

1 Refer to : <http://blog.woniper.net/255?category=531455>
2 <http://www.javajigi.net/pages/viewpage.action?pageId=5924>
3 https://www.slideshare.net/zipkyh/spring-datajpa?next_slideshow=1
4 <https://www.tutorialspoint.com/jpa/index.htm>

5 6 1. JPA란 무엇인가?

7 1)JPA(Java Persistent API)

8 2)JPA는 여러 ORM 전문가가 참여한 EJB 3.0 스펙 작업에서 기존 EJB ORM이던 Entity Bean을 JPA라고 바꾸고
JavaSE, JavaEE를 위한 영속성(persistence) 관리와 ORM을 위한 표준 기술이다.

9 3)A collection of classes and methods to persistently store the vast amounts of data into a database.

10 4)The basic understanding of Persistence (storing the copy of database object into temporary
memory), and The understanding of JAVA Persistence API (JPA).

11 5)JPA는 ORM 표준 기술로 Hibernate, OpenJPA, EclipseLink, TopLink Essentials과 같은 구현체가 있고 이에
표준 인터페이스가 바로 JPA이다.

12 6)ORM(Object Relational Mapping)이란 RDB 테이블을 객체지향적으로 사용하기 위한 기술이다.

13 7)RDB 테이블은 객체지향적 특징(상속, 다형성, 레퍼런스, 오브젝트 등)이 없고 자바와 같은 언어로 접근하기 쉽지 않다.

14 8)이 때문에 ORM을 사용해 오브젝트와 RDB 사이에 존재하는 개념과 접근을 객체지향적으로 다루기 위한 기술이다.

15 9)장점

16 -객체지향적으로 데이터를 관리할 수 있기 때문에 비즈니스 로직에 집중 할 수 있으며, 객체지향 개발이 가능하다.

17 -테이블 생성, 변경, 관리가 쉽다. (JPA를 잘 이해하고 있는 경우)

18 -로직을 쿼리에 집중하기 보다는 객체 자체에 집중 할 수 있다.

19 -빠른 개발이 가능하다.

20 10)단점

21 -어렵다. 장점을 더 극대화 하기 위해서 알아야 할게 많다.

22 -잘 이해하고 사용하지 않으면 데이터 손실이 있을 수 있다. (persistence context)

23 -성능상 문제가 있을 수 있다.(이 문제 또한 잘 이해해야 해결이 가능하다.)

24

25 11)History

26 -Earlier versions of EJB, defined persistence layer combined with business logic layer using
javax.ejb.EntityBean Interface.

27 -While introducing EJB 3.0, the persistence layer was separated and specified as JPA 1.0 (Java
Persistence API).

28 -The specifications of this API were released along with the specifications of JAVA EE5 on May
11, 2006 using JSR 220.

29 -JPA 2.0 was released with the specifications of JAVA EE6 on December 10, 2009 as a part of
Java Community Process JSR 317.

30 -JPA 2.1 was released with the specification of JAVA EE7 on April 22, 2013 using JSR 338.

31

32 12)JPA Providers

33 -JPA is an open source API, therefore various enterprise vendors such as Oracle, Redhat,
Eclipse, etc. provide new products by adding the JPA persistence flavor in them.

34 -Some of these products include: Hibernate, EclipseLink, Toplink, Spring Data JPA, etc.

35

36

37 2. JPA - Architecture

38 -Java Persistence API is a source to store business entities as relational entities.

39 -It shows how to define a PLAIN OLD JAVA OBJECT (POJO) as an entity and how to manage
entities with relations.

40 1)Class Level Architecture

41 -The following table describes each of the units shown in the above architecture.

42 Units	Description
43 EntityManagerFactory	This is a factory class of EntityManager. It creates and manages multiple EntityManager instances.
44 EntityManager	It is an Interface, it manages the persistence operations on objects. It works like factory for Query instance.
46 Entity	Entities are the persistence objects, stores as records in the database.
47 EntityManagerTransaction	It has one-to-one relationship with EntityManager. For each EntityManager, operations are maintained by EntityManagerTransaction class.
48 Persistence	This class contain static methods to obtain EntityManagerFactory instance.
49 Query	This interface is implemented by each JPA vendor to obtain relational objects that meet the criteria.

51

52

53

-The above classes and interfaces are used for storing entities into a database as a record.

-They help programmers by reducing their efforts to write codes for storing data into a database so that they can concentrate on more important activities such as writing codes for mapping the classes with database tables.

2)JPA Class Relationships

- The relationship between EntityManagerFactory and EntityManager is one-to-many.
 - It is a factory class to EntityManager instances.
- The relationship between EntityManager and EntityTransaction is one-to-one.
 - For each EntityManager operation, there is an EntityTransaction instance.
- The relationship between EntityManager and Query is one-to-many.
 - Many number of queries can execute using one EntityManager instance.
- The relationship between EntityManager and Entity is one-to-many.
 - One EntityManager instance can manage multiple Entities.

3. ORM

1)ORM is a programming ability to covert data from object type to relational type and vice versa.

2)The main feature of ORM is mapping or binding an object to its data in the database.

3)While mapping we have to consider the data, type of data and its relations with its self-entity or entity in any other table.

4)Advanced Features

- Idiomatic persistence : It enables you to write the persistence classes using object oriented classes.
- High Performance : It has many fetching techniques and hopeful locking techniques.
- Reliable : It is highly stable and eminent. Used by many industrial programmers.

