3:40 PM

```
package com.wipro.dao;


import java.util.List;


import com.wipro.model.Student;


public interface StudentDAO {


	int saveStudent(Student student);

	int updateStudent(Student student);

	int deleteStudent(int stid);

	Student getStudentById(int stid);

	List<Student> getAllStudents();
```

```
}
```

package com.wipro.daoimpl; import java.ut... by RK (Unverified)RK (Unverified)3:40 PM

```
package com.wipro.daoimpl;


import java.util.List;


import org.springframework.beans.factory.annotation.Autowired;
 import org.springframework.jdbc.core.JdbcTemplate;
 import org.springframework.stereotype.Repository;


import com.wipro.dao.StudentDAO;
 import com.wipro.model.Student;


@Repository(value = "student")
 public class StudentDAOImpl implements StudentDAO {


      @Autowired
       private  JdbcTemplate template;

       public int saveStudent(Student student) {


            String sql = "insert into student values(?,?,?)";

            return template.update(sql,
student.getStid(),student.getStname(),student.getStaddress());

       }


      public int updateStudent(Student student) {
            String sql = "update student set stname=?,staddress=? where stid=?";
```

```
            return
template.update(sql,student.getStname(),student.getStaddress(),student.getStid());

        }



        public int deleteStudent(int stid) {
                String sql = "delete from student where stid=?";
                return template.update(sql, stid);
        }



        public Student getStudentById(int stid) {
                String sql ="select * from student where stid=?";
                return template.queryForObject(sql, new StudentRowMapper(), stid);
        }



        public List<Student> getAllStudents() {
                String sql = "select * from student";
                return template.query(sql,new StudentRowMapper());
        }



}
```

package com.wipro.daoimpl; import java.sq... by RK (Unverified)RK (Unverified)3:40 PM

```
package com.wipro.daoimpl;



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



import org.springframework.jdbc.core.RowMapper;



import com.wipro.model.Student;
```

```
public class StudentRowMapper implements RowMapper<Student> {


	public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

		return new Student(rs.getInt("stid"),rs.getString("stname"),rs.getString("staddress"));

	}


}
```

If we r passing the student id then we will get the one student data

getStudentId

If we r not passing the student id then we will get the all student data by using the **getAllStudent()** method and it is returning the list of students

StudentDAO.java | StudentDAOIm... | MyConfig.ja

```java
package com.wipro.dao;

import java.util.List;

import com.wipro.model.Student;

public interface StudentDAO
{
    int saveStudent(Student student);

    int updateStudent(Student student);

    int deleteStudent(Student student);

    Student getStudentById(int stid);
    List<Student> getAllStudents();

}

```

Right now we r doing the **how to process the select query by using the spring jdbc**

```java
package com.wipro.daoimpl;

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

import org.springframework.jdbc.core.RowMapper;

import com.wipro.model.Student;

public class StudentRowMapper implements RowMapper<Student> {

    public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
        return new Student(rs.getInt("stid"),rs.getString("stname"),rs.getString("staddress"));
    }

}
```

Above code is from StudentDAOImple

Now we need to implement the rowmapper class

```java
package com.wipro.daoimpl;

import java.util.List;

@Repository(value = "student")
public class StudentDAOImpl implements StudentDAO {

    @Autowired
    private  JdbcTemplate template;
    public int saveStudent(Student student) {

        String sql = "insert into student values(?,?,?)";
        return template.update(sql, student.getStid(),student.getStname(),student.getStaddress());
    }

    public int updateStudent(Student student) {
        String sql = "update student set stname=?,staddress=? where stid=?";
        return template.update(sql,student.getStname(),student.getStaddress(),student.getStid());
    }

    public int deleteStudent(int stid) {
        String sql = "delete from student where stid=?";
        return template.update(sql, stid);
    }

    public Student getStudentById(int stid) {
        String sql ="select * from student where stid=?";
        return template.queryForObject(sql, new StudentRowMapper(), stid);
    }

    public List<Student> getAllStudents() {
        String sql = "select * from student";
        return template.query(sql,new StudentRowMapper());
    }


