Hibernate ORM

ORM

1.Object Relational Mapping (ORM)

DEFINITION/WHAT

Is a programming technique for converting data between incompatible type systems using object-oriented programming languages

It is a way of relating an object, with a record in a table in the database.

An object in a programming language becomes a record in a table and vice versa.

It also relates the operations of an object(get and set) with the operations on a record(insert/update/delete) in a table.

Modelling an object into a record and vice-versa

NEED/WHY

- ✓ Need 1: To enable programs to interact with the database in the native language of the database.
- ✓ Need 2: To eliminate hardcoding/usage of SQL statements in the program.

IMPLEMENTATION/HOW IT WORKS

REAL TIME EXAMPLE

Generally, when we want to persist some data in database using OR Mapping, we store data in Objects, which require class definition.

2.Object

DEFINITION/WHAT

Is an instance of a class.

Holds the data and pre-defined functions.

Combines the data and the functionality.

NEED/WHY

- ✓ Need 1: To handle structured data.
- ✓ Need 2: To access data in a predefined way.
- ✓ Need 3: To relate functionality with the data.

IMPLEMENTATION/HOW IT WORKS

An Object is created when JVM encounters new Keyword.

Objects are created in the heap memory and it's reference is stored in the stack.

REAL TIME EXAMPLE

Laptop is a class with RAM memory, SSD, keypad as attributes and HP, Lenovo, Dell are Objects of Laptop

ADDITIONAL INFORMATION

Multiple Objects can be created for Single Class.

Memory is allocated depending on size of attributes (datatypes) of the Object.

3. Relation / Relational

DEFINITION/WHAT

In RDBMS terminology, a Relation is a table in the database.

A table has rows and columns.

Tables are logical structures.

NEED/WHY

- ✓ Need 1: To structure the data.
- ✓ Need 2 : To group the related data.
- ✓ Need 3 : To support transactions.

REAL TIME EXAMPLE

Project table

Project ID	**Project name**	**Start date**	**End date**
BNKHSBC2021	HSBC Core Banking	20-Jun-2018	11-Feb-2023
RETCOMSHL	Shell Retail CMS	13-Apr-2020	27-Dec-2023
PGCRM	Procter and Gamble CRM	03-Dec-2018	15-Dec-2023

4. Mapping in ORM

DEFINITION/WHAT

Relating an object's structure in an OOP language to the structure of a table in the database.

Relating/Linking the fields of an object to the fields in the table in such a way that any changes done to the fields in an object should be reflected.

Relating the operations of an object to the operations on a table.

NEED /WHY

- ✓ Need 1: To convert programming language operations to the equivalent database operations
- ✓ Need 2: To maintain uniformity of structure between the program and the database.
- ✓ Need 3: To support Logical and Physical Data Independence as emphasized in Codd's rule for Relational databases.

REAL TIME EXAMPLE

A Project object having fields like ProjectID, ProjectName, StartDate and EndDate is related/mapped to the Project table having fields like ProjectID, ProjectName, StartDate and EndDate

5. Persistence

DEFINITION/WHAT

- Place to store/persist the data for some time.
- Data can be accessed and managed(querries, updates, deletes, inserts etc..)
- Usually a temporary storage. Even permanent storage can come under this category.
- Data is serialized before storing it and de-serialized after retrieving it.

NEED/WHY

- ✓ Need 1: Data handled inside a program is temporary and is lost when the program/application terminates. So store/persist the data so that it is not lost.
- ✓ Need 2: Data can be retrieved as and when required.

REAL TIME EXAMPLE

Software products like Reddis Framework, Apache Cassandra, Oracle Times Ten etc.

6. Hibernate

DEFINITION/WHAT

- Dictionary meaning :
    - Save the state so that it can be restored/retrieved later.

- Remain inactive/dormant and become active when some event happens

- A framework for persisting data in the language of programming.
- Combination of ORM and Persistence.
- Written in Java. Also available for .NET platform as NHibernate.
- Built over JPA and JDBC.
- Avoids hardcoding of SQL upto some extent.
- Bridges the problem of non-standard SQL implementations provided by the database softwares.
- Final Class cannot be extended

NEED/WHY

- ✓ Need 1: Communicate in the way of programming language rather than using SQL.
- ✓ Need 2: Support different variants of SQL.
- ✓ Need 3: To make the program database independent.

7. Hibernate

DEFINITION/WHAT

- Automatic table creation.
- Open Source and Light Weight.
- Good performance.
- Simplifies Join and Subquerry.
- Easier META-DATA management.

NEED/WHY

- ✓ Need 1: To manage the database easily. No need to pre-create tables.
- ✓ Need 2: For faster querries .
- ✓ Need 3: To make the program database independent.

