

```
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 getSname() {  
    return sname;  
}  
  
public void setSname(String sname) {  
    this.sname = sname;  
}  
  
public String getStaddress() {  
    return staddress;  
}  
  
public void setStaddress(String staddress) {  
    this.staddress = staddress;  
}  
  
public Student(Integer stid, String sname, String staddress) {  
    super();  
    this.stid = stid;  
    this.sname = sname;  
    this.staddress = staddress;  
}
```

@Override

```
public String toString()  
{  
    return stid + " " + sname + " " + staddress;  
}
```

```

    }

    public Student()

    {

    }

}

```

```

<dependency>          <groupId>org.springframework... 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.

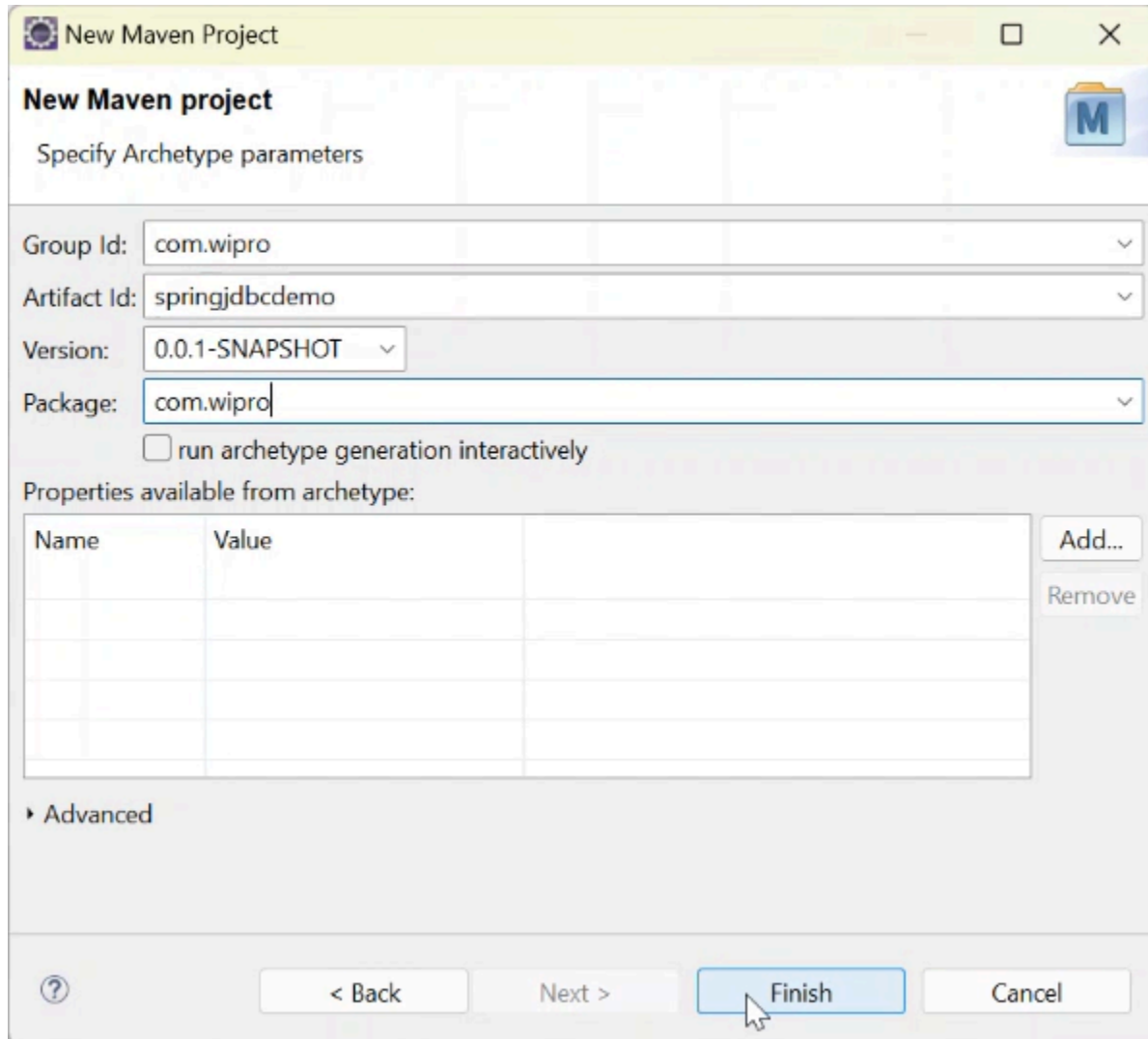
BOILER PLATE CODE

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>



New Maven Project

Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

☐ run archetype generation interactively

Properties available from archetype:

Name	Value

Advanced

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>
```