```

```java
package com.wipro;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import com.wipro.config.Myconfig;
import com.wipro.dao.StudentDAO;
import com.wipro.model.Student;

public class App {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext(Myconfig.class);

        // ☑ Use interface instead of implementation class
        StudentDAO dao = context.getBean(StudentDAO.class);

        /* Uncomment if needed
        int x = dao.saveStudent(new Student(24, "pavan", "kadiri"));
        System.out.println(x + " rows inserted");

        int y = dao.updateStudent(new Student(100, "kalyan", "kdr"));
        System.out.println(y + " rows updated");

        int z = dao.deleteStudent(100);
        System.out.println(z + " rows deleted");
        */

        Student student = dao.getStudentById(100);
        System.out.println(student.getStid() + " " + student.getStname() + " " + student.getStaddress());

        System.out.println("==========");
        List<Student> students=dao.getAllStudents();
        for(Student st:students)
        {
            System.out.println(st.getStid()+""+st.getStname()+""+st.getStaddress());
        }

```

```
100 mohan bagepalli
==========
24kalyankdr
24kalyankdr
24kalyankdr
24kalyankdr
24pavankadiri
24pavankadiri
24pavankadiri
100mohanbagepalli
```

What is Hibernate?

Hibernate is a Java-based ORM (Object-Relational Mapping) framework that simplifies database interactions by mapping Java objects to database tables. It eliminates the need for complex JDBC code, making database operations more efficient and reducing boilerplate code.

✅ **Key Features of Hibernate:**

- **ORM Support** – Maps Java objects to database tables.
- **Eliminates JDBC Boilerplate** – No need for manual SQL queries.
- **Automatic Table Creation** – Can generate database tables from Java classes.
- **HQL (Hibernate Query Language)** – Allows writing database-independent queries.
- **Caching** – Improves performance by storing frequently used data.

What is an ORM Tool?

An **ORM (Object-Relational Mapping) tool** is a framework that helps developers interact with relational databases using object-oriented programming concepts instead of raw SQL queries.

✅ **What ORM Tools Do:**

- Convert **Java objects** into **database tables**.
- Reduce the need for **SQL queries** by allowing CRUD operations using Java methods.
- Provide **database independence** (work with MySQL, PostgreSQL, etc.).
- Improve **security** by preventing SQL injection.

Why Hibernate Came into Picture?

Before Hibernate, developers used **JDBC (Java Database Connectivity)** for database operations, which had several limitations:

🔴 **Problems in JDBC**	✅ **Hibernate Solutions**
Complex SQL queries	Uses HQL (Hibernate Query Language)
Boilerplate code (Connection, Statement, ResultSet)	Simple object-based queries
Manual handling of result sets	Maps Java objects to tables automatically
Vendor-dependent SQL (MySQL, Oracle, etc.)	Database-independent queries
No caching mechanism	Built-in caching for better performance

🎯 Summary

1️⃣ **Hibernate** is a powerful **ORM tool** for Java that simplifies database operations.
2️⃣ **ORM tools** bridge the gap between object-oriented programming and relational databases.

3️⃣ **Hibernate replaced JDBC** because it reduced complexity, minimized SQL dependency, and improved performance.

Difference Between Hibernate and JDBC

Hibernate and JDBC (Java Database Connectivity) are both used to interact with databases, but they have significant differences in terms of abstraction, ease of use, and performance.

Feature	JDBC (Java Database Connectivity)	Hibernate (ORM Framework)
Definition	A low-level API that allows direct interaction with databases using SQL.	An Object-Relational Mapping (ORM) framework that automates SQL query generation using Java objects.
SQL Dependency	Requires manual SQL queries (e.g., `SELECT`, `INSERT`, `UPDATE`, `DELETE`).	Eliminates the need to write SQL queries manually. Uses HQL (Hibernate Query Language).
Database Independence	Tied to a specific database; changing databases requires updating SQL queries.	Supports multiple databases by simply changing the Hibernate dialect.
Boilerplate Code	Requires extensive boilerplate code for connection management, queries, and exception handling.	Reduces boilerplate code by handling everything internally.
Performance	Slower due to repetitive database calls and manual query execution.	Optimized performance using caching, lazy loading, and automatic query optimization.
Scalability	Suitable for small applications but difficult to manage in large applications.	Designed for large-scale enterprise applications.
Transaction Management	Requires manual transaction handling (`commit`, `rollback`).	Built-in transaction management.
Caching Support	No caching; repeated queries can impact performance.	Supports first-level and second-level caching for better performance.

Relationships & Mapping	Complex to handle relationships like One-to-Many and Many-to-Many.	Provides easy-to-use annotations like `@OneToMany`, `@ManyToOne`, etc.
Automatic Table Generation	No support for automatic table creation.	Supports table creation using `hibernate.hbm2ddl.auto`.

