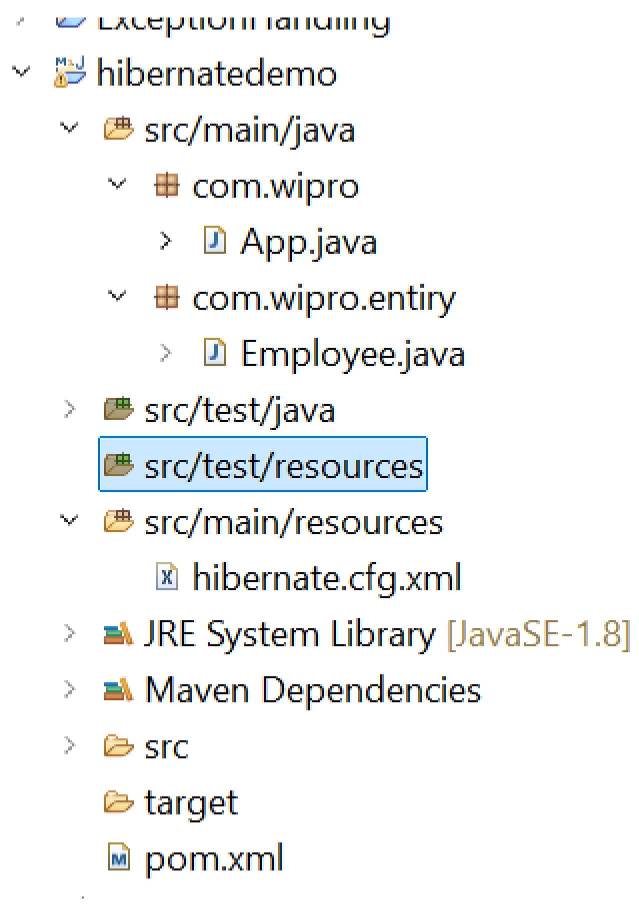
24-02-2025

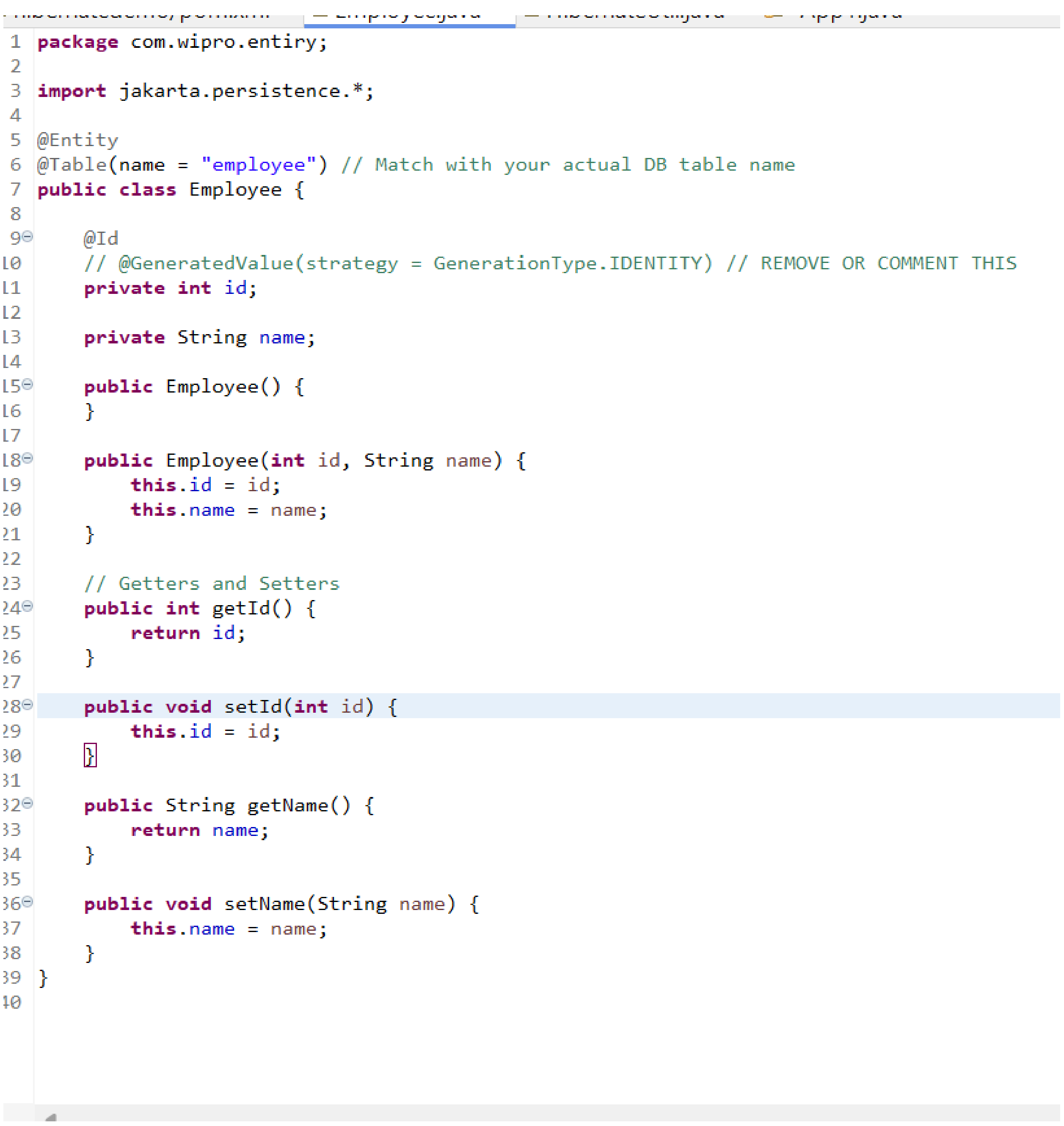
Spring ORM

Without the usage of the xml files,**How to develop the hibernate applications**

With the help of java based configurations

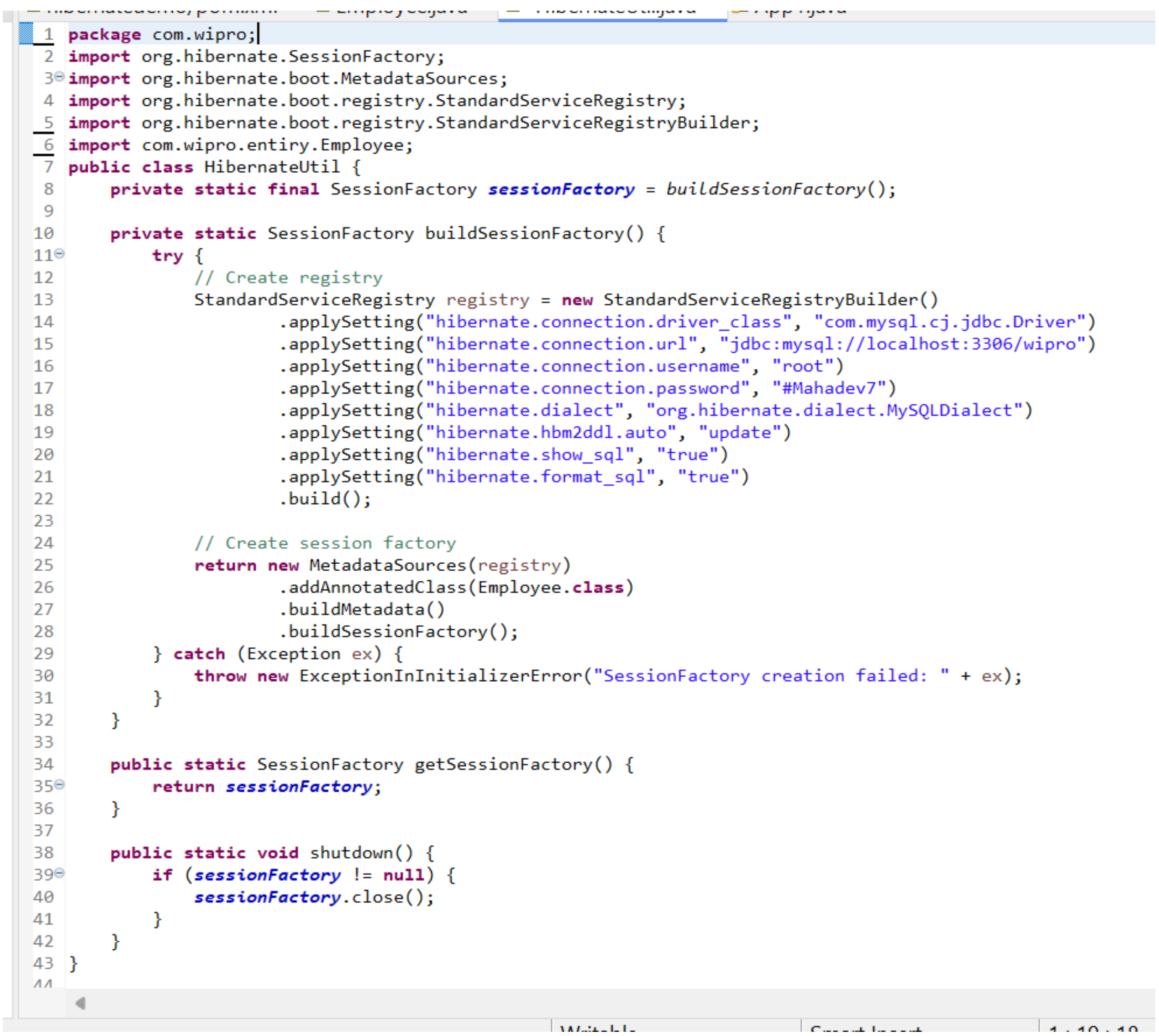


```
hibernatedemo
  src/main/java
    com.wipro
      App.java
    com.wipro.entiry
      Employee.java
  src/test/java
  src/test/resources
  src/main/resources
    hibernate.cfg.xml
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
```



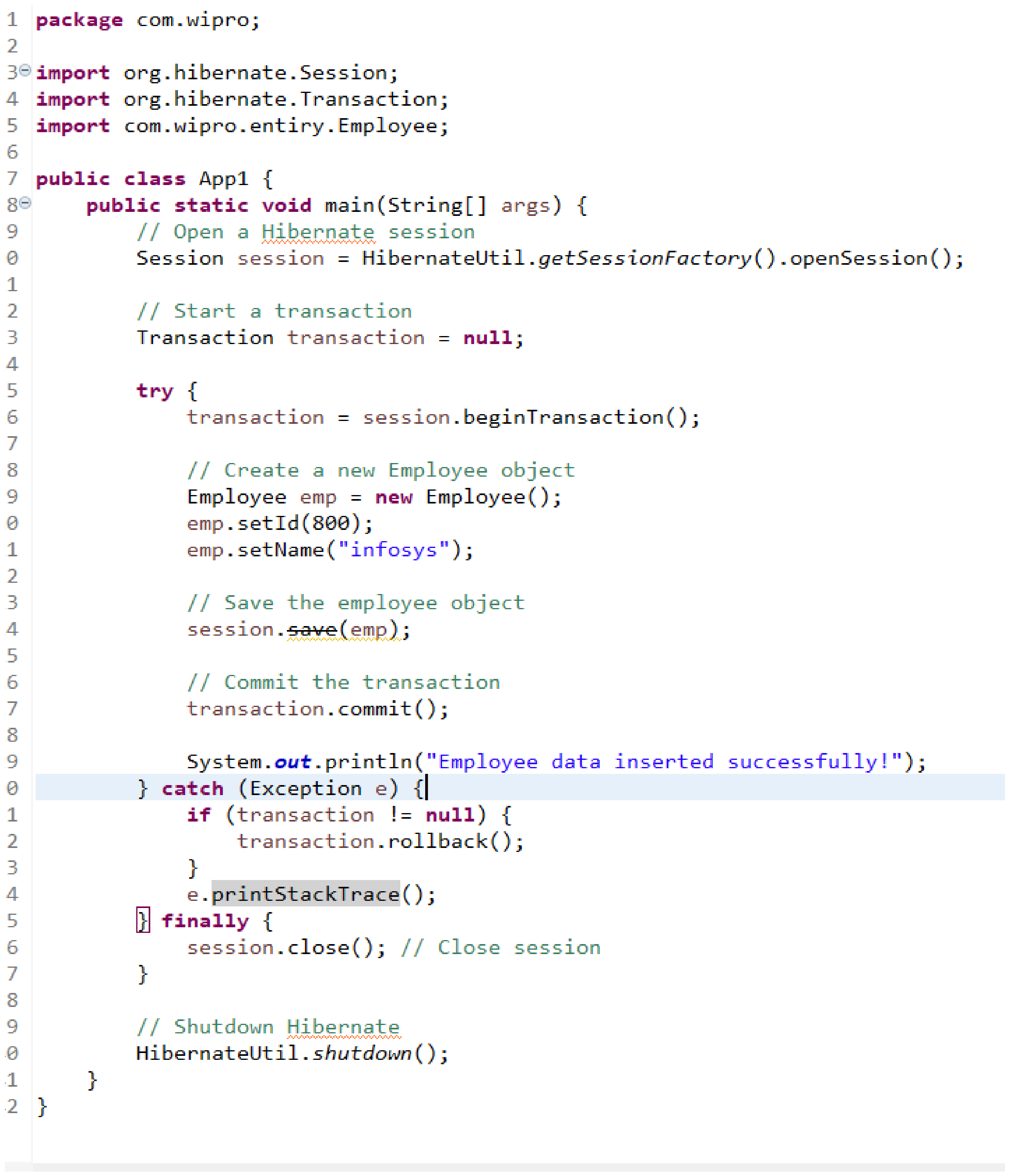
```java
package com.wipro.entiry;

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;
    }
}