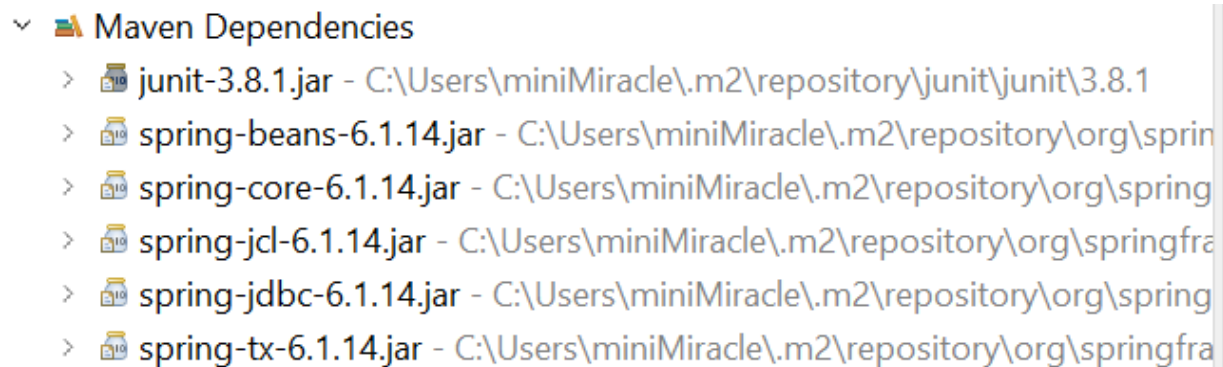
```
    <artifactId>spring-jdbc</artifactId>
```

```
    <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 default 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

```

1 package com.wipro.daoimpl;
2
3 import com.wipro.dao.StudentDAO;
4
5
6 public class StudentDAOImpl implements StudentDAO {
7
8     @Override
9     public int saveStudent(Student student) {
10         // TODO Auto-generated method stub
11         return 0;
12     }
13
14 }
15

```

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

For that we need to add the spring context maven dependency

```

<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

@repo: marks the class as a data access object

```

1 package com.wipro.daoimpl;
2
3 import org.springframework.jdbc.core.JdbcTemplate;
4 import org.springframework.stereotype.Repository;
5
6 import com.wipro.dao.StudentDAO;
7 import com.wipro.model.Student;
8 @Repository
9 public class StudentDAOImpl implements StudentDAO {
10
11     private JdbcTemplate template;
12     public int saveStudent(Student student) {
13
14         String sql="insert into student values(?,??)";
15         return template.update(sql, student.getStid(),student.getStname(),student.getStaddress());
16     }
17 }
18
19 }
20

```

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

```

1 package com.wipro.model;
2
3 public class Student {
4
5     private Integer stid;
6     private String stname;
7     private String staddress;
8     public Integer getStid() {
9         return stid;
10    }
11    public void setStid(Integer stid) {
12        this.stid = stid;
13    }
14    public String getStname() {
15        return stname;
16    }
17    public void setStname(String stname) {
18        this.stname = stname;
19    }
20    public String getStaddress() {
21        return staddress;
22    }
23    public void setStaddress(String staddress) {
24        this.staddress = staddress;
25    }
26
27    public Student(Integer stid, String stname, String staddress) {
28        super();
29        this.stid = stid;
30        this.stname = stname;
31        this.staddress = staddress;
32    }
33    @Override
34    public String toString()
35    {
36        return stid + " " + stname + " " + staddress;
37    }
38    public Student()
39    {
40
41    }
42
43 }