Example Comparison

Using JDBC

Copy Edit

```java
public class JDBCDemo {
    public static void main(String[] args) {
        try {
            // 1. Load the Driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2. Establish Connection
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test",

            // 3. Create Statement
            String sql = "INSERT INTO students (id, name, city) VALUES (?, ?, ?)";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, 1);
            pstmt.setString(2, "Pavan");
            pstmt.setString(3, "Kadiri");

            // 4. Execute Query
            int rows = pstmt.executeUpdate();
            System.out.println(rows + " record inserted");

            // 5. Close Connection
            pstmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Using Hibernate

```java
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.wipro.model.Student;
import com.wipro.util.HibernateUtil;

public class HibernateDemo {
    public static void main(String[] args) {
        // Get Hibernate Session
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction tx = session.beginTransaction();

        // Create Student Object
        Student student = new Student("Pavan", "Kadiri");

        // Save Student to Database
        session.save(student);
        tx.commit();

        System.out.println("Student Saved Successfully!");
        session.close();
    }
}
```

Why Choose Hibernate Over JDBC?

1. **Less Code** – Hibernate reduces boilerplate code significantly.
2. **Automatic Table Creation** – Hibernate can generate tables based on Java classes.
3. **Database Independent** – Works with MySQL, PostgreSQL, Oracle, etc.
4. **Better Performance** – Uses caching and optimizations to improve performance.
5. **Transaction Handling** – Built-in transaction management.

When to Use JDBC?

- If you need **direct database interaction** and a **simple, lightweight application**.

- When **performance is critical**, and you want to avoid the extra overhead of an ORM framework.

When to Use Hibernate?

- For **enterprise applications** where **scalability, maintainability, and portability** are required.
- When working with **complex relationships and object-oriented programming**.

Conclusion

- **JDBC** is good for small applications requiring raw SQL.
- **Hibernate** is ideal for larger, scalable applications with complex database interactions.

22-02-2025

Hiberante

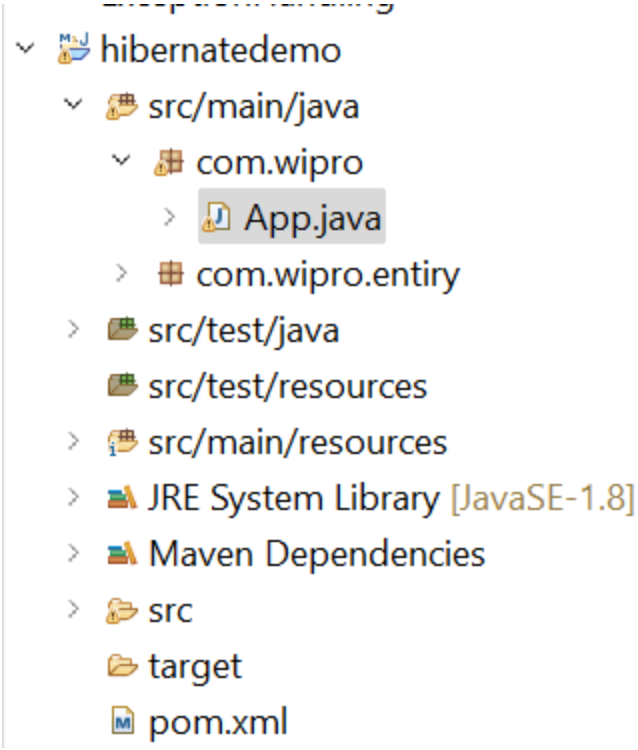
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.wipro</groupId>
  <artifactId>hibernatedemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>hibernatedemo</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1k</version>
      <scope>test</scope>
    </dependency>


    <!-- Jakarta Persistence API -->
<dependency>
    <groupId>jakarta.persistence</groupId>
    <artifactId>jakarta.persistence-api</artifactId>
    <version>3.1.0</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.2.0.Final</version>
</dependency>


<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.0.33</version>
</dependency>



  </dependencies>
</project>