```

Now i am going to create one util class named as **HibernateUtil**



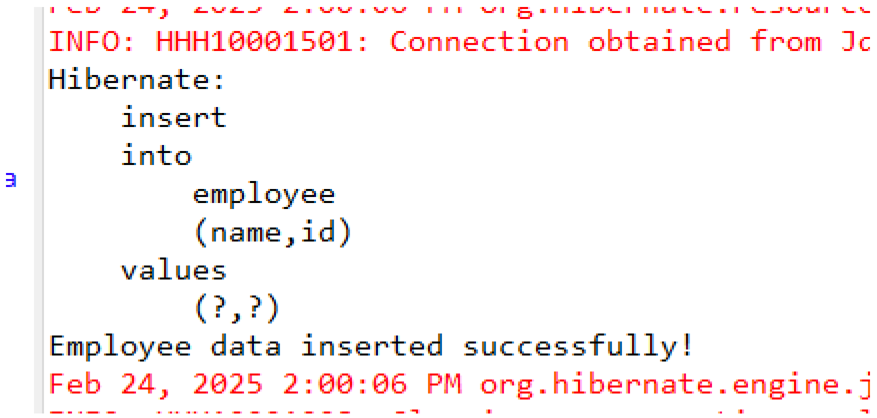
```java
package com.wipro;
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import com.wipro.entiry.Employee;
public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            // Create registry
            StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
                    .applySetting("hibernate.connection.driver_class", "com.mysql.cj.jdbc.Driver")
                    .applySetting("hibernate.connection.url", "jdbc:mysql://localhost:3306/wipro")
                    .applySetting("hibernate.connection.username", "root")
                    .applySetting("hibernate.connection.password", "#Mahadev7")
                    .applySetting("hibernate.dialect", "org.hibernate.dialect.MySQLDialect")
                    .applySetting("hibernate.hbm2ddl.auto", "update")
                    .applySetting("hibernate.show_sql", "true")
                    .applySetting("hibernate.format_sql", "true")
                    .build();

            // Create session factory
            return new MetadataSources(registry)
                    .addAnnotatedClass(Employee.class)
                    .buildMetadata()
                    .buildSessionFactory();
        } catch (Exception ex) {
            throw new ExceptionInInitializerError("SessionFactory creation failed: " + ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public static void shutdown() {
        if (sessionFactory != null) {
            sessionFactory.close();
        }
    }
}
```

Now i am going to create the test class named as App1.java

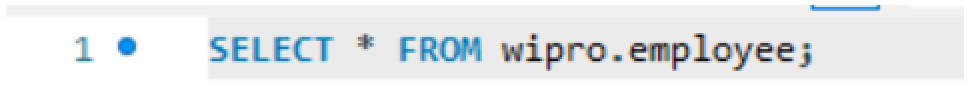


```java
package com.wipro;

import org.hibernate.Session;
import org.hibernate.Transaction;
import com.wipro.entiry.Employee;

public class App1 {
    public static void main(String[] args) {
        // Open a Hibernate session
        Session session = HibernateUtil.getSessionFactory().openSession();

        // Start a transaction
        Transaction transaction = null;

        try {
            transaction = session.beginTransaction();

            // Create a new Employee object
            Employee emp = new Employee();
            emp.setId(800);
            emp.setName("infosys");

            // Save the employee object
            session.save(emp);

            // Commit the transaction
            transaction.commit();

            System.out.println("Employee data inserted successfully!");
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        } finally {
            session.close(); // Close session
        }

        // Shutdown Hibernate
        HibernateUtil.shutdown();
    }
}
```

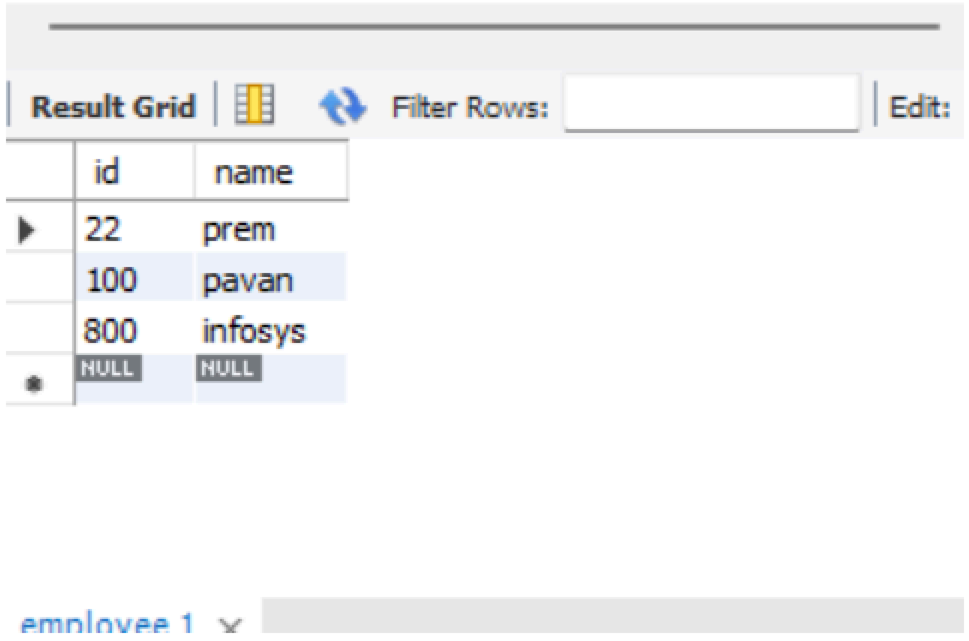
As soon as we run the **app1 java**



```
INFO: HHH10001501: Connection obtained from Jc
Hibernate:
    insert
    into
        employee
        (name,id)
    values
        (?,?)
Employee data inserted successfully!
Feb 24, 2025 2:00:06 PM org.hibernate.engine.j
```



```
SELECT * FROM wipro.employee;
```



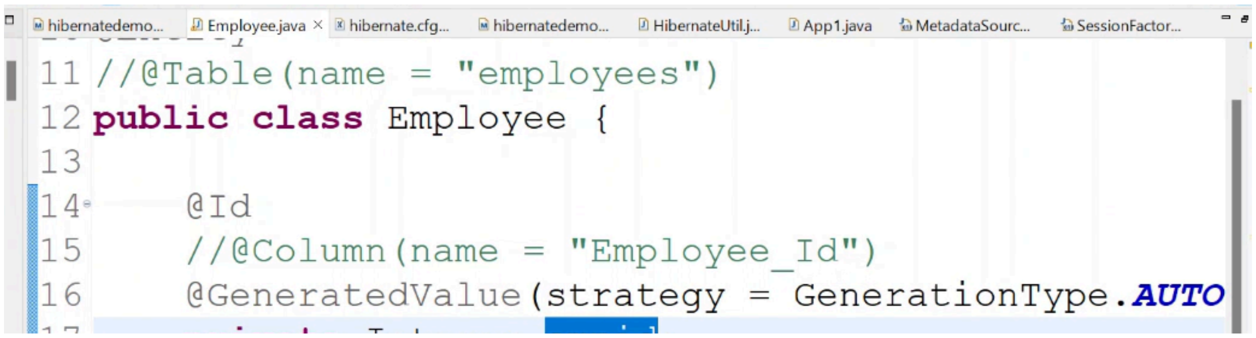
Result Grid | Filter Rows: | Edit:

id	name
22	prem
100	pavan
800	infosys
NULL	NULL

employee 1 ×

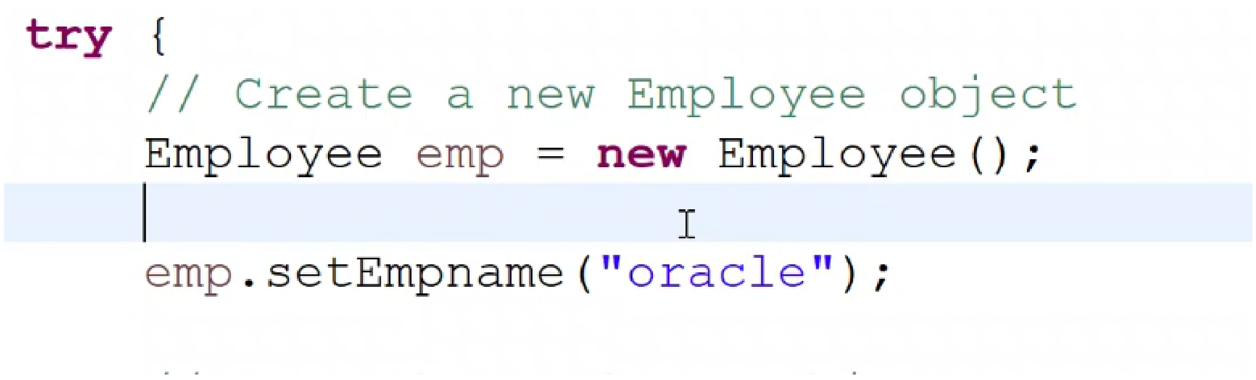
How to generate a value for the particular column ..Right now we r passing 800 ..Instead of manually passing this value..Is there any other method to autogenerate the value

GeneratedValue(strategy=GenerationType.Auto



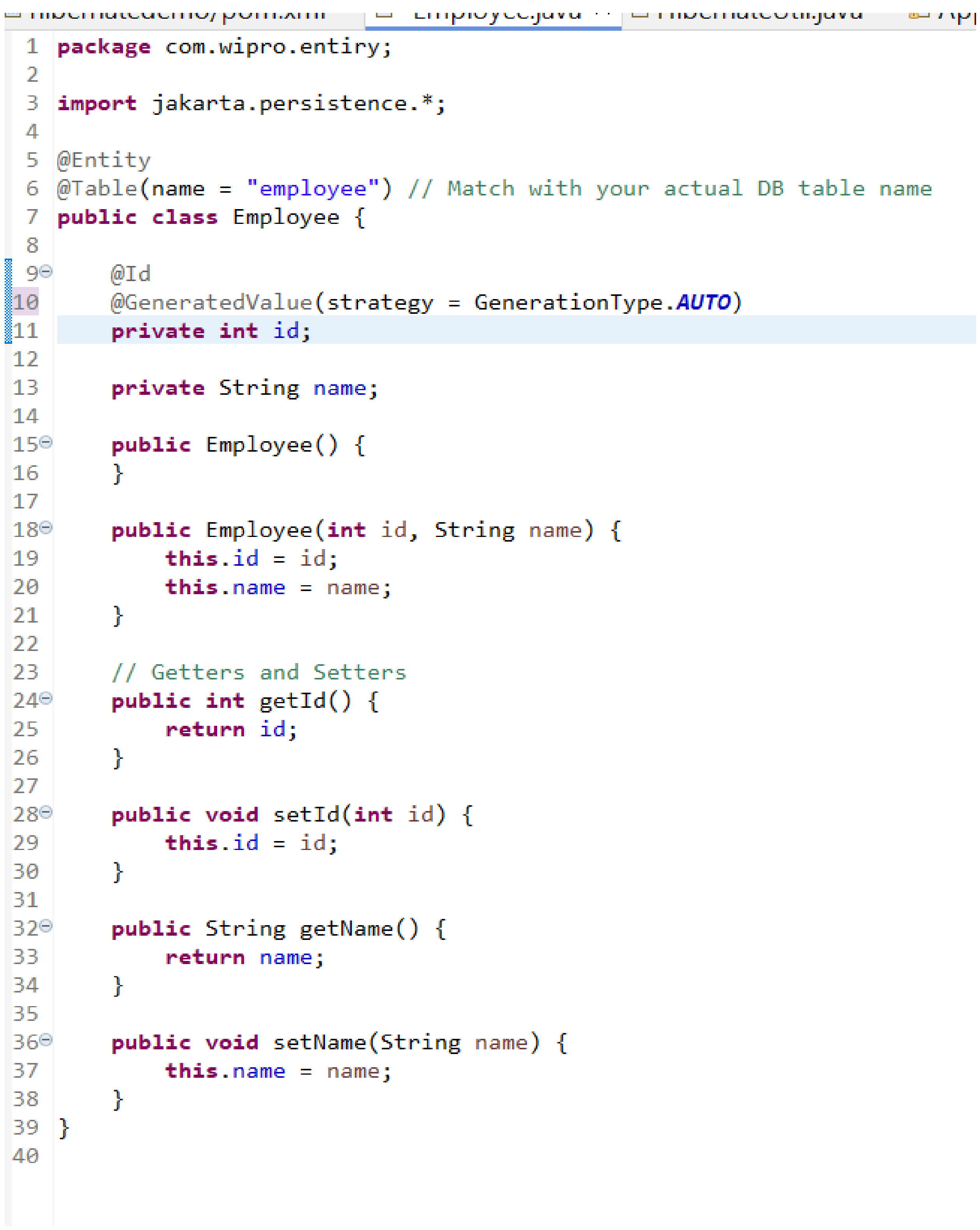
```
//@Table(name = "employees")
public class Employee {

    @Id
    //@Column(name = "Employee_Id")
    @GeneratedValue(strategy = GenerationType.AUTO
```

GeneratedValue(strategy=GenerationType.Auto…. It is used to autogenerate a value



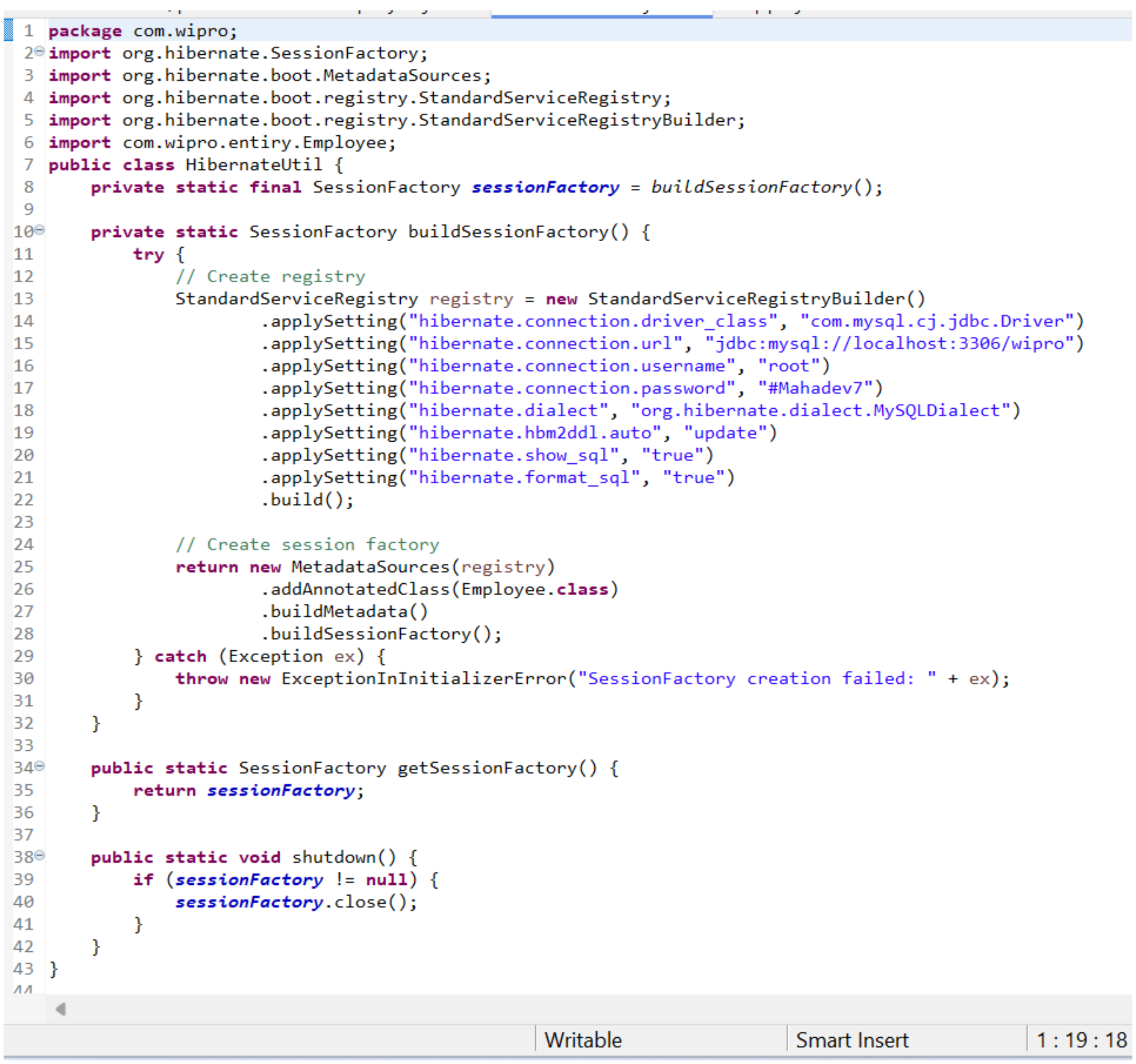
```
try {
    // Create a new Employee object
    Employee emp = new Employee();

    emp.setEmpname("oracle");
```