```

```

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

```

```

springjdbcdemo/pom.xml Student.java StudentDAO.java StudentDAOimpl.java < IvyConfig.java
1 package com.wipro.daoimpl;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.jdbc.core.JdbcTemplate;
5 import org.springframework.stereotype.Repository;
6
7 import com.wipro.dao.StudentDAO;
8 import com.wipro.model.Student;
9 @Repository
10 public class StudentDAOImpl implements StudentDAO {
11
12     @Autowired
13     private JdbcTemplate template;
14     public int saveStudent(Student student) {
15
16         String sql="insert into student(stid,stname,staddress) values(?,?,?)";
17         return template.update(sql, student.getStid(),student.getStname(),student.getStaddress());
18
19     }
20
21 }
22

```

Autowired:Used to inject the beans automatically

```






















1 package com.wipro.config;
2
3 import javax.sql.DataSource;
4
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.ComponentScan;
7 import org.springframework.context.annotation.Configuration;
8 import org.springframework.jdbc.core.JdbcTemplate;
9 import org.springframework.jdbc.datasource.DriverManagerDataSource;
10
11 @Configuration
12 @ComponentScan(basePackages = "com.wipro")
13
14 public class Myconfig
15 {
16
17     @Bean
18     public DataSource dataSource() {
19         DriverManagerDataSource dataSource = new DriverManagerDataSource();
20         dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
21         dataSource.setUrl("jdbc:mysql://localhost:3306/wipro");
22         dataSource.setUsername("root");
23         dataSource.setPassword("#Mahadev7");
24         return dataSource;
25     }
26
27     @Bean
28     public JdbcTemplate jdbcTemplate(DataSource dataSource) {
29         return new JdbcTemplate(dataSource);
30     }
31
32 }
33

```

@configuration=replaces xml based configuration

@componentscan=look for the component,service,repository etc


```
1 package com.wipro;
2
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6 import com.wipro.config.Myconfig;
7 import com.wipro.daoimpl.StudentDAOImpl;
8 import com.wipro.model.Student;
9
10 /**
11  * Hello world!
12  *
13  */
14 public class App
15 {
16     public static void main( String[] args )
17     {
18         ApplicationContext context=new AnnotationConfigApplicationContext(Myconfig.class);
19         StudentDAOImpl dao=context.getBean(StudentDAOImpl.class);
20
21         int x= dao.saveStudent(new Student(24,"pavan","kadiri"));
22         System.out.println(x + "rows inserted");
23     }
24 }
25
```

- ✓  springjdbcdemo
 - ✓  src/main/java
 - ✓  com.wipro
 - >  App.java
 - ✓  com.wipro.config
 - >  Myconfig.java
 - ✓  com.wipro.dao
 - >  StudentDAO.java
 - ✓  com.wipro.daoimpl
 - >  StudentDAOImpl.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.jar
 - >  micrometer-observation-1.12.11.jar
 - >  mysql-connector-j-8.0.33.jar - C:\
 - >  protobuf-java-3.21.9.jar - C:\User
 - >  spring-aop-6.1.14.jar - C:\Users\

springjdbcdemo/pom.xml Student.java StudentDAO.java StudentDAOImpl.java

```

1 http://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation with catalog)
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5
6   <groupId>com.wipro</groupId>
7   <artifactId>springjdbcdemo</artifactId>
8   <version>0.0.1-SNAPSHOT</version>
9   <packaging>jar</packaging>
10
11   <name>springjdbcdemo</name>
12   <url>http://maven.apache.org</url>
13
14   <properties>
15     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16   </properties>
17
18   <dependencies>
19     <dependency>
20       <groupId>junit</groupId>
21       <artifactId>junit</artifactId>
22       <version>3.8.1</version>
23       <scope>test</scope>
24     </dependency>
25
26     <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
27     <dependency>
28       <groupId>org.springframework</groupId>
29       <artifactId>spring-jdbc</artifactId>
30       <version>6.1.14</version>
31     </dependency>
32
33     <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
34     <dependency>
35       <groupId>mysql</groupId>
36       <artifactId>mysql-connector-java</artifactId>
37       <version>8.0.33</version>
38     </dependency>
39
40
41     <dependency>
42       <groupId>org.springframework</groupId>
43       <artifactId>spring-context</artifactId>
44       <version>6.1.14</version>
45     </dependency>
46