```

hibernatedemo/pom.xml | hibernate.cfg.xml | Employee.java | App.java

```xml
<hibernate-configuration>
    <session-factory>
        <!-- Database Connection Settings -->
        <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/wipro</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">#Mahadev7</property>

        <!-- Hibernate Dialect -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- Show SQL Output -->
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">create</property>

        <!-- Mapped Entity Classes -->
        <mapping class="com.wipro.entiry.Employee"/>
    </session-factory>
</hibernate-configuration>
```

```java
package com.wipro.entiry;

import jakarta.persistence.*;

@Entity
@Table(name = "employee") // Match with your actual DB table name
public class Employee {

    @Id
    // @GeneratedValue(strategy = GenerationType.IDENTITY) // REMOVE OR COMMENT THIS
    private int id;

    private String name;

    public Employee() {
    }

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    // Getters and Setters
    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;
    }
}
```

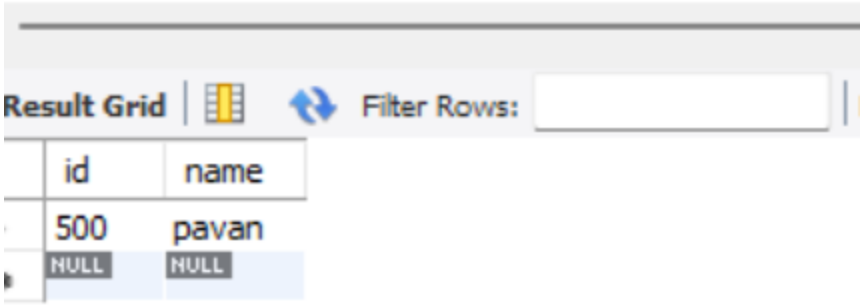
```java
package com.wipro;

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

import com.wipro.entiry.Employee;

public class App {
    public static void main(String[] args) {
        // Create Configuration object and configure Hibernate
        Configuration config = new Configuration();
        config.configure("hibernate.cfg.xml"); // Load hibernate.cfg.xml

        // Add annotated class
        config.addAnnotatedClass(Employee.class);

        // Build SessionFactory
        SessionFactory sessionFactory = config.buildSessionFactory();


        // Create Employee object
        Employee employee = new Employee(500, "pavan");

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

        // Use save() instead of persist()
        session.save(employee);


        /*update
        Employee empl = session.get(Employee.class, 500);
        if (empl != null) {
            empl.setName("Pavan Kalyan"); // Updating name field
            session.update(empl); // Save changes
            System.out.println("Employee updated successfully!");
        } else {
            System.out.println("Employee not found!");
        }*/


        transaction.commit();
        session.close();
        sessionFactory.close();
    }
}
```




```
Feb 22, 2025 1:34:54 PM org.hibernate.resource.transaction.backend.jdbc.internal.
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.e
Hibernate:
    create table employee (
        id integer not null,
        name varchar(255),
        primary key (id)
    ) engine=InnoDB
Feb 22, 2025 1:34:54 PM org.hibernate.resource.transaction.backend.jdbc.internal.
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.e
Feb 22, 2025 1:34:54 PM org.hibernate.tool.schema.internal.exec.GenerationTargetT
INFO: HHH000476: Executing script '[injected ScriptSourceInputNonExistentImpl scr
Hibernate:
    insert
    into
        employee
        (name,id)
    values
        (?,?)
Feb 22, 2025 1:34:54 PM org.hibernate.engine.jdbc.connections.internal.DriverMana
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/wipro
```

Automatically the table and data is inserted

```
1 •  use wipro;
2 •  select * from employee;
```



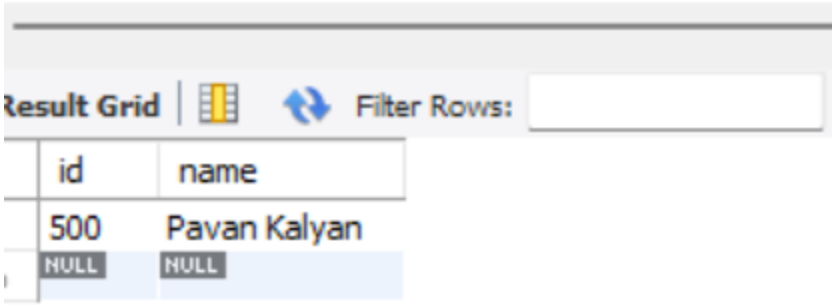
hibernatedemo/pom.xml | hibernate.cfg.xml | Employee.java | App.java

```java
import org.hibernate.cfg.Configuration;

import com.wipro.entiry.Employee;

public class App {
    public static void main(String[] args) {
        // Create Configuration object and configure Hibernate
        Configuration config = new Configuration();
        config.configure("hibernate.cfg.xml"); // Load hibernate.cfg.xml

        // Add annotated class
        config.addAnnotatedClass(Employee.class);

        // Build SessionFactory
        SessionFactory sessionFactory = config.buildSessionFactory();


        // Create Employee object
       Employee employee = new Employee(500, "pavan");

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

        // Use save() instead of persist()
       session.save(employee);


        //update
        Employee empl = session.get(Employee.class, 500);
        if (empl != null) {
            empl.setName("Pavan Kalyan"); // Updating name field
            session.update(empl); // Save changes
            System.out.println("Employee updated successfully!");
        } else {
            System.out.println("Employee not found!");
        }


        transaction.commit();
        session.close();
        sessionFactory.close();
    }
}
```