Now i removed the empid ..so that this generatedvale method will give u the autogenerated value



```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.AUTO)
    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;
    }
}

```

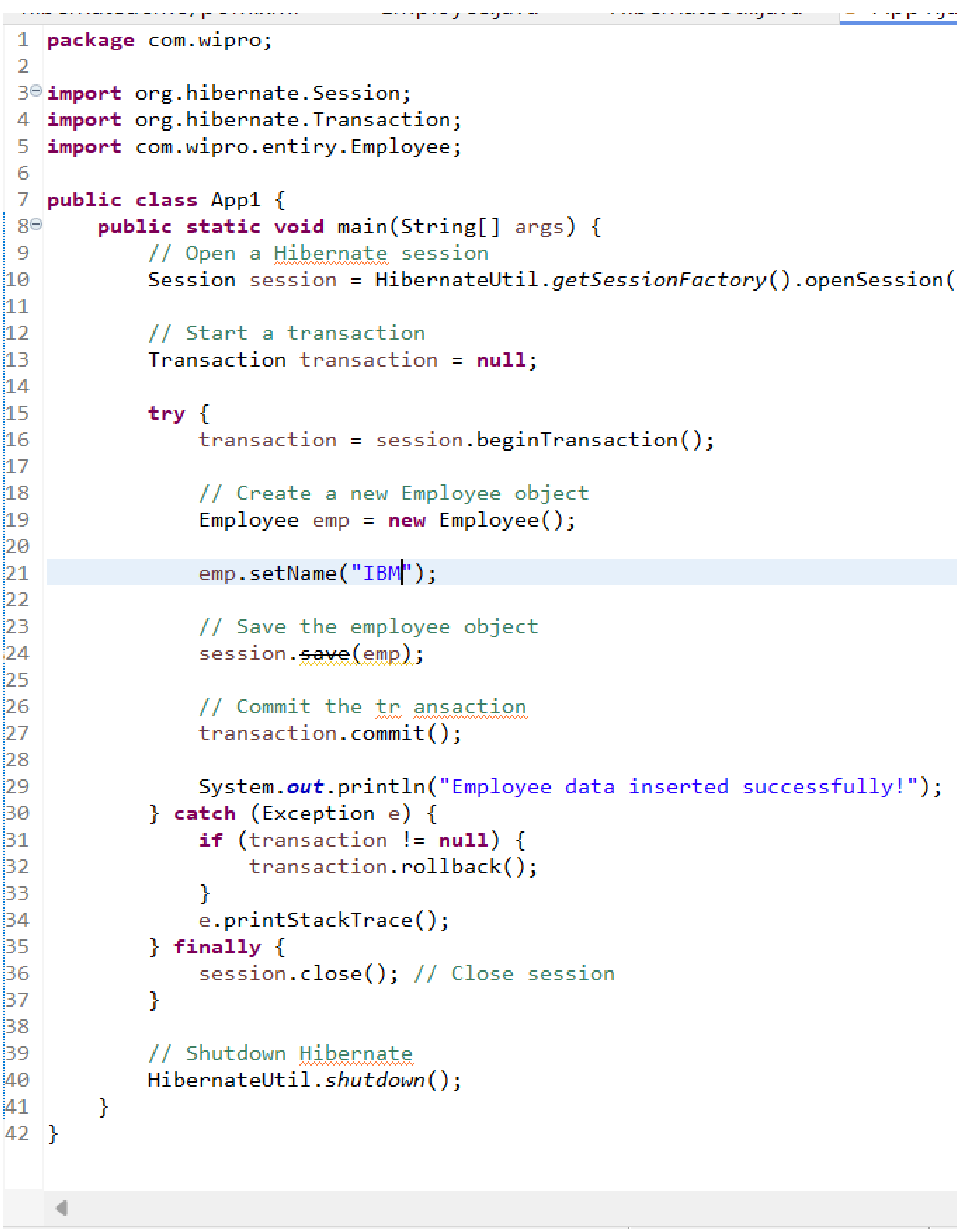
No changes in the HibernateUtil class



```
package com.wipro;
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import com.wipro.entiry.Employee;
public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            // Create registry
            StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
                    .applySetting("hibernate.connection.driver_class", "com.mysql.cj.jdbc.Driver")
                    .applySetting("hibernate.connection.url", "jdbc:mysql://localhost:3306/wipro")
                    .applySetting("hibernate.connection.username", "root")
                    .applySetting("hibernate.connection.password", "#Mahadev7")
                    .applySetting("hibernate.dialect", "org.hibernate.dialect.MySQLDialect")
                    .applySetting("hibernate.hbm2ddl.auto", "update")
                    .applySetting("hibernate.show_sql", "true")
                    .applySetting("hibernate.format_sql", "true")
                    .build();

            // Create session factory
            return new MetadataSources(registry)
                    .addAnnotatedClass(Employee.class)
                    .buildMetadata()
                    .buildSessionFactory();
        } catch (Exception ex) {
            throw new ExceptionInInitializerError("SessionFactory creation failed: " + ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public static void shutdown() {
        if (sessionFactory != null) {
            sessionFactory.close();
        }
    }
}
```

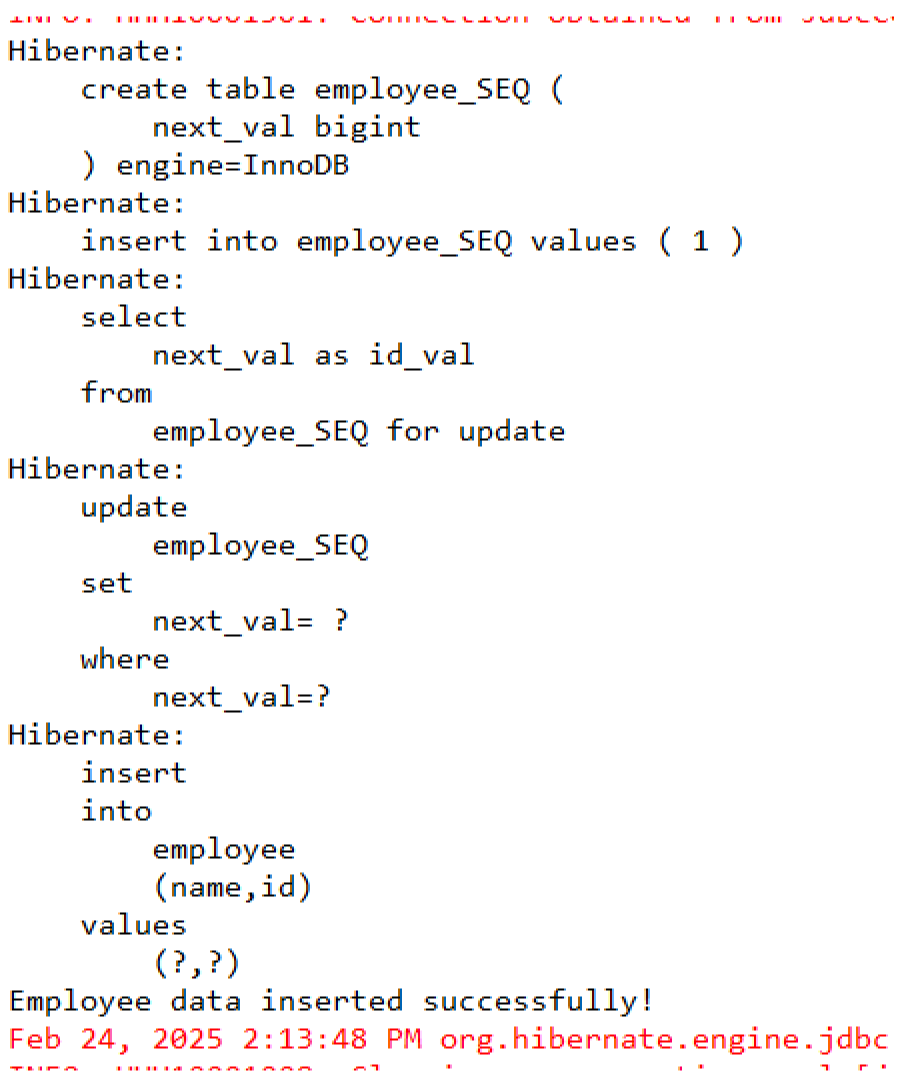
Writable | Smart Insert | 1 : 19 : 18

In the App1 java class
I removed the empid (which we r manually passing one)

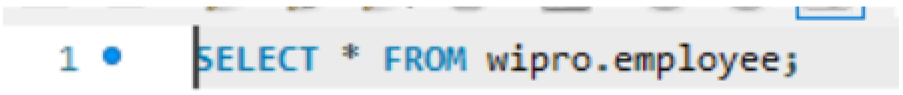


```java
package com.wipro;

import org.hibernate.Session;
import org.hibernate.Transaction;
import com.wipro.entiry.Employee;

public class App1 {
    public static void main(String[] args) {
        // Open a Hibernate session
        Session session = HibernateUtil.getSessionFactory().openSession(

        // Start a transaction
        Transaction transaction = null;

        try {
            transaction = session.beginTransaction();

            // Create a new Employee object
            Employee emp = new Employee();

            emp.setName("IBM");

            // Save the employee object
            session.save(emp);

            // Commit the tr ansaction
            transaction.commit();

            System.out.println("Employee data inserted successfully!");
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        } finally {
            session.close(); // Close session
        }

        // Shutdown Hibernate
        HibernateUtil.shutdown();
    }
}
```

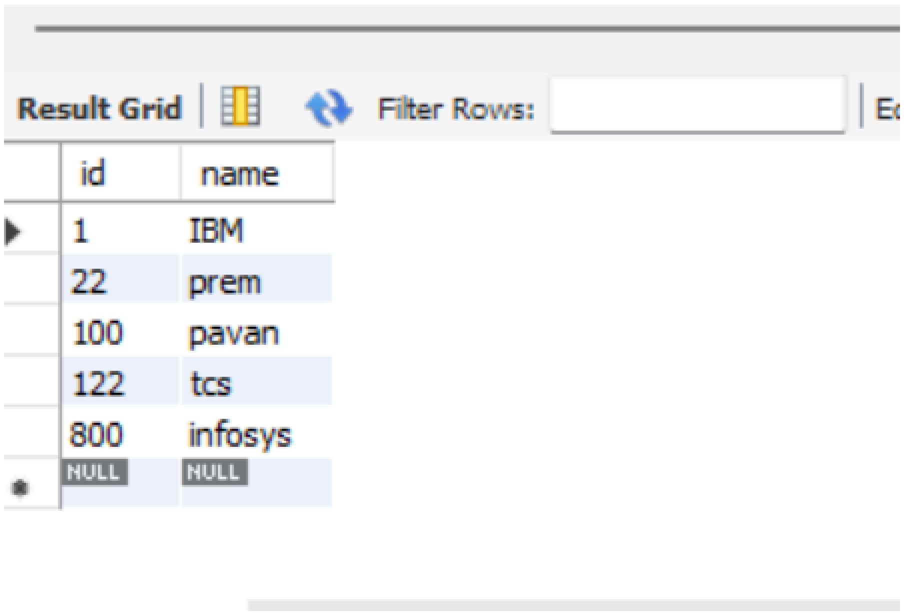
now run the app1 java code



```
INFO: HHH10001501: Connection obtained from JdbcC
Hibernate: 
    create table employee_SEQ (
        next_val bigint
    ) engine=InnoDB
Hibernate: 
    insert into employee_SEQ values ( 1 )
Hibernate: 
    select
        next_val as id_val 
    from
        employee_SEQ for update
Hibernate: 
    update
        employee_SEQ 
    set
        next_val= ? 
    where
        next_val=?
Hibernate: 
    insert 
    into
        employee
        (name,id) 
    values
        (?,?)
Employee data inserted successfully!
Feb 24, 2025 2:13:48 PM org.hibernate.engine.jdbc
INFO: HHH10001008: Cleaning up connection pool [j
```



```
SELECT * FROM wipro.employee;
```

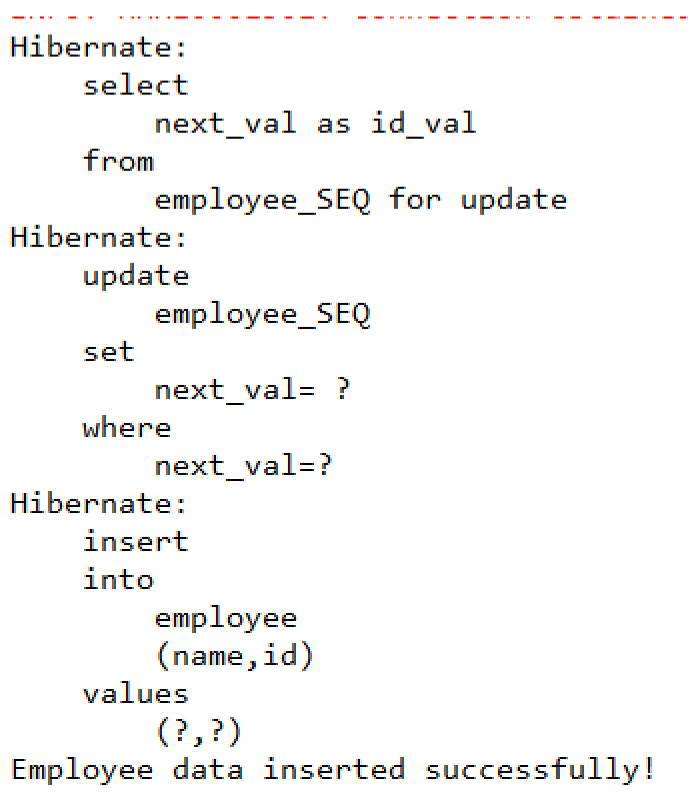


id	name
1	IBM
22	prem
100	pavan
122	tcs
800	infosys
NULL	NULL

Here now observe it…I din't mentioned any empid for the ibm…It is autogenerating like 1