```

```

39
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

```

terminated: app v.
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 x

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

```
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     int updateStudent(Student student);
10
11     int deleteStudent(Student student);
12
13 }
14
```

```

1 package com.wipro.daoimpl;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.jdbc.core.JdbcTemplate;
5 import org.springframework.stereotype.Repository;
6
7 import com.wipro.dao.StudentDAO;
8 import com.wipro.model.Student;
9 @Repository
10 public abstract class StudentDAOImpl implements StudentDAO {
11
12     @Autowired
13     private JdbcTemplate template;
14     public int saveStudent(Student student) {
15
16         String sql="insert into student(stdid,stname,staddress) values(?,?,?)";
17         return template.update(sql, student.getStdid(),student.getStname(),student.getStaddress());
18     }
19
20
21     public int updateStudent(Student student) {
22
23         String sql="update student set stname=?, staddress=? where stdid=?";
24         return template.update(sql, student.getStname(),student.getStaddress(),student.getStdid());
25     }
26
27     public int deleteStudent(int stdid) {
28         String sql = "delete from student where stdid=?";
29         return template.update(sql, stdid);
30     }
31
32
33 }
34
35

```

```

1 package com.wipro;
2
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6 import com.wipro.config.Myconfig;
7 import com.wipro.daoimpl.StudentDAOImpl;
8 import com.wipro.model.Student;
9
10 /**
11  * Hello world!
12  *
13  */
14 public class App
15 {
16     public static void main( String[] args )
17     {
18         ApplicationContext context=new AnnotationConfigApplicationContext(Myconfig.class);
19         StudentDAOImpl dao=context.getBean(StudentDAOImpl.class);
20
21         int x= dao.saveStudent(new Student(24,"pavan","kadiri"));
22         System.out.println(x + "rows inserted");
23
24
25         int y= dao.updateStudent(new Student(100,"kalyan","kdr"));
26         System.out.println(y + "rows updated");
27
28         int z = dao.deleteStudent(100);
29         System.out.println(z + " rows deleted");
30
31     }
32 }
33

```

	stdid	sname	staddress
	24	kalyan	kdr
	24	kalyan	kdr
	24	kalyan	kdr
	24	pavan	kadiri
	24	pavan	kadiri
	24	pavan	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.

1 `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 <code>Student</code> object
0 rows found	✗ Throws <code>EmptyResultDataAccessException</code>
More than 1 row	✗ Throws <code>IncorrectResultSizeDataAccessException</code>

2 `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.getSname(),student.getStaddress());
```

```
    }
```

```
    public int updateStudent(Student student) {
```

```
        String sql = "update student set sname=?,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.sql... 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 x StudentDAOIm... Myconfig.ja
1 package com.wipro.dao;
2
3 import java.util.List;
4
5 import com.wipro.model.Student;
6
7 public interface StudentDAO
8 {
9     int saveStudent(Student student);
10
11     int updateStudent(Student student);
12
13     int deleteStudent(Student student);
14
15     Student getStudentById(int stid);
16     List<Student> getAllStudents();
17
18 }
19

```

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

```

1 package com.wipro.daoimpl;
2
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5
6 import org.springframework.jdbc.core.RowMapper;
7
8 import com.wipro.model.Student;
9
10 public class StudentRowMapper implements RowMapper<Student> {
11
12     public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
13         return new Student(rs.getInt("stid"),rs.getString("stname"),rs.getString("staddress"));
14     }
15
16 }

```

Above code is from StudentDAOImple

Now we need to implement the rowmapper class