Update

```
INFO: HHH000490: Using JtaPlatform implementation: [o
Hibernate:
    drop table if exists employee
Feb 22, 2025 1:38:36 PM org.hibernate.resource.transa
INFO: HHH10001501: Connection obtained from JdbcConne
Hibernate:
    create table employee (
        id integer not null,
        name varchar(255),
        primary key (id)
    ) engine=InnoDB
Feb 22, 2025 1:38:36 PM org.hibernate.resource.transa
INFO: HHH10001501: Connection obtained from JdbcConne
Feb 22, 2025 1:38:36 PM org.hibernate.tool.schema.int
INFO: HHH000476: Executing script '[injected ScriptSo
Employee updated successfully!
Hibernate:
    insert
    into
        employee
        (name,id)
    values
        (?,?)
Hibernate:
    update
        employee
    set
        name=?
    where
        id=?
Feb 22, 2025 1:38:36 PM org.hibernate.engine.jdbc.con
```

```
1  use wipro;
2  select * from employee;
```

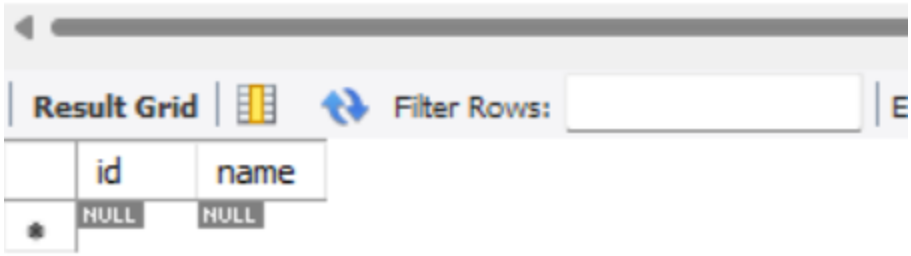


//delete

```
42
43        //delete
44        int employeeId = 500; // The ID of the employee you want to delet
45        Employee empl2 = session.get(Employee.class, employeeId);
46
47        if (empl2 != null) {
48            session.delete(employee); // Delete employee record
49            System.out.println("Employee deleted successfully!");
50        } else {
51            System.out.println("Employee not found!");
52        }
53
54
```

```
INFO: HHH000476: Executing script
Employee deleted successfully!
Hibernate:
    insert
    into
        employee
        (name,id)
    values
        (?,?)
Hibernate:
    delete
    from
        employee
    where
        id=?
Feb 22, 2025 2:00:20 PM org.hibern
```

```
1 •  use wipro;
2 •  select * from employee;
```