If i mentioned the **company name as google and i din't mention the empid then we will see what will happen**

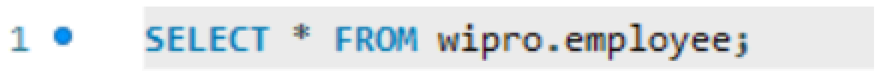
```
try {
    transaction = session.beginTransaction();

    // Create a new Employee object
    Employee emp = new Employee();

    emp.setName("Google");
```



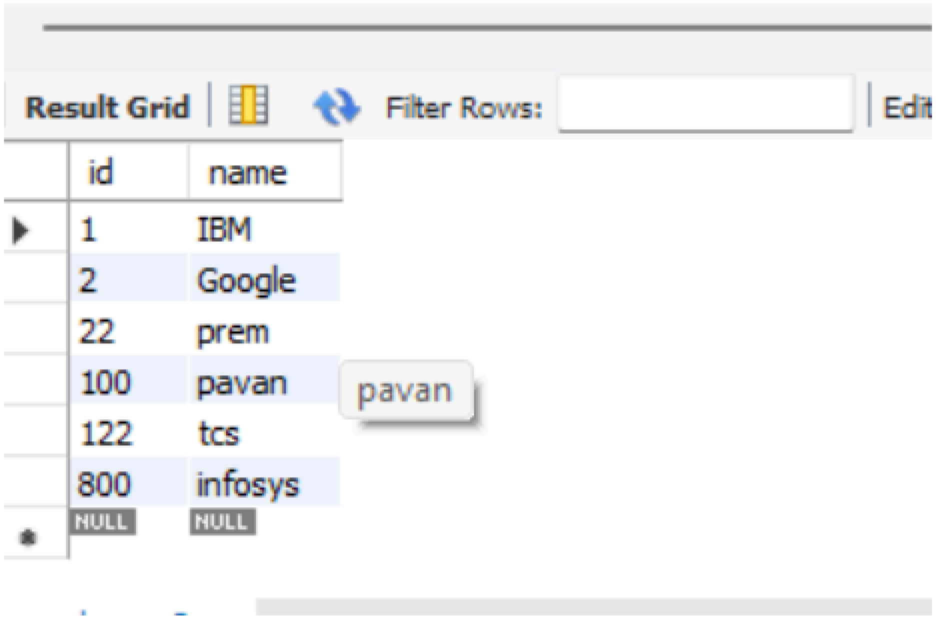
```
Hibernate:
    select
        next_val as id_val
    from
        employee_SEQ for update
Hibernate:
    update
        employee_SEQ
    set
        next_val= ?
    where
        next_val=?
Hibernate:
    insert
    into
        employee
        (name,id)
    values
        (?,?)
Employee data inserted successfully!
```

The data is added successfully

Now check in the sql table



```sql
SELECT * FROM wipro.employee;
```

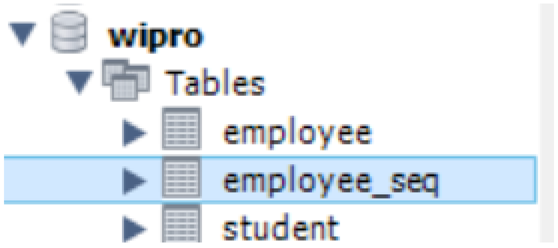


id	name
1	IBM
2	Google
22	prem
100	pavan
122	tcs
800	infosys
NULL	NULL

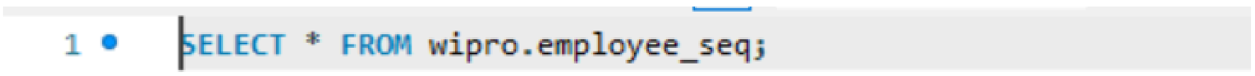
Now this line of code is responsible for the autoincrementing the value by 1 ,if we didn't mention the empid

@GeneratedValue(**strategy = GenerationType.*AUTO***)

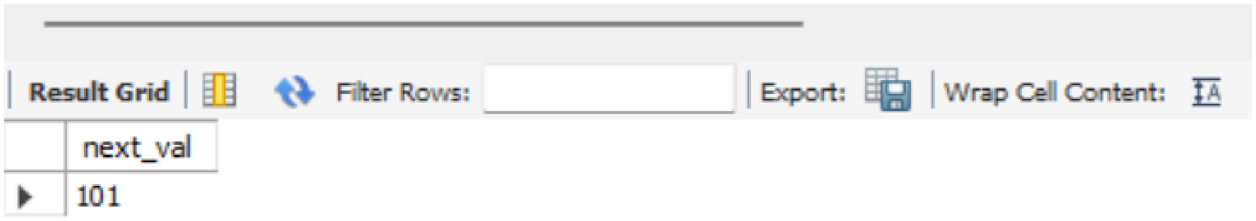
In addition to that if we observe the database by refreshing there is one **sequence object is created There we can see the next value**



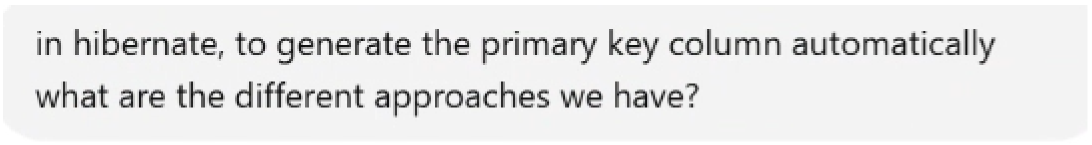
If we check the rows in the seq table then we can find th next value



```
SELECT * FROM wipro.employee_seq;
```

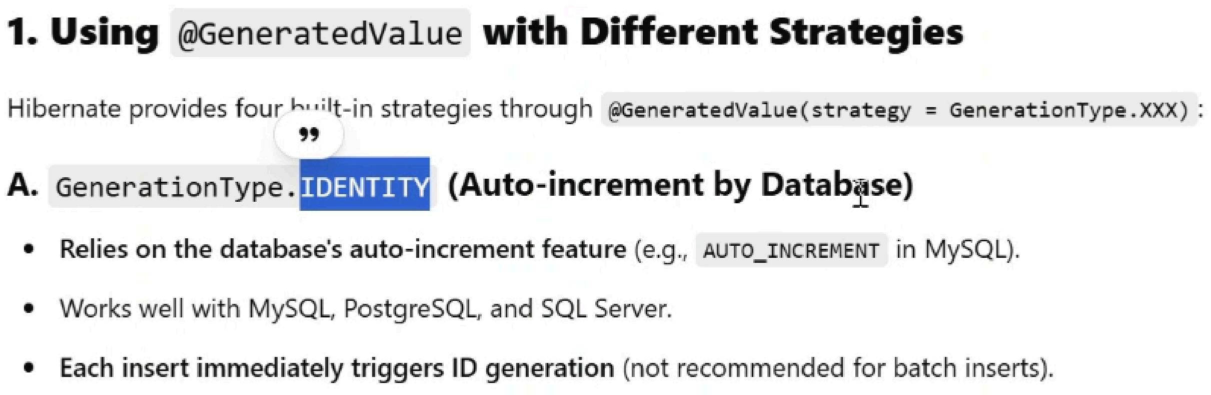


```
next_val
101
```



in hibernate, to generate the primary key column automatically what are the different approaches we have?

We r having the 4 approaches to autoincrement the primary key in the hibernate



1. Using `@GeneratedValue` with Different Strategies

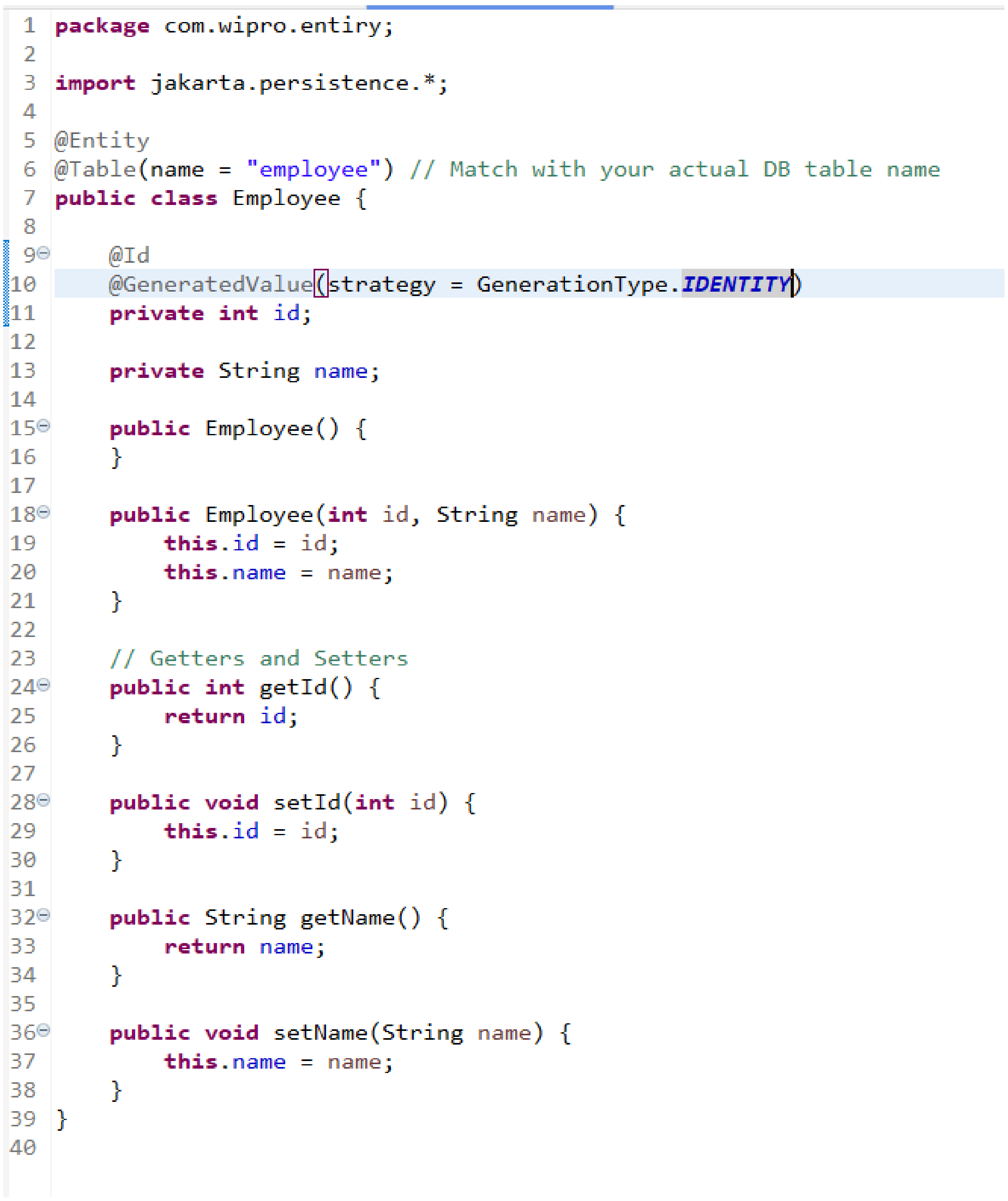
Hibernate provides four built-in strategies through `@GeneratedValue(strategy = GenerationType.XXX)`:

A. `GenerationType.IDENTITY` (Auto-increment by Database)

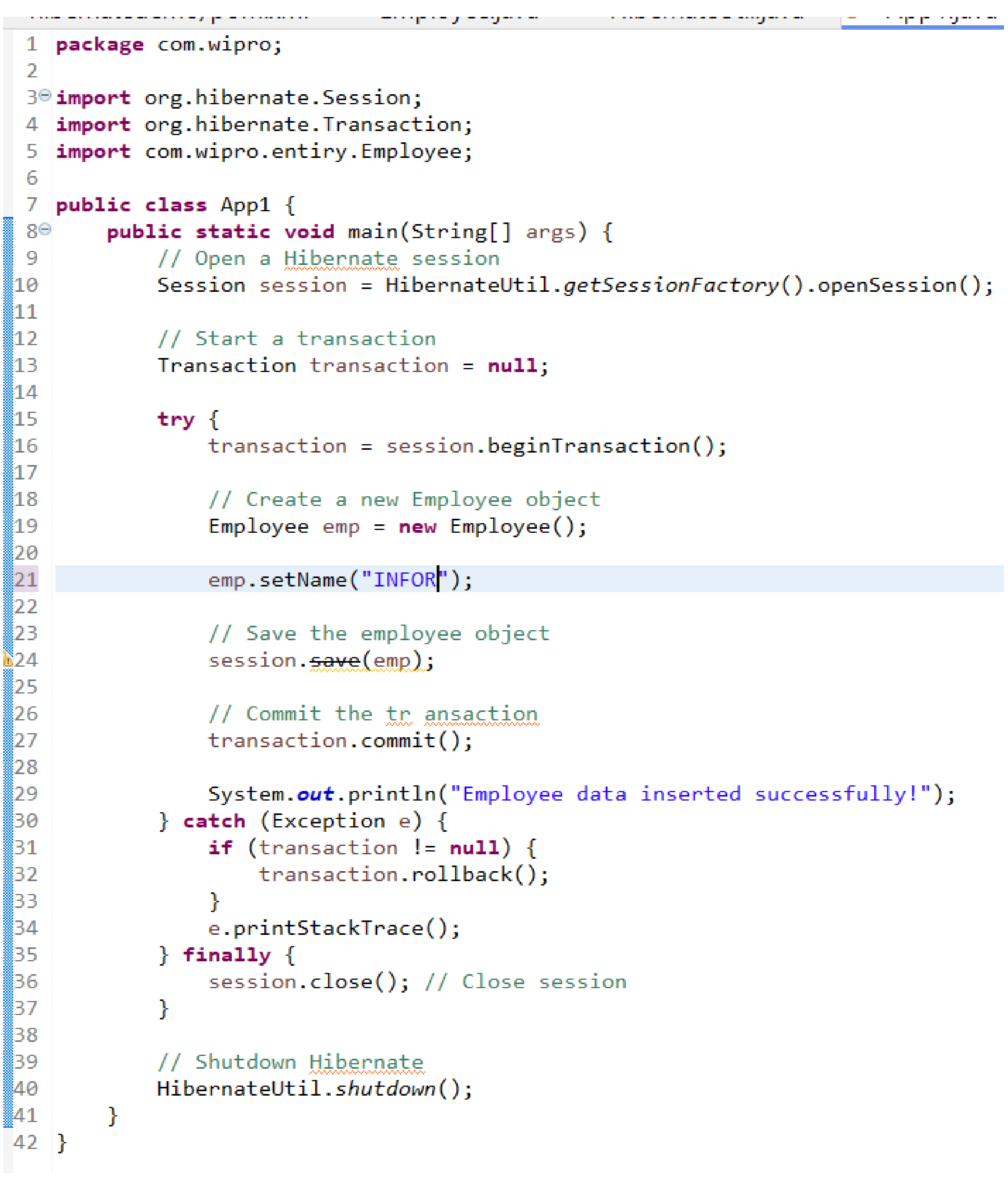
- **Relies on the database's auto-increment feature** (e.g., `AUTO_INCREMENT` in MySQL).
- Works well with MySQL, PostgreSQL, and SQL Server.
- **Each insert immediately triggers ID generation** (not recommended for batch inserts).