```

1 package com.wipro.daoimpl;
2
3 import java.util.List;
11
12 @Repository(value = "student")
13 public class StudentDAOImpl implements StudentDAO {
14
15     @Autowired
16     private JdbcTemplate template;
17     public int saveStudent(Student student) {
18
19         String sql = "insert into student values(?,?,?)";
20         return template.update(sql, student.getStid(), student.getStname(), student.getStaddress());
21     }
22
23     public int updateStudent(Student student) {
24         String sql = "update student set stname=?, staddress=? where stid=?";
25         return template.update(sql, student.getStname(), student.getStaddress(), student.getStid());
26     }
27
28     public int deleteStudent(int stid) {
29         String sql = "delete from student where stid=?";
30         return template.update(sql, stid);
31     }
32
33     public Student getStudentById(int stid) {
34         String sql = "select * from student where stid=?";
35         return template.queryForObject(sql, new StudentRowMapper(), stid);
36     }
37
38     public List<Student> getAllStudents() {
39         String sql = "select * from student";
40         return template.query(sql, new StudentRowMapper());
41     }
42
43

```

```

1 package com.wipro;
2 import java.util.List;
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5 import com.wipro.config.Myconfig;
6 import com.wipro.dao.StudentDAO;
7 import com.wipro.model.Student;
8
9 public class App {
10     public static void main(String[] args) {
11         ApplicationContext context = new AnnotationConfigApplicationContext(Myconfig.class);
12
13         // Use interface instead of implementation class
14         StudentDAO dao = context.getBean(StudentDAO.class);
15
16         /* Uncomment if needed
17         int x = dao.saveStudent(new Student(24, "pavan", "kadiri"));
18         System.out.println(x + " rows inserted");
19
20         int y = dao.updateStudent(new Student(100, "kalyan", "kdr"));
21         System.out.println(y + " rows updated");
22
23         int z = dao.deleteStudent(100);
24         System.out.println(z + " rows deleted");
25         */
26
27         Student student = dao.getStudentById(100);
28         System.out.println(student.getStid() + " " + student.getStname() + " " + student.getStaddress());
29
30         System.out.println("=====");
31         List<Student> students=dao.getAllStudents();
32         for(Student st:students)
33         {
34             System.out.println(st.getStid()+" "+st.getStname()+" "+st.getStaddress());
35         }
36

```

```

terminated: App (1) pava / p...
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

Complex SQL queries

Boilerplate code (Connection, Statement, ResultSet)

Manual handling of result sets

Vendor-dependent SQL (MySQL, Oracle, etc.)

No caching mechanism

✓ Hibernate Solutions

Uses HQL (Hibernate Query Language)

Simple object-based queries

Maps Java objects to tables automatically

Database-independent queries

Built-in caching for better performance

🎯 Summary

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

③ **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., <code>SELECT</code> , <code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code>).	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 (<code>commit</code> , <code>rollback</code>).	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 <code>@OneToMany</code> , <code>@ManyToOne</code> , etc.
Automatic Table Generation	No support for automatic table creation.	Supports table creation using <code>hibernate.hbm2ddl.auto</code> .

Example Comparison

Using JDBC

```
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


```
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

```
1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3  <modelVersion>4.0.0</modelVersion>
4
5  <groupId>com.wipro</groupId>
6  <artifactId>hibernatedemo</artifactId>
7  <version>0.0.1-SNAPSHOT</version>
8  <packaging>jar</packaging>
9
10 <name>hibernatedemo</name>
11 <url>http://maven.apache.org</url>
12
13<properties>
14  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15</properties>
16
17<dependencies>
18  <dependency>
19    <groupId>junit</groupId>
20    <artifactId>junit</artifactId>
21    <version>3.8.1</version>
22    <scope>test</scope>
23  </dependency>
24
25  <!-- Jakarta Persistence API -->
27<dependency>
28  <groupId>jakarta.persistence</groupId>
29  <artifactId>jakarta.persistence-api</artifactId>
30  <version>3.1.0</version>
31</dependency>
32
33<dependency>
34  <groupId>org.hibernate</groupId>
35  <artifactId>hibernate-core</artifactId>
36  <version>6.2.0.Final</version>
37</dependency>
38
39
40<dependency>
41  <groupId>com.mysql</groupId>
42  <artifactId>mysql-connector-j</artifactId>
43  <version>8.0.33</version>
44</dependency>
45
46
47
48
49  </dependencies>
50</project>
51
```


