What is the need of ORM, when we already have JDBC as persistence logic?

++++++++++++++++++++++
Limitations of JDBC API
++++++++++++++++++++++

1. JDBC API is database dependent, because query are database dependent this makes
indirectly java as platform dependent which is against to rule of java (WORA).

2. JDBC API Exceptions are checked exceptions.
In a project where ever we write jdbc api code, those code should compulsorily be handled through
"throws" or "try catch".

3. In JDBC API, we get ResultSet object which holds the data of a particular entity like Employee,
Student,Customer,..... The ResultSet object is not transferrable across the network, because it is not Serializable.

4. If the database structure is modified after developing JDBC project then that project does not execute because the
Queries are hardcoded inside java program,To execute the JDBC project we need to change the complete queries in our
code,this would increse the burden on the programmer and also it is time consuming.

5. In case of JDBC API, we don't have mechanism of caching to store the result obtained from database for future usage.
eg: select sid,sname,sage,saddress from student where sid = 10; => hit the database get the record
select sid,sname,sage,saddress from student where sid = 10; => again hit the database for the same record
This would decrease the efficeny of the application if we write a code in JDBC API.

6. Transaction support is not good in case of JDBC API because JDBC API supports Local Transaction but not Global Transaction.

7. JDBC API support positional parameters(?) not the named parameters(:name).
eg: select sid,sname,sage,saddress from student where sid<= ? and sage >= ? (positional parameters)
pstmt.setInt(1,10);
pstmt.setInt(2,35);

select sid,sname,sage,saddress from student where sid<= :id and sage >= :age(named parameters)
pstmt.setInt(id,10);
pstmt.setInt(age,35);

8. Strong SQLKnowledge is requried to write JDBC code(Queries are embeded inside java).

9. For every operation we do, we need to have the below mentioned 7 steps otherwise the code won't run
Create -> 7 steps
Read -> 7 steps
Update -> 7 steps

Delete -> 7 steps

Steps followed in JDBC API for persistence logic
a. Load and register the driver
b. Establish the connection
c. Create statement/PreparedStatement object

d. Execute the query(ResultSet/int)
e. Process the result

f. handle the exception if occured
g. close the resources.

a,b,c,f,g steps are common where as d,e would vary from query to query,This would increase redundancy.

10. While developing persistence logic in JDBC, we cannot enjoy inheritance, polymorphism, compoistion etc features of oops
because Query language doesn't permit the user to give inputs in the form of objects.

```
eg: insert into student(sname,sage,saddress) values(?,?,?);
      pstmt.setString(1,"sachin");
      pstmt.setInt(2,49);
      pstmt.setString(3,"MI");
      pstmt.executeUpdate();


Student std = new Student("sachin",49,"MI");
      insert into student(sname,sage,saddress) values(std); // no support like this is available in sql.
```

11. JDBC API doesn't support Versioning directly
JDBC API doesn't support TimeStamp directly

Versioning : Keeps track of no of times the record got modified.
Just now i opened an account in bank. ====> insert query executed(new record)

```
                                        balance(1000)
                                        balance(1000 + 35000= 36000)
ATM ---> 4 transactions in a month
       5th transaction(Extra money should be payed for transaction)
```

TimeStamp : To keep track of when the record is inserted and lastly modified.
If i want to get the balancesheet from particular date to particular date ====> select query will be triggered

To resolve the above mentioned problem we need to go for "ORM(Object Relation Mapping)".

What is OR-Mapping?
It stands for Object-Relational-Mapping.
Short Answer :: It is all about mapping/linking classes with db tables.

Note: hiberante internally uses JDBC API, Transaction API to perform the persitence operation.

Download hibernate using the following link ::
https://sourceforge.net/projects/hibernate/postdownload