For suppose if i use the Oracle database there we cannot use this one **bcoz it cannot support to the oracle**

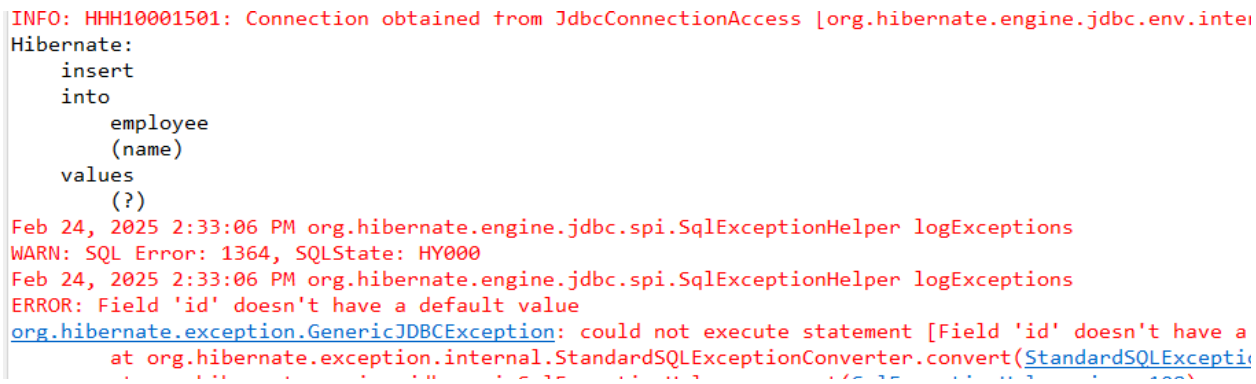
This only works for the Mysql,PostgreSQL and SQL SERVER



```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)
    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;
    }
}
```

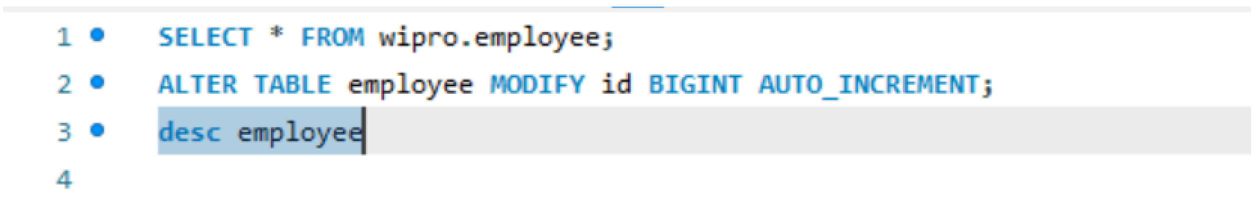


```java
package com.wipro;

import org.hibernate.Session;
import org.hibernate.Transaction;
import com.wipro.entiry.Employee;

public class App1 {
    public static void main(String[] args) {
        // Open a Hibernate session
        Session session = HibernateUtil.getSessionFactory().openSession();

        // Start a transaction
        Transaction transaction = null;

        try {
            transaction = session.beginTransaction();

            // Create a new Employee object
            Employee emp = new Employee();

            emp.setName("INFOR");

            // Save the employee object
            session.~~save~~(emp);

            // Commit the tr ansaction
            transaction.commit();

            System.out.println("Employee data inserted successfully!");
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        } finally {
            session.close(); // Close session
        }

        // Shutdown Hibernate
        HibernateUtil.shutdown();
    }
}
```

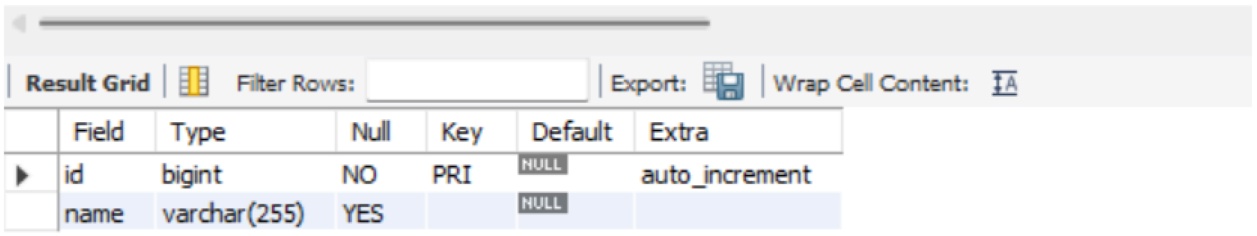


```
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.inte
Hibernate: 
    insert 
    into
        employee
        (name) 
    values
        (?)
Feb 24, 2025 2:33:06 PM org.hibernate.engine.jdbc.spi.SqlExceptionHelper logExceptions
WARN: SQL Error: 1364, SQLState: HY000
Feb 24, 2025 2:33:06 PM org.hibernate.engine.jdbc.spi.SqlExceptionHelper logExceptions
ERROR: Field 'id' doesn't have a default value
org.hibernate.exception.GenericJDBCException: could not execute statement [Field 'id' doesn't have a
        at org.hibernate.exception.internal.StandardSQLExceptionConverter.convert(StandardSQLExcepti
```

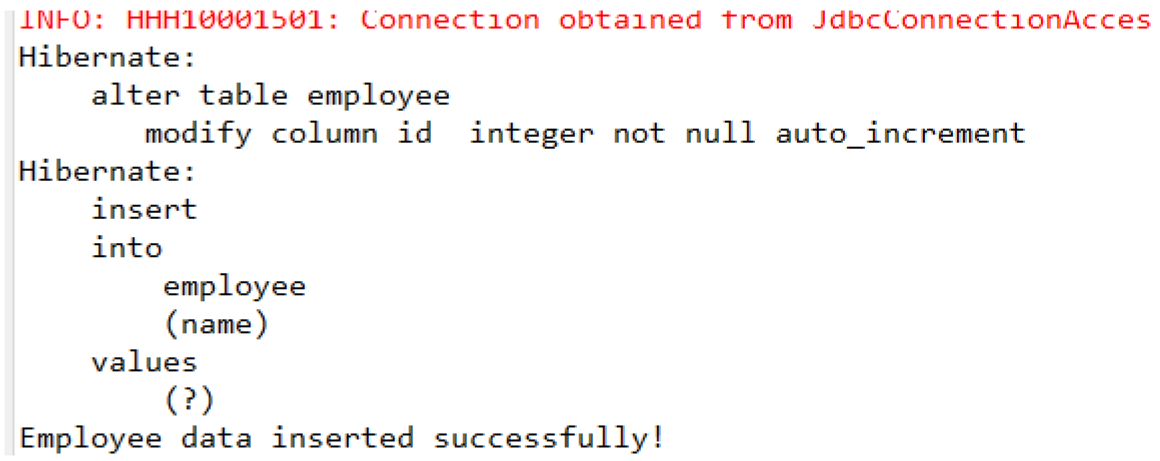
WE R GETTIGN THE ERROR Y BCOZ HERE WE NEED TO MENTION THE **autoincrement in mysql for the primary key**



```sql
SELECT * FROM wipro.employee;
ALTER TABLE employee MODIFY id BIGINT AUTO_INCREMENT;
desc employee
```

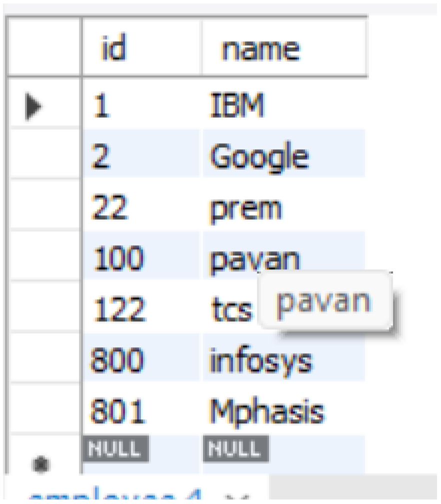


Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	auto_increment
name	varchar(255)	YES		NULL	



```
INFO: HHH10001501: Connection obtained from JdbcConnectionAcces
Hibernate: 
    alter table employee 
       modify column id  integer not null auto_increment
Hibernate: 
    insert 
    into
        employee
        (name) 
    values
        (?)
Employee data inserted successfully!
```

Now it is working fine without the exceptions



	id	name
▶	1	IBM
	2	Google
	22	prem
	100	pavan
	122	tcs
	800	infosys
	801	Mphasis
	NULL	NULL

See here mphasis company name is added

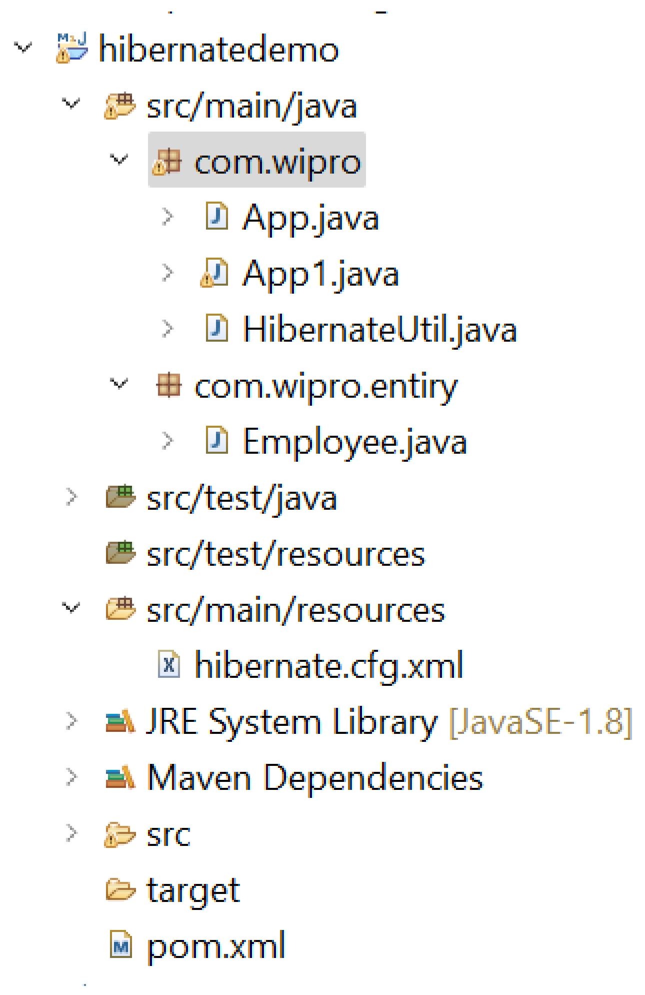
B. `GenerationType.SEQUENCE` (Database Sequence)

- Uses a **database sequence object** to generate primary keys.
- Works well with **Oracle, PostgreSQL, and H2** (Databases that support sequences).
- **Efficient for batch inserts** since it does not require immediate insert execution.

As of Now i am using the MYSQL database so i am not using the Sequence it is only used for the oracle database

C. `GenerationType.TABLE` (Table-based Strategy)

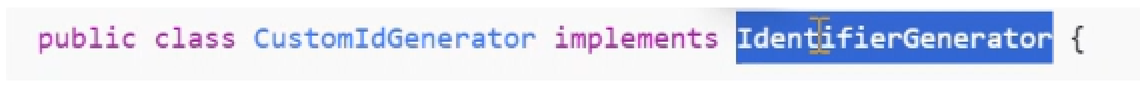
- Uses a separate table to **maintain primary key values.**
- This is **not commonly used** as it is **less efficient** than sequences or identity.



```
hibernatedemo
    src/main/java
        com.wipro
            App.java
            App1.java
            HibernateUtil.java
        com.wipro.entiry
            Employee.java
    src/test/java
    src/test/resources
    src/main/resources
        hibernate.cfg.xml
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    pom.xml
```

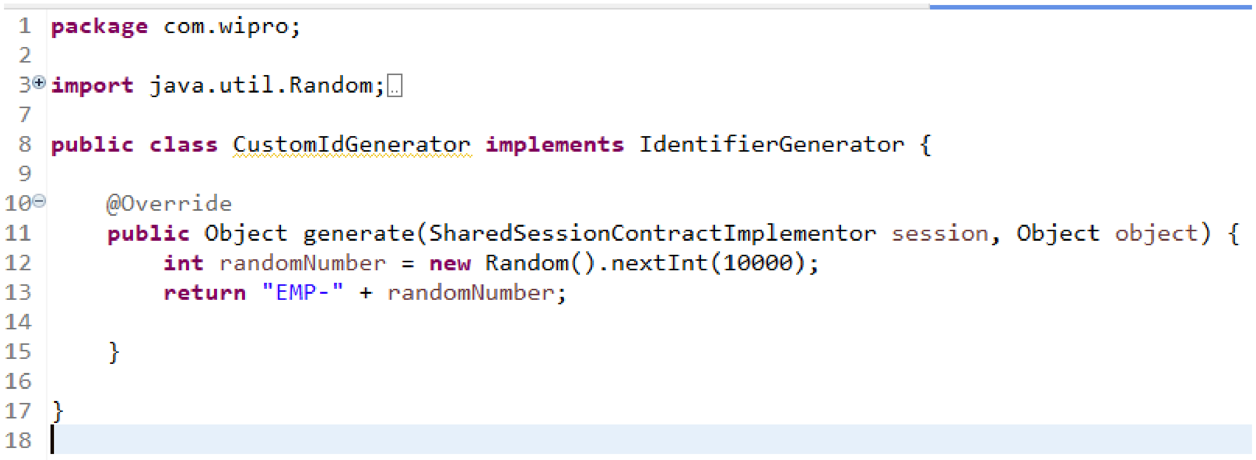
Instead of depending on the pre defined generators…..i need to customize this as i want …then i need to create the Customgenetator which is implementing from the IdentifierGenerator

Now we r going to generate the Customized primary key like abc 1,abc, 2,abc 3….