8. POJO

DEFINITION/WHAT

- Plain Old Java Object.
- A class with setter and getter methods for each variable.
- Getters and setters forms the interface for the objects of POJO class.
- Variables are private and methods are public.
- Map/relate the POJO class to the table in the database.

- It should have accessible constructors.

NEED/WHY

- ✓ Need 1: To relate a class to the structure of the table in the database.
- ✓ Need 2: To sync the data and the operations between Java objects and the table in the database.
- ✓ Need 3: To facilitate structural interaction between the Java program and the database.

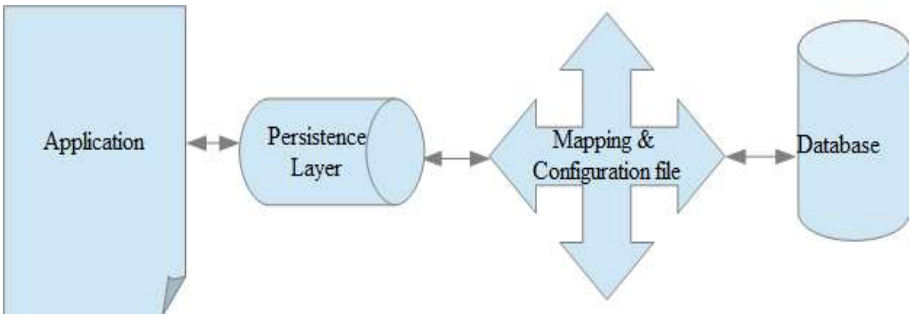
REAL TIME EXAMPLE

```
class Book{
        private Int   ISBN;
        private String  title;
        public   Book(){
        }
        public   void  setISBN(int    ISBN)
        {
        this.ISBN  =  ISBN;
        }
        public  void    setTitle(String   title){
        this.title  =   this.title;
        }
        public  String    getTitle(){
        return   title;
        }
        publc   int   getISBN(){
        return   ISBN;
        }
}
```

9. Hibernate architecture

DEFINITION/WHAT

- 4 layered architecture.
- Application-Persistence-Mapping-Database.
- Changes done in one layer will not affect the other layer.
- Layers are independent.



NEED/WHY

- ✓ Need 1: To make programs/applications independent of changes in the configuration.
- ✓ Need 2: To make programs accustom to any database configuration dynamically.

REAL TIME EXAMPLE

An Eclipse / IntelliJ project having Java client program and the hibernate.cfg.xml configuration file.

10. Hibernate.cfg.xml

DEFINITION/WHAT

Provides all the details that is required to connect to the database.

Hibernate connection factories loads this file first and establishes connections to the database.

Mandatory file in XML configuration

NEED/WHY

- ✓ Need 1: To supply connection details to the hibernate connection factory to manage the connections to the database.
- ✓ Need 2: To make the program flexible and adaptable to the backend database. If the database is changed in the backend, programs need not be changed and recompiled.
- ✓ Need 3: Avoids hardcoding of connection details in the program which makes the program more tightly coupled with the database.

REAL TIME EXAMPLE

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>

      <property name = "hibernate.dialect">
         org.hibernate.dialect.MySQLDialect
      </property>

      <property name = "hibernate.connection.driver_class">
         com.mysql.jdbc.Driver
      </property>

      <!-- Assume test is the database name -->

      <property name = "hibernate.connection.url">
         jdbc:mysql://localhost/test
      </property>

      <property name = "hibernate.connection.username">
         root
      </property>

      <property name = "hibernate.connection.password">
         root123
      </property>

      <!-- List of XML mapping files -->
      <mapping resource =     "Book.hbm.xml"/>

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

11. XML Mapping file

DEFINITION/WHAT

Contains all the mapping information.

Maps/relates the classes and variables to the tables and fields in the tables.

Mandatory file in XML mapping

NEED/WHY

- ✓ Need 1: To map/relate the POJO class and it's fields to the tables and fields in the database.
- ✓ Need 2: To specifiy the keys, constraints and data types that is followed in the database .
- ✓ Need 3: Avoids hardcoding of mapping details in the program. Hardcoding the mapping makes the program more lengthier and difficult to manage. Separation of mapping from the program

REAL TIME EXAMPLE

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name = "Book" table = "Book">

      <meta attribute = "class-description">
         This class contains the detaild of the book.
      </meta>

      <id name = "id" type = "int" column = "ISBN">
         <generator class="native"/>
      </id>

      <property name = "title" column = "first_name" type = "string"/>
      <property name = "price" column = "price" type = "int"/>
   </class>
</hibernate-mapping>
```

ADDITIONAL INFORMATION

May lead to more no of files in the program

12. StandardServiceRegistry

DEFINITION/WHAT

Registry of all databases.

Encapsulates the information about the database that is provided in the **hibernate.cfg.xml**.

Used for connecting to the database.

It is an interface

NEED/WHY

- ✓ Need 1: To convert the configuration info mentioned in XML to the underlying JDBC code.
- ✓ Need 2: To pool the connections to a database so that it can be used efficiently.
- ✓ Need 3: To enable the client look up the registry and retrieve the connections to the database.

REAL TIME EXAMPLE

```
StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

13. MetaData

DEFINITION/WHAT

It is an interface.