To insert data we r using the **persistent method**

To update the data we r using the **merge method**

To delete the data we r using the **remove method**

Hibernate Object States

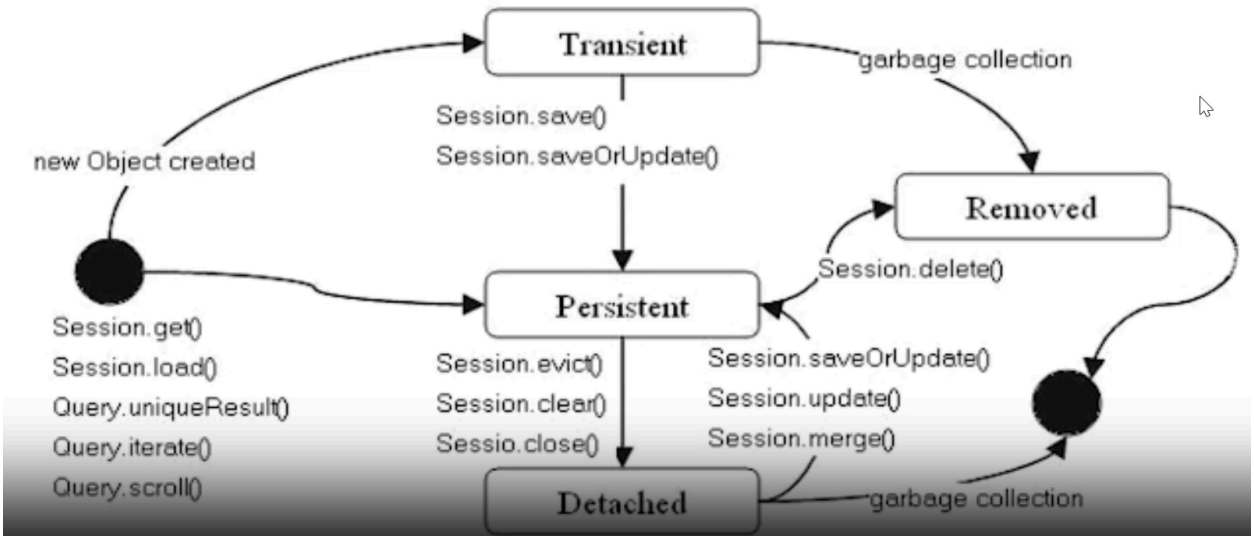
In hibernate every **entity** can be there in either of these 4 states

Hibernate Object States (Transient, Persistent, Detached, Removed) with Example

In **Hibernate**, an entity object can exist in one of four states:

1. **Transient** – Object is not associated with Hibernate.
2. **Persistent** – Object is associated with Hibernate and tracked.
3. **Detached** – Object was associated but is now disconnected.
4. **Removed** – Object is scheduled for deletion.

Entity Lifecycle and States



1 Transient State

An object is in the **transient** state when:

- ✅ It is created using `new`, but **not yet saved** in the databas
- ✅ It has no **persistent identity** (primary key not assigned).
- ✅ Hibernate does **not track** it.

When the object very newly that is associated with the java environment but not associated with the database then we can call it is in the Transistent state

```
Employee employee = new Employee();  // Transient state
employee.setId(101);
employee.setName("RK");
// Object is not associated with Hibernate yet
```

📌 At this point, the object exists in Java but not in the database.

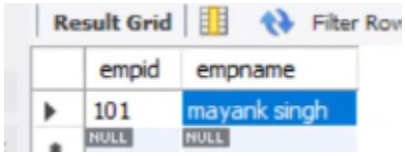
2 Persistent State

An object is in the **persistent** state when:

- ✅ It is associated with a **Hibernate session**.
- ✅ Hibernate **tracks** changes and syncs them with the database.
- ✅ Changes to the object are automatically saved **without calling save()** (Dirty Checking).

```
SessionFactory sessionFactory = config.buildSessionFactory()
Session session = sessionFactory.openSession();

Transaction transaction = session.beginTransaction();

Employee employee = new Employee(101,"mayank");//transient s

session.persist(employee);//persistent state

String empname
employee.setEmpname("mayank singh");

//session.merge(employee);
transaction.commit();
session.close();
sessionFactory.close();
System.out.println("updated successfully(dirty ch");
```

```
Hibernate:
    update
        Employee
    set
        empname=?
    where
        empid=?
Feb 22, 2025 2:36:51 PM org.hibernate.e
INFO: HHH10001008: Cleaning up connecti
updated successfully(dirty checking)
```



3 Detached State

An object is in the **detached** state when:

- ✅ It was **persistent**, but the session is **closed**.
- ✅ Hibernate **no longer tracks** the object.
- ✅ You need `merge()` to update changes in the database.

```java
Session session1 = sessionFactory.openSession();
Transaction transaction1 = session1.beginTransaction();

Employee employee = session1.get(Employee.class, 101);  // Persistent
session1.close();  // Now Detached

employee.setName("RK Detached Updated");  // No effect on DB
```

Removed State

An object is in the **removed** state when:

- ✅ It is **scheduled for deletion** in Hibernate.
- ✅ The object still exists in Java but will be deleted when `commit()` is called.