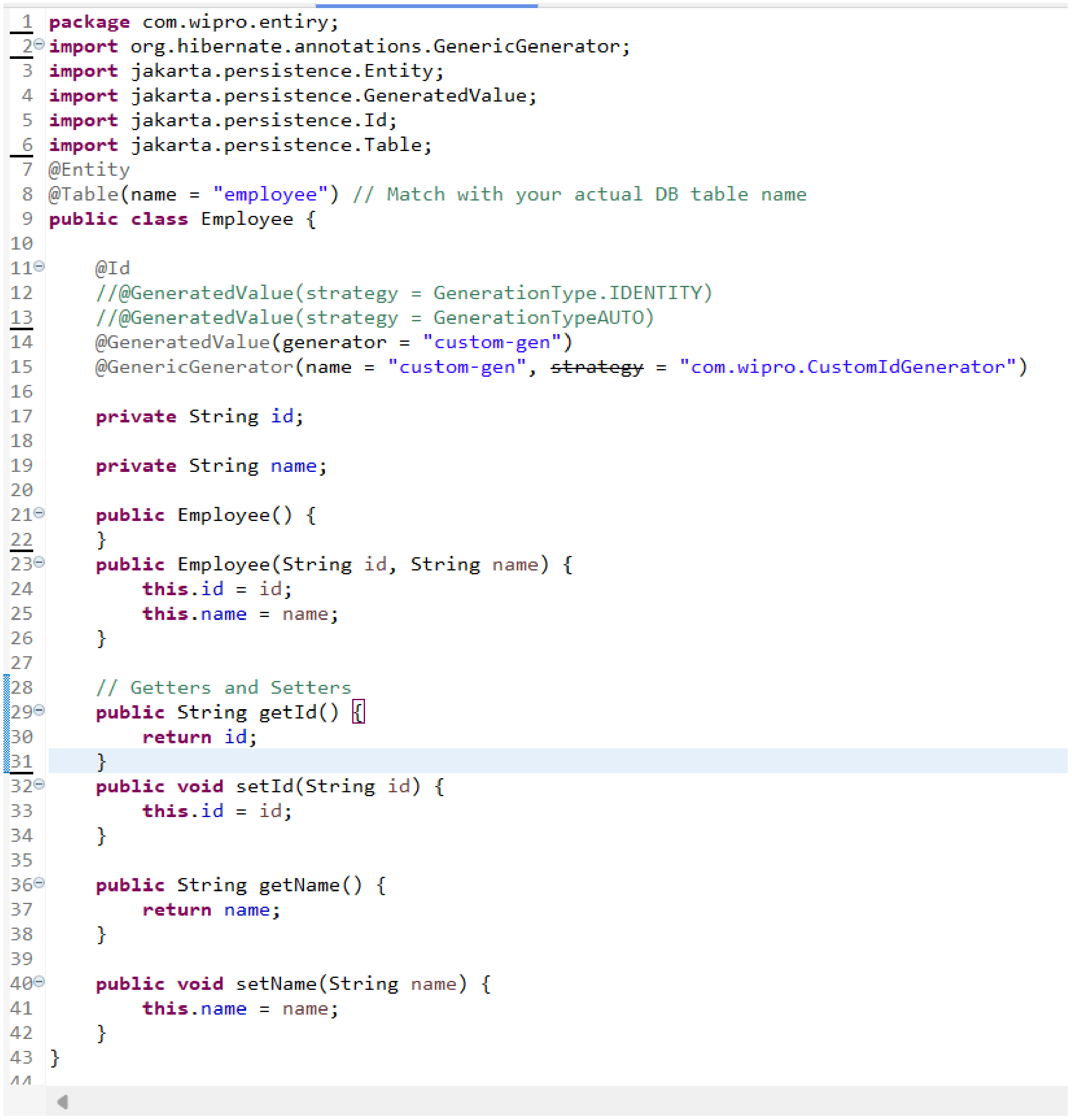
```java
public class CustomIdGenerator implements IdentifierGenerator {
```

Now i am going to create one more class in the same com.wipro package like CustomIdGenerator which must be **implemented** from **IdentifierGenerator**

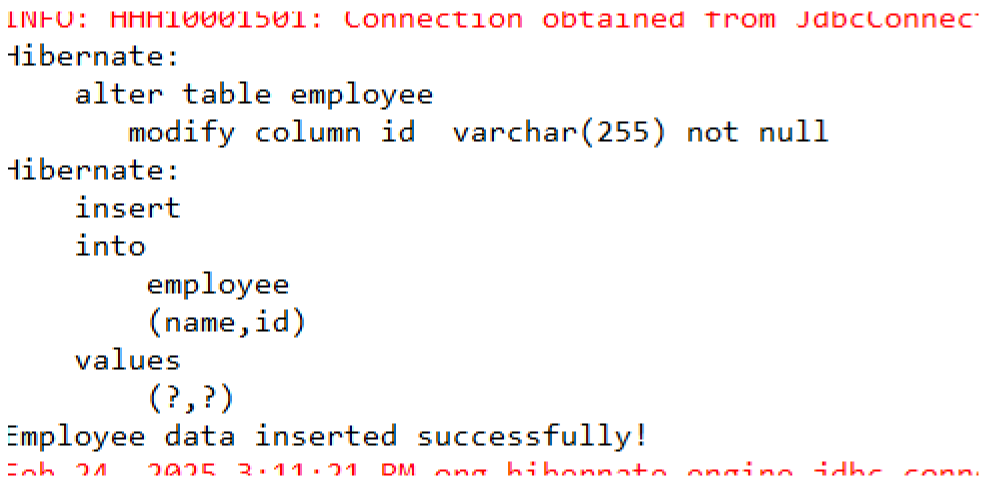


```java
package com.wipro;

import java.util.Random;

public class CustomIdGenerator implements IdentifierGenerator {

    @Override
    public Object generate(SharedSessionContractImplementor session, Object object) {
        int randomNumber = new Random().nextInt(10000);
        return "EMP-" + randomNumber;

    }

}

```

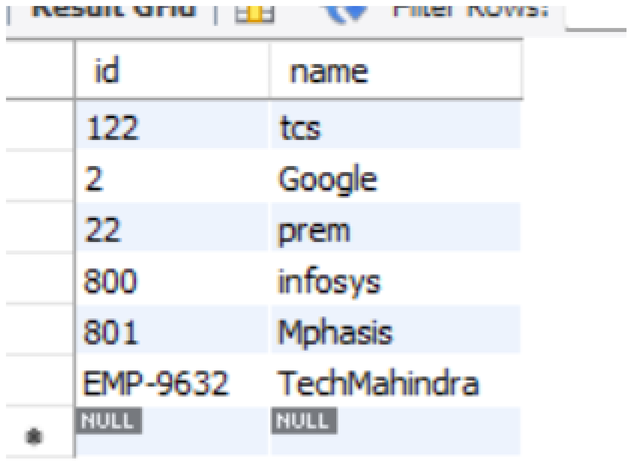
here emp is the customized one,,we can specify anything here



```java
package com.wipro.entiry;
import org.hibernate.annotations.GenericGenerator;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
@Entity
@Table(name = "employee") // Match with your actual DB table name
public class Employee {

    @Id
    //@GeneratedValue(strategy = GenerationType.IDENTITY)
    //@GeneratedValue(strategy = GenerationTypeAUTO)
    @GeneratedValue(generator = "custom-gen")
    @GenericGenerator(name = "custom-gen", ~~strategy~~ = "com.wipro.CustomIdGenerator")

    private String id;

    private String name;

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

    // Getters and Setters
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```



```
INFO: HHH10001501: Connection obtained from JdbcConnec
Hibernate:
    alter table employee
       modify column id  varchar(255) not null
Hibernate:
    insert
    into
        employee
        (name,id)
    values
        (?,?)
Employee data inserted successfully!
Feb 24, 2025 3:11:21 PM org.hibernate.engine.jdbc.conn
```



id	name
122	tcs
2	Google
22	prem
800	infosys
801	Mphasis
EMP-9632	TechMahindra
NULL	NULL

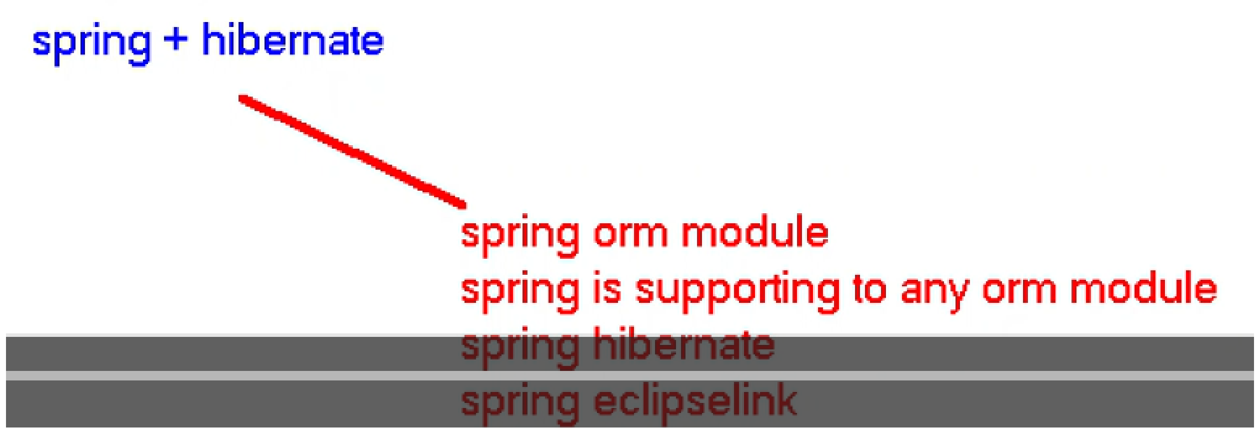
here we got 9632 y bcoz it is a random number between 1 to 10k…10 k we already mentioned in the code itself

```
package com.wipro;

import java.io.Serializable;
import java.util.Random;

import org.hibernate.engine.spi.SharedSessionContractImplementor;
import org.hibernate.generator.EventType;
import org.hibernate.id.IdentifierGenerator;
import org.hibernate.generator.BeforeExecutionGenerator;
```

```
public class CustomIdGenerator implements IdentifierGenerator, BeforeExecutionGenerator {


        @Override

        public Serializable generate(SharedSessionContractImplementor session, Object object) {

        // Generating a custom ID with "EMP-" prefix and a random number

        int randomNumber = new Random().nextInt(10000);

        return "EMP-" + randomNumber;