hibernatedemo/pom.xml x Employee.java App.java

```
1 <hibernate-configuration>
2   <session-factory>
3     <!-- Database Connection Settings -->
4     <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
5     <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/wipro</property>
6     <property name="hibernate.connection.username">root</property>
7     <property name="hibernate.connection.password">#Mahadev7</property>
8
9     <!-- Hibernate Dialect -->
10    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
11
12    <!-- Show SQL Output -->
13    <property name="hibernate.show_sql">true</property>
14    <property name="hibernate.format_sql">true</property>
15    <property name="hibernate.hbm2ddl.auto">create</property>
16
17    <!-- Mapped Entity Classes -->
18    <mapping class="com.wipro.entiry.Employee"/>
19  </session-factory>
20 </hibernate-configuration>
21
```

```
1 package com.wipro.entiry;
2
3 import jakarta.persistence.*;
4
5 @Entity
6 @Table(name = "employee") // Match with your actual DB table name
7 public class Employee {
8
9     @Id
10    // @GeneratedValue(strategy = GenerationType.IDENTITY) // REMOVE OR COMMENT THIS
11    private int id;
12
13    private String name;
14
15    public Employee() {
16    }
17
18    public Employee(int id, String name) {
19        this.id = id;
20        this.name = name;
21    }
22
23    // Getters and Setters
24    public int getId() {
25        return id;
26    }
27
28    public void setId(int id) {
29        this.id = id;
30    }
31
32    public String getName() {
33        return name;
34    }
35
36    public void setName(String name) {
37        this.name = name;
38    }
39 }
40
```

```

1 package com.wipro;
2
3 import org.hibernate.Session;
4 import org.hibernate.SessionFactory;
5 import org.hibernate.Transaction;
6 import org.hibernate.cfg.Configuration;
7
8 import com.wipro.entiry.Employee;
9
10 public class App {
11     public static void main(String[] args) {
12         // Create Configuration object and configure Hibernate
13         Configuration config = new Configuration();
14         config.configure("hibernate.cfg.xml"); // Load hibernate.cfg.xml
15
16         // Add annotated class
17         config.addAnnotatedClass(Employee.class);
18
19         // Build SessionFactory
20         SessionFactory sessionFactory = config.buildSessionFactory();
21
22         // Create Employee object
23         Employee employee = new Employee(500, "pavan");
24
25         Session session = sessionFactory.openSession();
26         Transaction transaction = session.beginTransaction();
27
28         // Use save() instead of persist()
29         session.save(employee);
30
31         /*update
32         Employee empl = session.get(Employee.class, 500);
33         if (empl != null) {
34             empl.setName("Pavan Kalyan"); // Updating name field
35             session.update(empl); // Save changes
36             System.out.println("Employee updated successfully!");
37         } else {
38             System.out.println("Employee not found!");
39         }
40         */
41
42         transaction.commit();
43         session.close();
44         sessionFactory.close();
45     }
46 }
47
48
49

```

- ▼ hibernatedemo
 - ▼ src/main/java
 - ▼ com.wipro
 - > App.java
 - > com.wipro.entiry
 - > src/test/java
 - > src/test/resources
 - > src/main/resources
 - > JRE System Library [JavaSE-1.8]
 - > Maven Dependencies
 - > src
 - > target
 - > pom.xml

```

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;
```

Result Grid



Filter Rows:

id	name
500	pavan
NULL	NULL

hibernatedemo/pom.xml hibernate.ctg.xml Employee.java App.java ×

```

6 import org.hibernate.cfg.Configuration;
7
8 import com.wipro.entiry.Employee;
9
10 public class App {
11     public static void main(String[] args) {
12         // Create Configuration object and configure Hibernate
13         Configuration config = new Configuration();
14         config.configure("hibernate.cfg.xml"); // Load hibernate.cfg.xml
15
16         // Add annotated class
17         config.addAnnotatedClass(Employee.class);
18
19         // Build SessionFactory
20         SessionFactory sessionFactory = config.buildSessionFactory();
21
22         // Create Employee object
23         Employee employee = new Employee(500, "pavan");
24
25         Session session = sessionFactory.openSession();
26         Transaction transaction = session.beginTransaction();
27
28         // Use save() instead of persist()
29         session.save(employee);
30
31         //update
32         Employee empl = session.get(Employee.class, 500);
33         if (empl != null) {
34             empl.setName("Pavan Kalyan"); // Updating name field
35             session.update(empl); // Save changes
36             System.out.println("Employee updated successfully!");
37         } else {
38             System.out.println("Employee not found!");
39         }
40
41         transaction.commit();
42         session.close();
43         sessionFactory.close();
44     }
45 }

```

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.idgc con

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

Result Grid |   Filter Rows:

id	name
500	Pavan Kalyan
NULL	NULL

//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: HHH0000476: Executing script

Employee deleted successfully!