State	Description	Tracked by Hibernate?	Example Method
Transient	Newly created object, not in DB	❌ No	`new Employee()`
Persistent	Object is in session & synced with DB	✅ Yes	`persist(employee)`, `get()`
Detached	Object was persistent but session closed	❌ No	`merge(employee)`
Removed	Object is marked for deletion	✅ Yes (until commit)	`remove(employee)`

Executing a SELECT Query in Hibernate

In Hibernate, we can retrieve data using:

1. **session.get() / session.find()** – Fetch a single entity by primary key.
2. **session.createQuery()** – Execute **HQL (Hibernate Query Language)** queries.
3. **session.createNativeQuery()** – Execute **SQL queries** directly.
4. **Criteria API** – Dynamic query building (Hibernate 6 uses `CriteriaBuilder`).

Get method

```
 // Create Employee object
Employee employee = new Employee(500, "pavan"); //transistent

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

 // Use save() instead of persist()
session.persist(employee); //persistent state


//by using the get method
Employee empl = session.get(Employee.class, 502);
System.out.println(empl);    //returns null if the data is not there
```

```
null
```

Find method

```
 // Create Employee object
Employee employee = new Employee(500, "pavan"); //transistent

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

 // Use save() instead of persist()
session.persist(employee); //persistent state


//by using the get method
Employee empl = session.find(Employee.class, 502);
System.out.println(empl);    //returns null if the data is not there
```

```
null
```

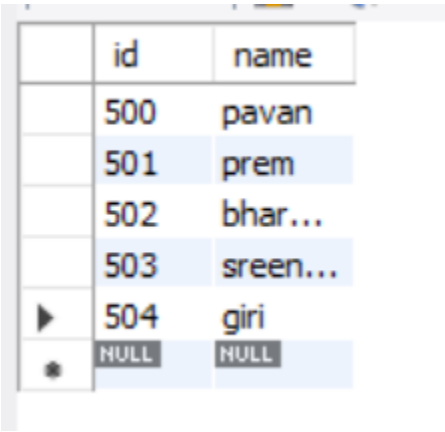
In the hibernate older versions it is giving that

get=null

find=exception

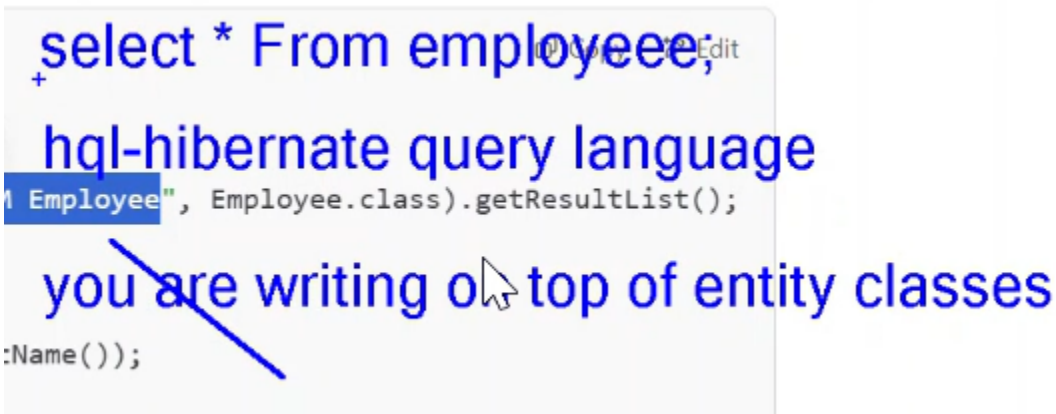
But now in the newer versions of the hibernate it is displaying both the null values

How to retrieve the all the rows from the table by using the hibernate



	id	name
	500	pavan
	501	prem
	502	bhar...
	503	sreen...
▶	504	giri
*	NULL	NULL

Now by using the **createQuery** we r going to fetch all the data from the table



```
  List<Employee> employees = session.createQuery("FROM Employee", Employee.class).getResultList();
        for(Employee emp:employees) {
            System.out.println(emp.getId()+" "+emp.getName());
        }
        transaction.commit();
        session.close();
        sessionFactory.close();
    }
}
```

```
22 prem
100 pavan
```

	id	name
	22	prem
▶	100	pavan
*	NULL	NULL

List<Employee> employees = session.createQuery("FROM Employee", Employee.**class**).getResultList();

This is a HQL query :HQL automatically getting converted into the sql internally

We can use the **where** condition also

CreateNativeQuery:used to write the sql queries in the hibernate