Encapsulates META information about the database, tables and other objects.

It is a catalogue describing the contents of the database.

NEED/WHY

- ✓ Need 1: To extract META information about the database.
- ✓ Need 2: To run querries, transactions etc, META information has to be checked.

REAL TIME EXAMPLE

```
StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
```

14. SessionFactory

DEFINITION/WHAT

It is an interface.

Factory to create sessions. Extract sessions from the factory.

Sessions are pre-created. Just retrieve it from the pool.

NEED/WHY

- ✓ Need 1: To manage the sessions efficiently.
- ✓ Need 2: To reuse the sessions once it is used.
- ✓ Need 3 : To share a single connection among multiple threads. Many threads can communicate with the same database concurrently using the same connection.

REAL TIME EXAMPLE

```
StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();
```

15. Session

DEFINITION/WHAT

It is an interface.

Represents a session with the database.

It is the underlying structure through which the transactions are carried out.

NEED/WHY

- ✓ Need 1: To perform multiple transactions as a bunch.
- ✓ Need 2: To exercise control over all the transactions with the database.

REAL TIME EXAMPLE

```
StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();
```

16. Session

DEFINITION/WHAT

It is an interface.

Represents a session with the database.

It is the underlying structure through which the transactions are carried out.

NEED/WHY

- ✓ Need 1: To perform multiple transactions as a bunch.
- ✓ Need 2: To exercise control over all the transactions with the database.

REAL TIME EXAMPLE

```
StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();
```

17. Transaction

DEFINITION/WHAT

A basic unit of work in the database.

A set of related actions performed as an atomic operation, on the database.

It exclusively holds the resources by locking them until the end of the transaction. This prevents concurrent modifications and subsequent inconsistencies of the data.

It ends when the commit/rollback is executed.

NEED/WHY

- ✓ Need 1: To perform a large no of related operations on a database.
- ✓ Need 2: To minimize the network and database access so that database operations can be performed faster.
- ✓ Need 3: To ensure that the database is in consistent state at the end of the transaction.

REAL TIME EXAMPLE

```
StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();
```

18. Saving the data

DEFINITION/WHAT

Save or update the object in the persistence layer before saving it in the database.

Generate the database specific SQL statements.

NEED/WHY

- ✓ Need 1: To increase the opaqueness of the transactions.
- ✓ Need 2: To maintain a copy in the persistence layer so that if the data is lost it can be retransmitted.
- ✓ Need 3: To cache the data so that if the same has to be retransmitted, it can be done from here.

REAL TIME EXAMPLE

```
StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

session.save(e1);
```

19. Commit

DEFINITION/WHAT

Complete the transaction by saving the data in the database.

Releases the locks acquired on the records in the table.

Ensures that the database is in consistent state.

NEED/WHY

- ✓ Need 1: To save all the changed data in one single operation into the database.
- ✓ Need 2: To ensure prevalence of ACID properties of a transaction.
- ✓ Need 3: To prevent loss of data due to application/transaction failures.

REAL TIME EXAMPLE

```
StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
SessionFactory factory = meta.getSessionFactoryBuilder().build();
Session session = factory.openSession();
Transaction t = session.beginTransaction();
Employee e1=new Employee();
e1.setId(101);
e1.setFirstName("Aakash");
e1.setLastName("Gowda");
session.save(e1);
t.commit();
```

20. Close function

DEFINITION/WHAT

Close () function can be called on session and session factory.

Generally called at the end of all transactions.

NEED/WHY

- ✓ Need 1: To close the connection and release the memory held by the connection.
- ✓ Need 2: To ensure that unused resources are put in place.

REAL TIME EXAMPLE

```
StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
SessionFactory factory = meta.getSessionFactoryBuilder().build();
Session session = factory.openSession();
Transaction t = session.beginTransaction();
Employee e1=new Employee();
e1.setId(101);
e1.setFirstName("Aakash");
e1.setLastName("Gowda");
session.save(e1);
t.commit();
session.close();
factory.close();
```

20. Annotation Configuration

DEFINITION/WHAT

Annotations introduced in Java 5 to develop shorter applications.