Hibernate:

insert

into

employee

(name,id)

values

(?,?)

Hibernate:

delete

from

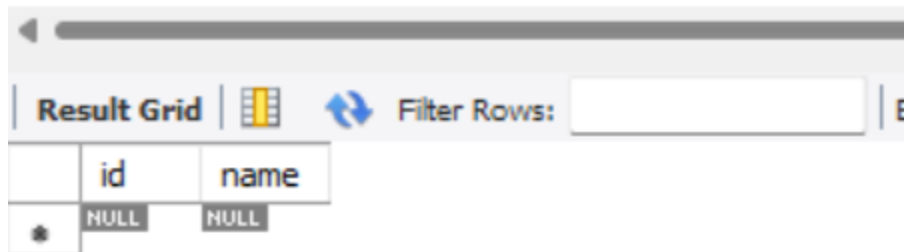
employee

where

id=?

Feb 22 2025 2:00:22 PM org.hibernate

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



The screenshot shows a database client window with a 'Result Grid' tab. The grid has two columns: 'id' and 'name'. The first row contains the values 'NULL' and 'NULL'. There is a 'Filter Rows' input field to the right of the grid.

	id	name
*	NULL	NULL

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

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

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

Hibernate Object States

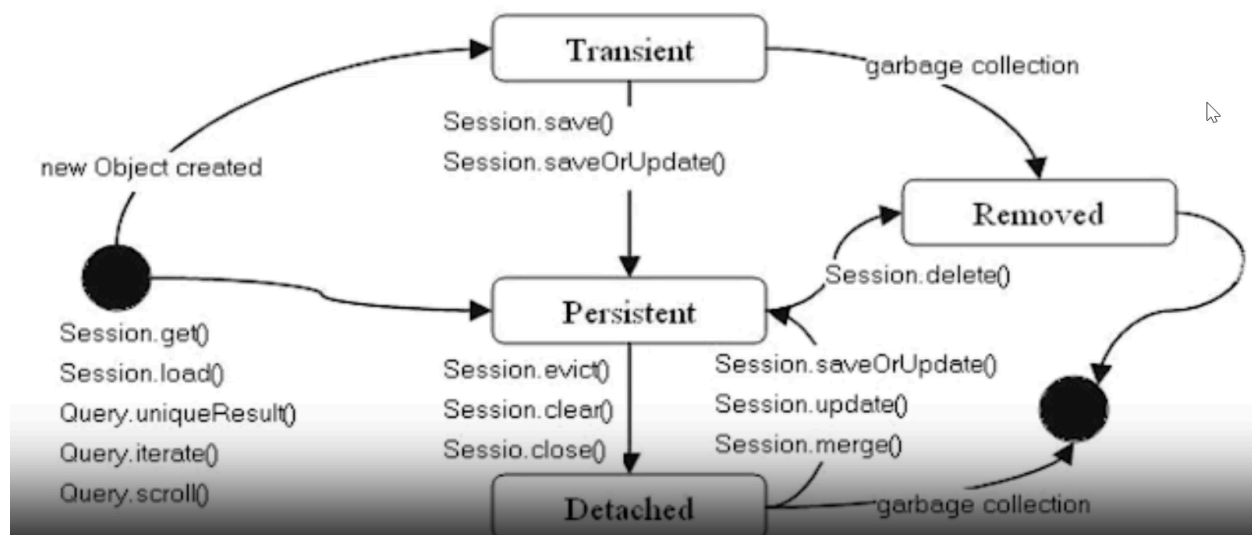
In hibernate every **entity** can be there in either of these 4 states

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 database.
- ✓ 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
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=?