        }


        public EventType[] getGeneratedOn() {

        return new EventType[]{EventType.INSERT}; // Ensures ID is generated on INSERT

        }

}
```

alternate code for new version

Spring+Hibernate



spring + hibernate

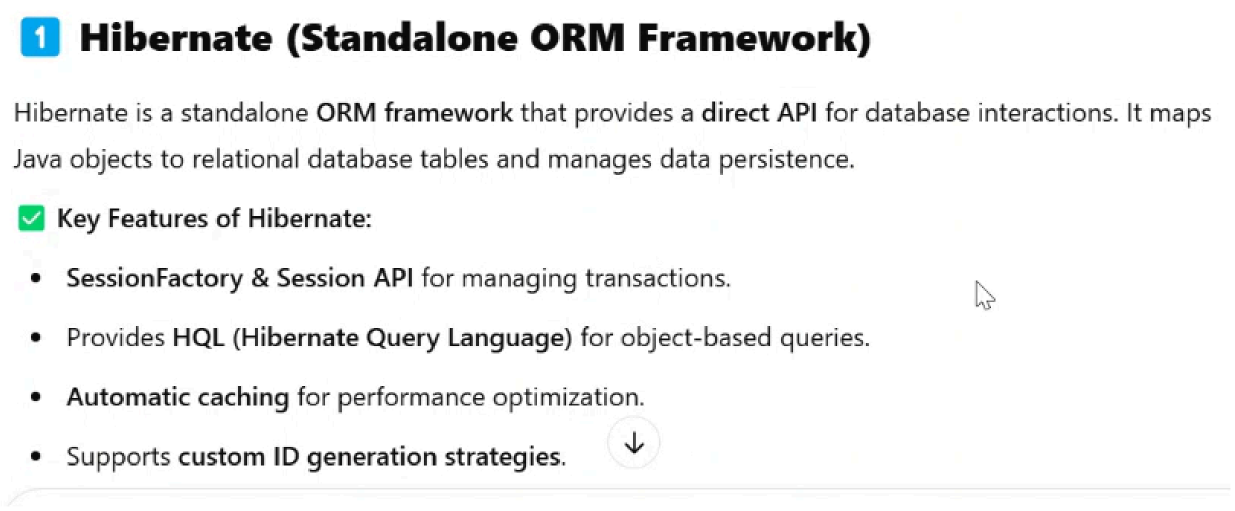
spring orm module

spring is supporting to any orm module

spring hibernate

spring eclipselink

what are the difference between hibernate and spring hibernate integration or orm vs spring orm module



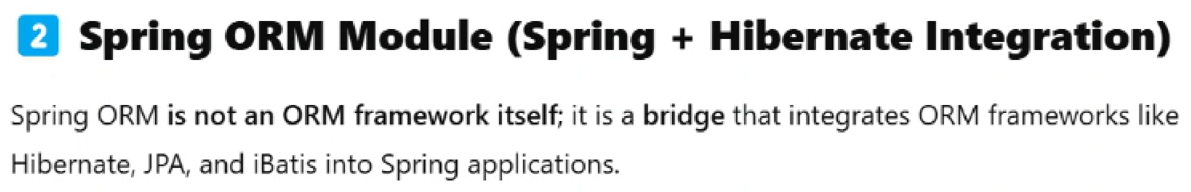
1 Hibernate (Standalone ORM Framework)

Hibernate is a standalone **ORM framework** that provides a **direct API** for database interactions. It maps Java objects to relational database tables and manages data persistence.

✅ Key Features of Hibernate:

- **SessionFactory & Session API** for managing transactions.
- Provides **HQL (Hibernate Query Language)** for object-based queries.
- **Automatic caching** for performance optimization.
- Supports **custom ID generation strategies**.

Hibernate **manages the transactions directly using the sessionFactory and session**



2 Spring ORM Module (Spring + Hibernate Integration)

Spring ORM **is not an ORM framework itself**; it is a **bridge** that integrates ORM frameworks like Hibernate, JPA, and iBatis into Spring applications.

✅ Why Use Spring ORM Module with Hibernate?

✔ **Eliminates Boilerplate Code:** No need to manually handle `SessionFactory`, transactions, or exception handling.
✔ **Declarative Transaction Management:** Uses `@Transactional` instead of manual `beginTransaction()` & `commit()`.
✔ **Spring Data JPA Integration:** Can work with repositories (`CrudRepository`, `JpaRepository`).
✔ **Dependency Injection Support:** Hibernate's `SessionFactory` is managed by Spring.
✔ **Exception Translation:** Converts Hibernate exceptions into Spring's `DataAccessException`.

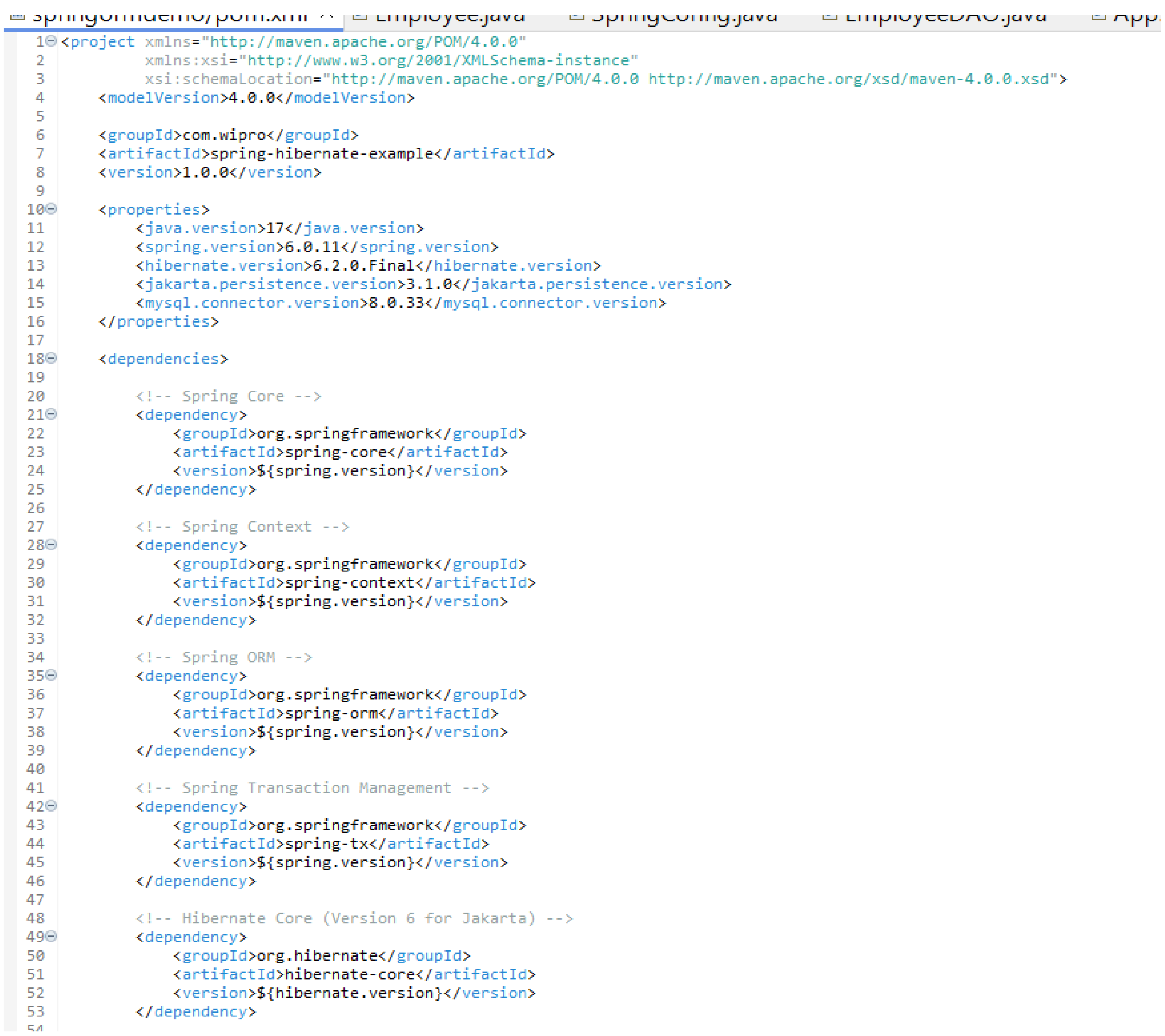
Feature	Hibernate (Standalone)	Spring ORM (Spring + Hibernate)
Type	Standalone ORM framework	Integration layer for Hibernate
Configuration	Requires `hibernate.cfg.xml`	Uses `application.properties`
Transaction Management	Manual (`beginTransaction()`)	Declarative (`@Transactional`)
Session Handling	Explicit `Session` handling	Managed via `EntityManager`
Exception Handling	Throws `HibernateException`	Converts to `DataAccessException`
Spring Boot Compatibility	Requires explicit setup	Fully supported & auto-configured
Boilerplate Code	More (`SessionFactory`, `Session`, etc.)	Less (Spring manages it)

4️⃣ When to Use Which?

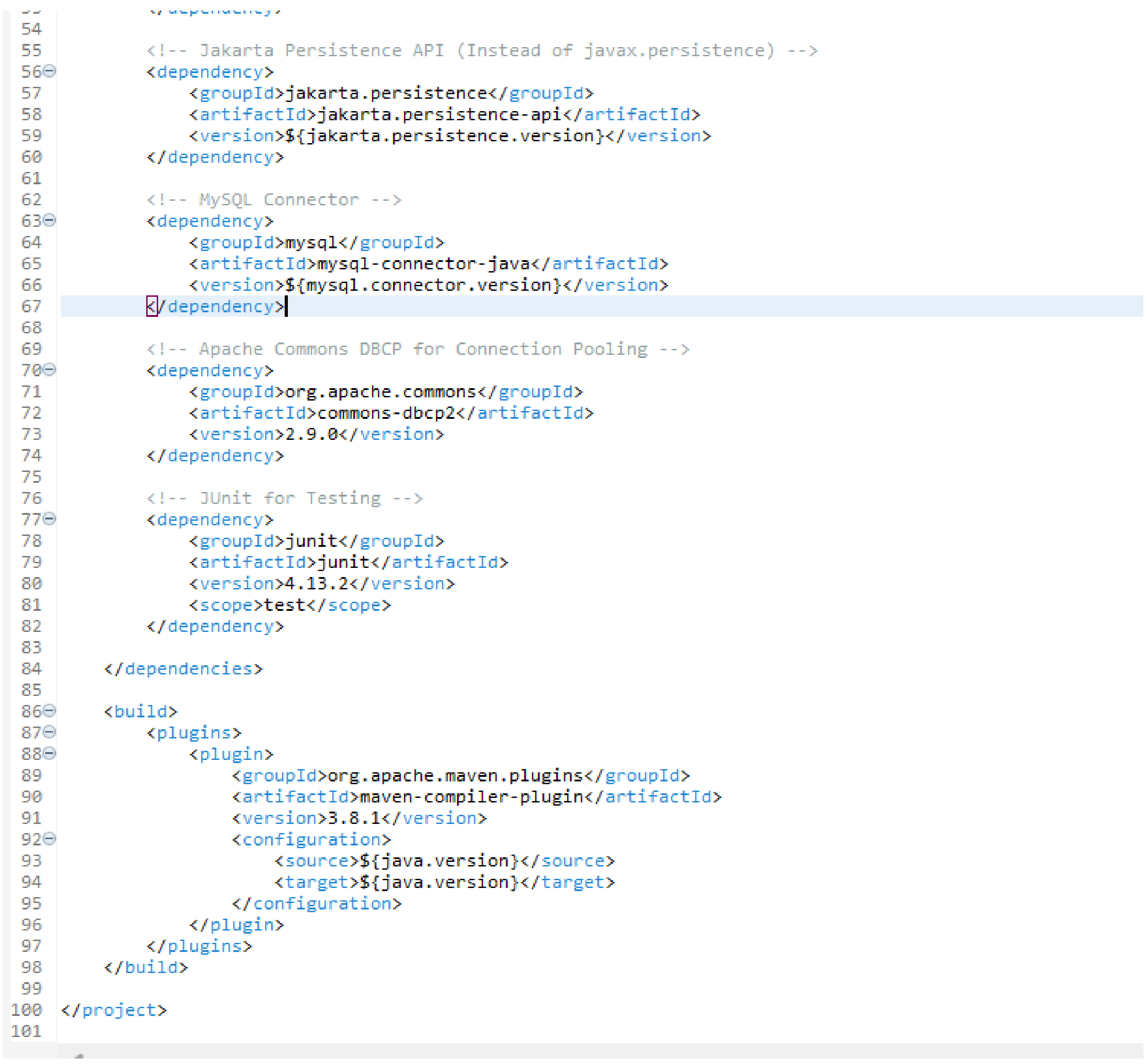
Scenario	Use Hibernate	Use Spring ORM (Spring + Hibernate)
Standalone Java App	✅ Best choice	❌ Not needed
Spring Boot App	❌ Avoid	✅ Best choice
Manual Control over Transactions	✅ Yes	❌ Spring manages it
Using Spring Data JPA	❌ No	✅ Recommended

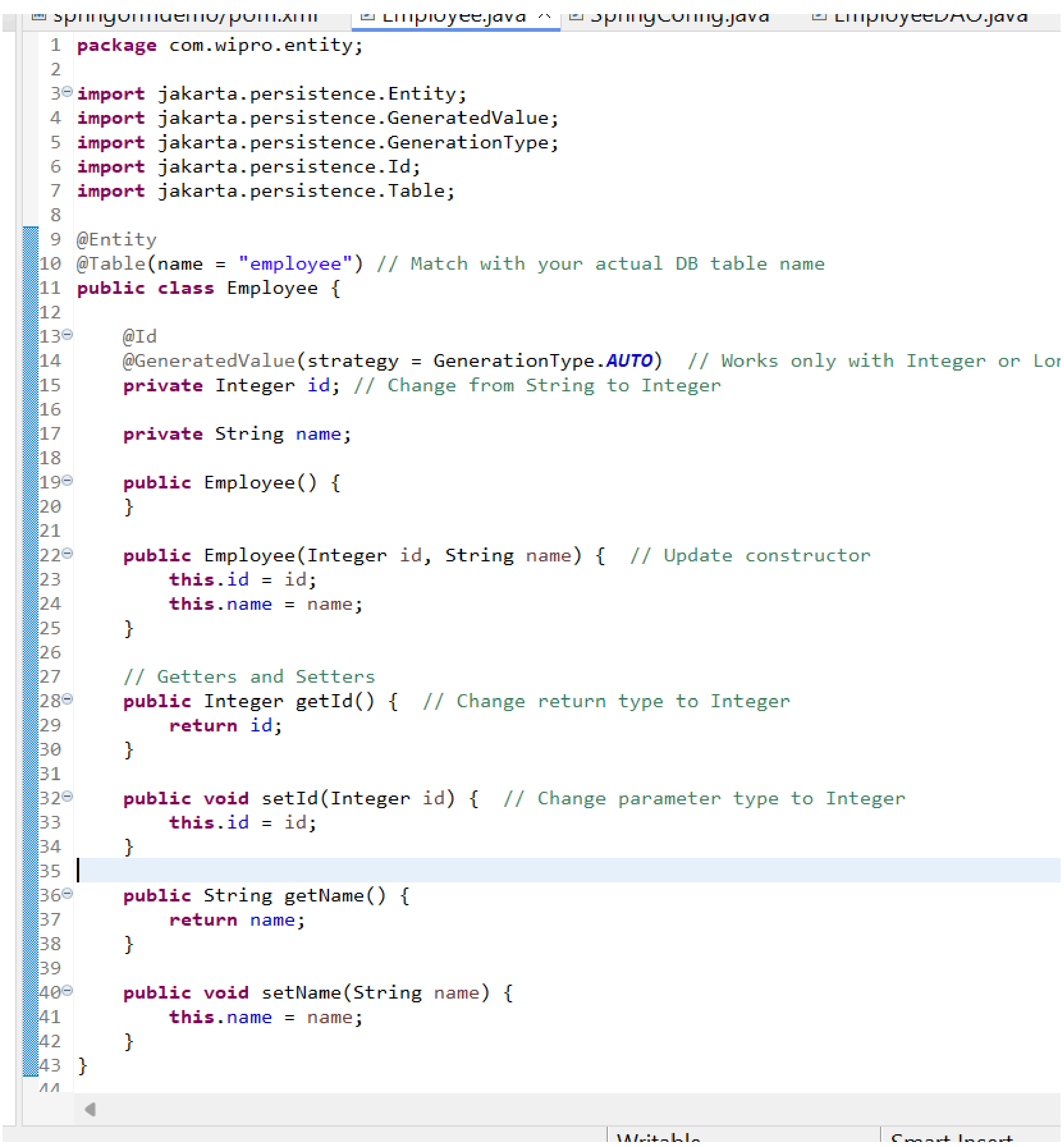
🔹 Conclusion

- If **not using Spring**, go with **pure Hibernate**.
- If using **Spring or Spring Boot**, prefer **Spring ORM (Spring + Hibernate integration)** for simpler **configuration, transaction management, and exception handling**.



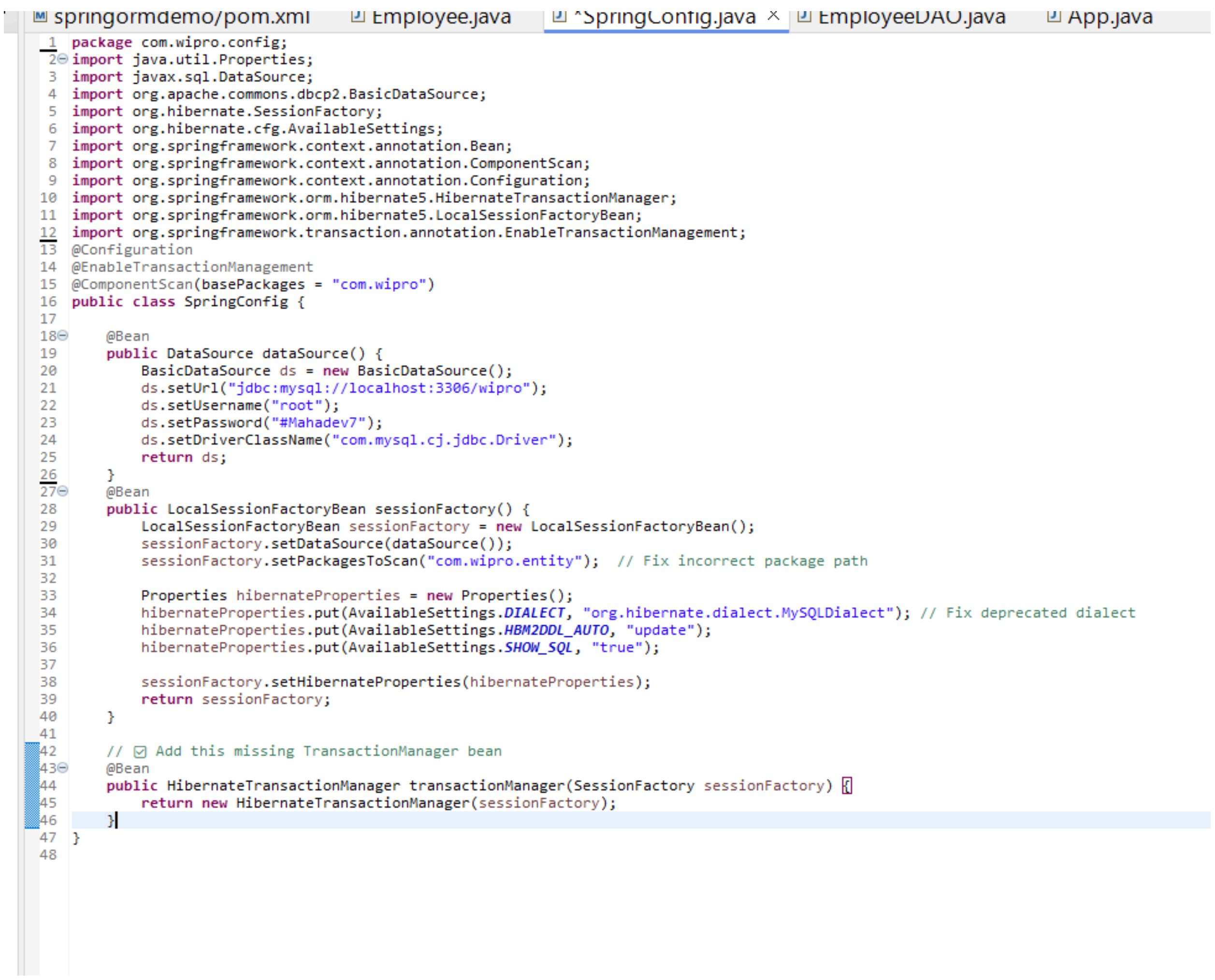
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.wipro</groupId>
    <artifactId>spring-hibernate-example</artifactId>
    <version>1.0.0</version>

    <properties>
        <java.version>17</java.version>
        <spring.version>6.0.11</spring.version>
        <hibernate.version>6.2.0.Final</hibernate.version>
        <jakarta.persistence.version>3.1.0</jakarta.persistence.version>
        <mysql.connector.version>8.0.33</mysql.connector.version>
    </properties>

    <dependencies>

        <!-- Spring Core -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- Spring Context -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- Spring ORM -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- Spring Transaction Management -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- Hibernate Core (Version 6 for Jakarta) -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>${hibernate.version}</version>
        </dependency>
```


```
        </dependency>

        <!-- Jakarta Persistence API (Instead of javax.persistence) -->
        <dependency>
            <groupId>jakarta.persistence</groupId>
            <artifactId>jakarta.persistence-api</artifactId>
            <version>${jakarta.persistence.version}</version>
        </dependency>

        <!-- MySQL Connector -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${mysql.connector.version}</version>
        </dependency>

        <!-- Apache Commons DBCP for Connection Pooling -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-dbcp2</artifactId>
            <version>2.9.0</version>
        </dependency>

        <!-- JUnit for Testing -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>${java.version}</source>
                    <target>${java.version}</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

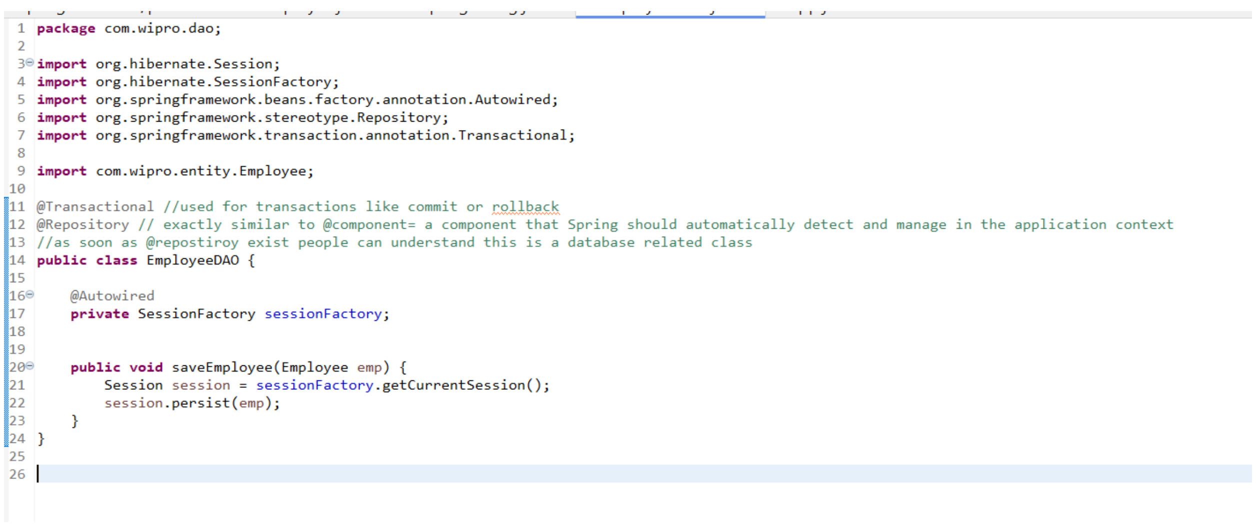