Step1: Annotation used while mapping the java class to table are
	a. @Entity -> To specify the classname to map with tablename
	b. @Id -> To specify the primary key column name, which is mapped to field of entity.
	c. @Column -> To specify the column name of table which is mapped to field of an entity.

```
package in.pwskills.nitin.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity(name="EMPLOYEE")
public class Employee {

	@Id
	@Column(name="EID")
	private Integer eid;

	@Column(name="ENAME")
	private String ename;

	@Column(name="EAGE")
	private Integer eage;

	@Column(name="EADDRESS")
	private String eaddress;

	public Employee() {
		System.out.println("USED BY HIBERNATE INTERNALLY");
	}

	public Integer getEid() {
		return eid;
	}

	public void setEid(Integer eid) {
		this.eid = eid;
	}

	public String getEname() {
		return ename;
	}

	public void setEname(String ename) {
		this.ename = ename;
	}

	public Integer getEage() {
		return eage;
	}

	public void setEage(Integer eage) {
		this.eage = eage;
	}

	public String getEaddress() {
		return eaddress;
```

```
	}

	public void setEaddress(String eaddress) {
		this.eaddress = eaddress;
	}

	@Override
	public String toString() {
		return "Employee [eid=" + eid + ", ename=" + ename + ", eage=" + eage +
", eaddress=" + eaddress + "]";
	}

}


Step2: Write the db information through configuration file.
		Note: Good practise is to give the file name as "hibernate.cfg.xml"

hiberante.cfg.xml
=================
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!--  To inform HIBERNATE to generate the query specific to database engine
-->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>

        <!--  To inform HIBERNATE about the database details -->
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/pwskillsbatch</
property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root123</property>

        <!-- To inform HIBERNATE to display the queries generated during execution
-->
        <property name = "hibernate.show_sql">true</property>

        <!-- To inform HIBERNATE to display the queries in formatted style
generated during execution -->
        <property name = "hibernate.format_sql">true</property>

        <!--  To inform HIBERNATE whether to create a table or use existing table
-->
        <property name="hibernate.hbm2ddl.auto">create</property>

        <!-- Specifying the mapping information -->
        <mapping class="in.pwskills.nitin.entity.Employee"/>

    </session-factory>
</hibernate-configuration>

TestApp.java
```

++++++++++++
```
package in.pwskills.nitin.main;

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

import in.pwskills.nitin.entity.Employee;

public class TestApp {

	public static void main(String[] args) {

		Employee emp = new Employee();
		emp.setEid(10);
		emp.setEname("sachin");
		emp.setEage(49);
		emp.setEaddress("MI");

		// Activate the HIBERNATE software
		Configuration configuration = new Configuration();

		// load hibernate.cfg.xml file into configuration object
		configuration.configure();

		// Build sessionFactory object using configuration object
		//[load driver,establish connection,create PreparedStatement obj)
		SessionFactory sessionFactory = configuration.buildSessionFactory();

		// To perform operation(task) create one Session object using
SessionFactory
		Session session = sessionFactory.openSession();

		// Begin the transaction w.r.t particular session
		Transaction transaction = session.beginTransaction();

		// Perform insert operation
		session.save(emp);

		// Commit the operation
		transaction.commit();

		// close the session
		session.close();

	}

}
```

output
 Record got save as a row in table called student

2. update()
 => To perform update() , we need to ensure the primary key value used for searching should be availble compulsorily otherwise it would
    result in "org.hibernate.StaleObjectStateException".

code

```
=====
Employee employee = new Employee();
      employee.setEid(7);// eid = 7 doesn't exists then it would result in
exception, if it exists then it  would update.
      employee.setEname("messi");
      employee.setEage(36);
      employee.setEaddress("Argentina");
      session.update(employee);

output
======
 Hibernate:
    update
        EMPLOYEE
    set
        EADDRESS=?,
        EAGE=?,
        ENAME=?
    where
        EID=?


3. saveOrUpdate()
       => This method first would perform select operation
            a. if record found then update
            b. if record not available then perform insert operation.

code
====
 Employee employee = new Employee();
  employee.setEid(7); //eid = 7 not found so perform insertion operation
  employee.setEname("messi");
  employee.setEage(36);
  employee.setEaddress("Argentina");

  session.saveOrUpdate(employee);

output
======
Hibernate:
    select
        employee_.EID,
        employee_.EADDRESS as eaddress2_0_,
        employee_.EAGE as eage3_0_,
        employee_.ENAME as ename4_0_
    from
        EMPLOYEE employee_
    where
        employee_.EID=?
Hibernate:
    insert
    into
        EMPLOYEE
        (EADDRESS, EAGE, ENAME, EID)
    values
        (?, ?, ?, ?)


case 2: record found, so do update
```

```
Employee employee = new Employee();
      employee.setEid(10);
      employee.setEname("tendulkar");
      employee.setEage(49);
      employee.setEaddress("IND");

      session.saveOrUpdate(employee);

Output
======
Hibernate:
    select
        employee_.EID,
        employee_.EADDRESS as eaddress2_0_,
        employee_.EAGE as eage3_0_,
        employee_.ENAME as ename4_0_
    from
        EMPLOYEE employee_
    where
        employee_.EID=?

Hibernate:
    update
        EMPLOYEE
    set
        EADDRESS=?,
        EAGE=?,
        ENAME=?
    where
        EID=?
```