I

Feb 22, 2025 2:36:51 PM org.hibernate.e

INFO: HHH10001008: Cleaning up connecti

updated successfully(dirty checking)

07

Result Grid		Filter Rows
	empid	empname
▶	101	mayank singh
•	NULL	NULL

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.

```

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	<code>new Employee()</code>
Persistent	Object is in session & synced with DB	✓ Yes	<code>persist(employee)</code> , <code>get()</code>
Detached	Object was persistent but session closed	✗ No	<code>merge(employee)</code>
Removed	Object is marked for deletion	✓ Yes (until commit)	<code>remove(employee)</code>

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

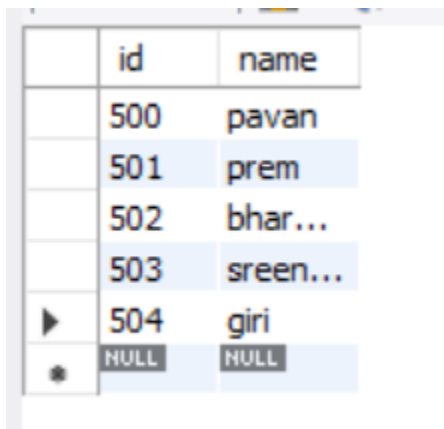
In the hibernate older versions it is giving that

get=null

find=exception

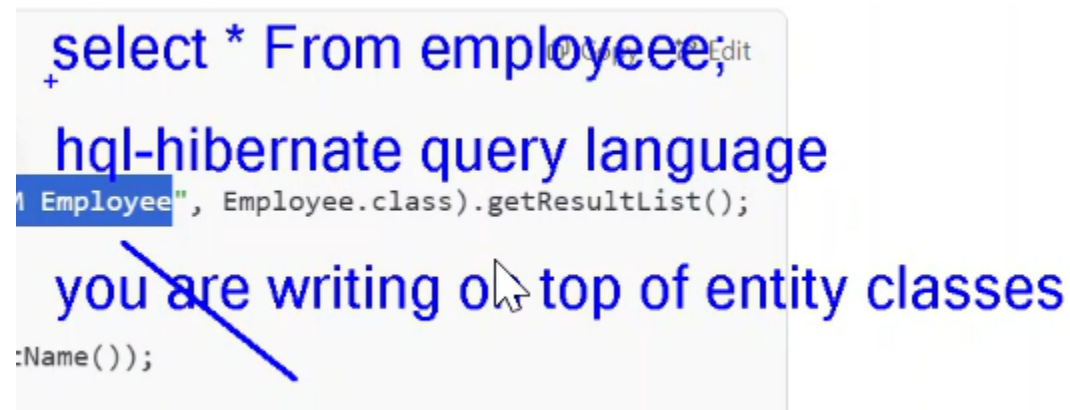
But now in the newer versions of the hibernate it is displaying both the null values

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



```
select * From employee;  
hql-hibernate query language  
Employee", Employee.class).getResultList();  
you are writing o top of entity classes  
:Name());
```



```

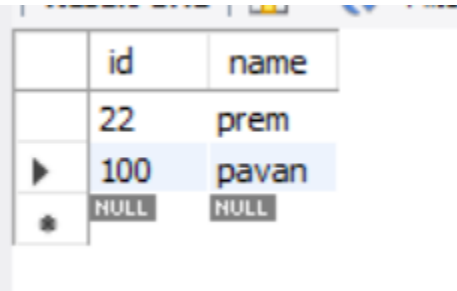
List<Employee> employees = session.createQuery("FROM Employee", Employee.class).getResultList();
for(Employee emp:employees) {
    System.out.println(emp.getId()+" "+emp.getName());
}
transaction.commit();
session.close();
sessionFactory.close();
}
}

```

```

emp
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

```
java                                                                    Copy Edit

Session session = sessionFactory.openSession();
List<Employee> employees = session.createQuery("FROM Employee WHERE name = :empName", Employee.class)
    .setParameter("empName", "RK")
    .getResultList();

for (Employee emp : employees) {
    System.out.println(emp.getId() + " - " + emp.getName());
}

session.close();
```

CreateNativeQuery:used to write the sql queries in the hibernate

<https://chatgpt.com/share/67c86ff0-9540-8000-bf5c-dca64d0c5409>