```java
package com.wipro.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

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

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)  // Works only with Integer or Lo
    private Integer id; // Change from String to Integer

    private String name;

    public Employee() {
    }

    public Employee(Integer id, String name) {  // Update constructor
        this.id = id;
        this.name = name;
    }

    // Getters and Setters
    public Integer getId() {  // Change return type to Integer
        return id;
    }

    public void setId(Integer id) {  // Change parameter type to Integer
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

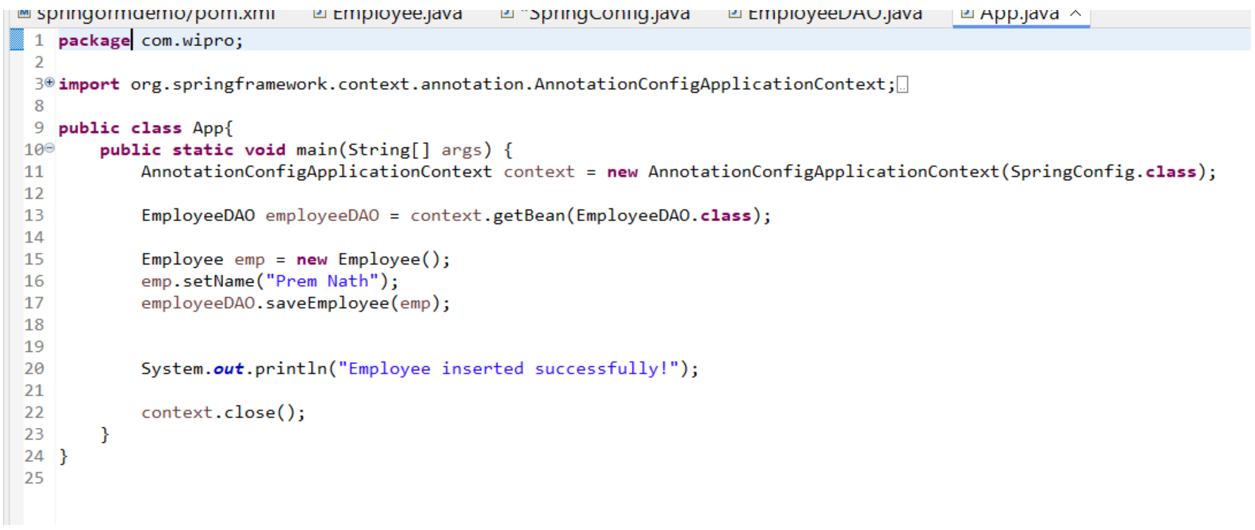
- `LocalSessionFactoryBean` auto-manages `SessionFactory`.
- `dataSource()` sets up MySQL connectivity.
- `sessionFactory()` registers Hibernate and scans for entities.



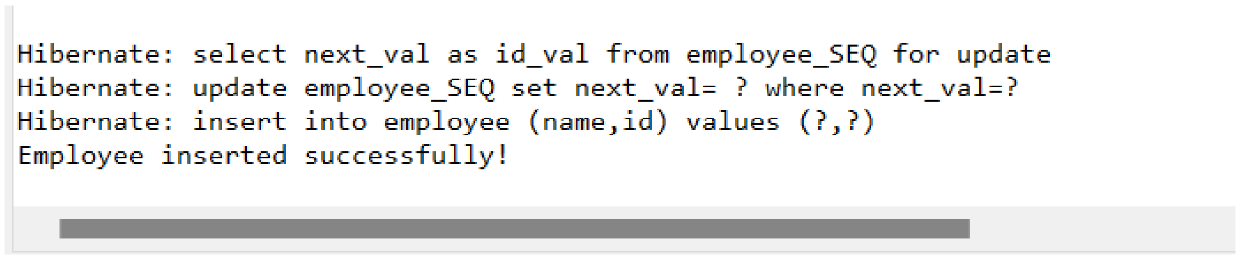
```java
package com.wipro.config;
import java.util.Properties;
import javax.sql.DataSource;
import org.apache.commons.dbcp2.BasicDataSource;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AvailableSettings;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.orm.hibernate5.HibernateTransactionManager;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
import org.springframework.transaction.annotation.EnableTransactionManagement;
@Configuration
@EnableTransactionManagement
@ComponentScan(basePackages = "com.wipro")
public class SpringConfig {

    @Bean
    public DataSource dataSource() {
        BasicDataSource ds = new BasicDataSource();
        ds.setUrl("jdbc:mysql://localhost:3306/wipro");
        ds.setUsername("root");
        ds.setPassword("#Mahadev7");
        ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
        return ds;
    }
    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
        sessionFactory.setDataSource(dataSource());
        sessionFactory.setPackagesToScan("com.wipro.entity");  // Fix incorrect package path

        Properties hibernateProperties = new Properties();
        hibernateProperties.put(AvailableSettings.DIALECT, "org.hibernate.dialect.MySQLDialect"); // Fix deprecated dialect
        hibernateProperties.put(AvailableSettings.HBM2DDL_AUTO, "update");
        hibernateProperties.put(AvailableSettings.SHOW_SQL, "true");

        sessionFactory.setHibernateProperties(hibernateProperties);
        return sessionFactory;
    }

    // ☑ Add this missing TransactionManager bean
    @Bean
    public HibernateTransactionManager transactionManager(SessionFactory sessionFactory) {
        return new HibernateTransactionManager(sessionFactory);
    }
}

```



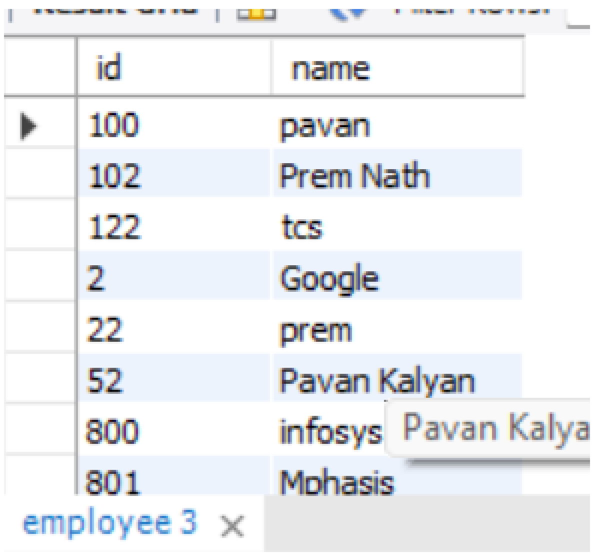
```java
package com.wipro.dao;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.wipro.entity.Employee;

@Transactional //used for transactions like commit or rollback
@Repository // exactly similar to @component= a component that Spring should automatically detect and manage in the application context
//as soon as @repostiroy exist people can understand this is a database related class
public class EmployeeDAO {

    @Autowired
    private SessionFactory sessionFactory;


    public void saveEmployee(Employee emp) {
        Session session = sessionFactory.getCurrentSession();
        session.persist(emp);
    }
}


```



```
package com.wipro;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

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

        EmployeeDAO employeeDAO = context.getBean(EmployeeDAO.class);

        Employee emp = new Employee();
        emp.setName("Prem Nath");
        employeeDAO.saveEmployee(emp);


        System.out.println("Employee inserted successfully!");

        context.close();
    }
}
```



```
Hibernate: select next_val as id_val from employee_SEQ for update
Hibernate: update employee_SEQ set next_val= ? where next_val=?
Hibernate: insert into employee (name,id) values (?,?)
Employee inserted successfully!
```



id	name
100	pavan
102	Prem Nath
122	tcs
2	Google
22	prem
52	Pavan Kalyan
800	infosys
801	Mphasis

1. Application Startup (`App.java`)

- This is the **starting point** of your application.
- It loads the Spring context using `SpringConfig.class` (your configuration file).
- It retrieves the `EmployeeDAO` bean from the Spring container.
- Creates an `Employee` object, sets the name, and calls `saveEmployee(emp)`.
- Finally, it **closes** the context.

👉 **Important File:** `App.java`
👉 **Key Role:** Starts the Spring container and initiates the employee insertion process.

⚙️ 2. Spring Configuration (`SpringConfig.java`)

- This file configures **database connection, Hibernate setup, and transaction management**.
- **Important Beans:**
  - `dataSource()`: Defines **database connection** (MySQL URL, username, password).
  - `sessionFactory()`: Configures **Hibernate session factory**.
  - `transactionManager()`: Enables **Spring's transaction management**.

👉 **Important File:** `SpringConfig.java`
👉 **Key Role:** Configures database connectivity and Hibernate setup.

🗄️ 3. Employee Entity (`Employee.java`)

- This file represents the **Employee table** in the database.
- It is annotated with `@Entity` and `@Table(name = "employee")` to define a Hibernate entity.
- The `@Id` and `@GeneratedValue` ensure Hibernate **automatically generates the ID** for employees.

👉 **Important File:** `Employee.java`
👉 **Key Role:** Defines the **structure** of the `employee` table in the database.

💾 4. DAO Layer (`EmployeeDAO.java`)

- This file is responsible for **database operations**.
- `@Repository` tells Spring **this is a database-related class**.
- `@Transactional` ensures **automatic transaction handling (commit/rollback)**.
- `session.persist(emp);` saves the employee object into the database.

👉 **Important File:** `EmployeeDAO.java`
👉 **Key Role: Handles database interactions** using Hibernate.

🔄 Overall Flow Summary

1. `App.java` starts execution.
2. Spring loads the configuration (`SpringConfig.java`).
3. `EmployeeDAO` is retrieved as a **Spring bean**.
4. A new **Employee object** is created and saved in the database.
5. Hibernate takes care of **inserting data** into the `employee` table.
6. Program prints **"Employee inserted successfully!"**
7. Application shuts down.

🎯 Key Takeaways

- `App.java` → **Starts the program.**
- `SpringConfig.java` → **Configures database and Hibernate.**
- `Employee.java` → **Defines the Employee entity (mapped to the database).**
- `EmployeeDAO.java` → **Saves the employee record using Hibernate.**

This is the basic **flow** of your **Spring + Hibernate** application. Let me know if you need any clarifications! 😊