In some applications XML coding/configuration was more.

Annotations can be used wherever lengthy coding is required.

NEED/WHY

- ✓ Need 1: To reduce amount of XML coding or totally avoid it.
- ✓ Need 2: To reduce the number of XML configuration files in the application.

REAL TIME EXAMPLE

Annotations used to map the entities to tables, variables to fields

Annotations for ID mapping, ID generation strategy, type of mapping

21. Entity mapping - @Entity

DEFINITION/WHAT

Maps/relates a POJO class to a table in the database.
POJO class name need not be same as the table name.

By default it assumes that the POJO class name and the table name are same.

NEED/WHY

- ✓ To map/relate a POJO class to a table.
- ✓ Very important and basic mapping in hibernate
- ✓ To ensure that the generated DDLs will operate on the named table.

REAL TIME EXAMPLE

```
import   javax.persistence.Entity;
@Entity
class Book{
        private Int   ISBN;
        private String  title;
        public   Book(){
        }
        public   void  setISBN(int    ISBN)
        {
        this.ISBN  =  ISBN;
        }
        public  void    setTitle(String   title){
        this.title  =   this.title;
        }
        public  String    getTitle(){
        return   title;
        }
        publc   int   getISBN(){
        return   ISBN;
        }
}
@Entity(name=”Author”)
class Author
{
int   authorID;  String  authorName;
public int getAuthorID() {
return authorID;
}
        public void setAuthorID(int  authorID) {
this.authorID = authorID;
```

```
}
        public String getAuthorName() {
return authorName;
}
        public void setAuthorName(String  authorName) {
this.authorName = authorName;
}
}
```

22. Entity/Table mapping - @Table

DEFINITION/WHAT

Maps/relates a POJO class to a table

Distinguishes Entity mapping from a Table mapping

Relates .

NEED/WHY

- ✓ To separate a table mapping with entity mapping.
- ✓ Very important and basic mapping in hibernate
- ✓ To ensure that the generated DDLs will operate on the named table.

REAL TIME EXAMPLE

```
import   javax.persistence.Entity;
@Entity
class Book{
        private Int    ISBN;
        private String  title;
        public   Book(){
        }
        public   void  setISBN(int    ISBN)
        {
        this.ISBN  =  ISBN;
        }
        public  void    setTitle(String   title){
        this.title  =   this.title;
        }
        public  String    getTitle(){
        return   title;
        }
        publc   int   getISBN(){
        return   ISBN;
        }
}
@Entity
@Table(name=”Author”)
```

```
class Author
{
int  authorID;  String  authorName;
public int getAuthorID() {
return authorID;
}
        public void setAuthorID(int  authorID) {
this.authorID = authorID;
}
        public String getAuthorName() {
return authorName;
}
        public void setAuthorName(String  authorName) {
this.authorName = authorName;
}
}
```

23. ID mapping - @Id

DEFINITION/WHAT

Maps/relates a variable to a primary key field in the database

Like in the table, the value of the variable marked as Id must be unique when it is pushed into the persistence layer.

Uniqueness and nullness is checked in the persistence layer before it is done in the database.

NEED/WHY

- ✓ To implement a check for uniqueness in the persistence layer. It is faster.
- ✓ To avoid uniqueness and null checking in the database as it is costly in terms of time.
- ✓ To ensure that the corrections are made even before the data is posted to the database.

REAL TIME EXAMPLE

```
import  javax.persistence.Entity;
@Entity
class Book{
        private Int  ISBN;
        private String  title;
        public  Book(){
        }
        public  void  setISBN(int  ISBN)
        {
        this.ISBN  =  ISBN;
        }
        public  void  setTitle(String  title){
        this.title  =  this.title;
```

```
        }
        public String   getTitle(){
        return  title;
        }
@Id
        publc  int  getISBN(){
        return  ISBN;
        }
}
```

24. Annotation Configuration

DEFINITION/WHAT

.

NEED/WHY

- ✓ To implement a check for uniqueness in the persistence layer. It is faster.
- ✓ To avoid uniqueness and null checking in the database as it is costly in terms of time.
- ✓ To ensure that the corrections are made even before the data is posted to the database.

REAL TIME EXAMPLE

```
import  javax.persistence.Entity;
@Entity
class Book{
        private Int   ISBN;
        private String  title;
        public  Book(){
        }
        public  void  setISBN(int   ISBN)
        {
        this.ISBN  =  ISBN;
        }
        public  void   setTitle(String  title){
        this.title  =  this.title;
        }
        public  String   getTitle(){
        return  title;
        }
@Id
        publc  int  getISBN(){
        return  ISBN;
        }
}
```