5)ORM Architecture

-Phase1

- The first phase, named as the Object data phase contains POJO classes, service interfaces and classes.
- It is the main business component layer, which has business logic operations and attributes.
- For example let us take an employee database as schema-
- Employee POJO class contain attributes such as ID, name, salary, and designation.
- And methods like setter and getter methods of those attributes.
- Employee DAO/Service classes contains service methods such as create employee, find employee, and delete employee.

-Phase 2

- The second phase named as mapping or persistence phase which contains JPA provider, mapping file (ORM.xml), JPA Loader, and Object Grid.
- JPA Provider : The vendor product which contains JPA flavor (javax.persistence).
- For example EclipseLink, Toplink, Hibernate, etc.
- Mapping file : The mapping file (ORM.xml) contains mapping configuration between the data in a POJO class and data in a relational database.
- JPA Loader : The JPA loader works like cache memory, which can load the relational grid data. It works like a copy of database to interact with service classes for POJO data (Attributes of POJO class).
- Object Grid : The Object grid is a temporary location which can store the copy of relational data, i.e. like a cache memory.
- All queries against the database is first effected on the data in the object grid.
- Only after it is committed, it effects the main database.

-Phase 3

- The third phase is the Relational data phase.
- It contains the relational data which is logically connected to the business component.
- As discussed above, only when the business component commit the data, it is stored into the database physically.
- Until then the modified data is stored in a cache memory as a grid format.
- Same is the process for obtaining data.

-The mechanism of the programmatic interaction of above three phases is called as object relational mapping.

6)Mapping.xml

-The mapping.xml file is to instruct the JPA vendor for mapping the Entity classes with database tables.

-Let us take an example of Employee entity which contains four attributes.

-The POJO class of Employee entity named Employee.java is as follows:

```
public class Employee {  
    private int eid;  
    private String ename;  
    private double salary;  
    private String deg;  
  
    public Employee(int eid, String ename, double salary, String deg) {  
        this.eid = eid;  
        this.ename = ename;  
        this.salary = salary;  
        this.deg = deg;  
    }  
  
    public Employee( ) {}  
  
    public int getEid( ) {  
        return eid;  
    }  
  
    public void setEid(int eid) {  
        this.eid = eid;  
    }  
  
    public String getEname( ) {  
        return ename;  
    }  
  
    public void setEname(String ename) {  
        this.ename = ename;  
    }  
  
    public double getSalary( ) {  
        return salary;  
    }  
  
    public void setSalary(double salary) {  
        this.salary = salary;  
    }  
  
    public String getDeg( ) {  
        return deg;  
    }  
  
    public void setDeg(String deg) {  
        this.deg = deg;  
    }  
}
```

-The above code is the Employee entity POJO class.

-It contain four attributes eid, ename, salary, and deg.

-Consider these attributes are the table fields in the database and eid is the primary key of this table.

-Now we have to design hibernate mapping file for it.

-The mapping file named mapping.xml is as follows:

```
<? xml version="1.0" encoding="UTF-8" ?>  
  
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm  
        http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"  
    version="1.0">  
  
    <description> XML Mapping file</description>
```

```

173     <entity class="Employee">
174         <table name="EMPLOYEE" />
175         <attributes>
176             <id name="eid">
177                 <generated-value strategy="TABLE" />
178             </id>
179             <basic name="ename">
180                 <column name="EMP_NAME" length="100" />
181             </basic>
182             <basic name="salary">
183             </basic>
184             <basic name="deg">
185             </basic>
186         </attributes>
187     </entity>
188 </entity-mappings>

```

- The above script for mapping the entity class with database table. In this file
- <entity-mappings> : tag defines the schema definition to allow entity tags into xml file.
- <description> : tag defines description about application.
- <entity> : tag defines the entity class which you want to convert into table in a database. Attribute class defines the POJO entity class name.
- <table> : tag defines the table name. If you want to keep class name as table name then this tag is not necessary.
- <attributes> : tag defines the attributes (fields in a table).
- <id> : tag defines the primary key of the table. The <generated-value> tag defines how to assign the primary key value such as Automatic, Manual, or taken from Sequence.
- <basic> : tag is used for defining remaining attributes for table.
- <column-name> : tag is used to define user defined table field name.

7)Annotations

- Generally Xml files are used to configure specific component, or mapping two different specifications of components.
- In our case, we have to maintain xml separately in a framework.
- That means while writing a mapping xml file we need to compare the POJO class attributes with entity tags in mapping.xml file.
- Here is the solution: In the class definition, we can write the configuration part using annotations.
- The annotations are used for classes, properties, and methods.
- Annotations starts with '@' symbol.
- Annotations are declared before the class, property or method is declared.
- All annotations of JPA are defined in javax.persistence package.
- Here follows the list of annotations used in our examples
- @Entity
 - This annotation specifies to declare the class as entity or a table.
- @Table
 - This annotation specifies to declare table name.
- @Basic
 - This annotation specifies non constraint fields explicitly.
- @Embedded
 - This annotation specifies the properties of class or an entity whose value instance of an embeddable class.
- @Id
 - This annotation specifies the property, use for identity (primary key of a table) of the class.
- @GeneratedValue
 - This annotation specifies, how the identity attribute can be initialized such as Automatic, manual, or value taken from sequence table.
- @Transient

---This annotation specifies the property which is not persistent i.e. the value is never stored into database.

--@Column
---This annotation is used to specify column or attribute for persistence property.

--@SequenceGenerator
---This annotation is used to define the value for the property which is specified in @GeneratedValue annotation. It creates a sequence.

--@TableGenerator
---This annotation is used to specify the value generator for property specified in @GeneratedValue annotation. It creates a table for value generation.

--@AccessType
---This type of annotation is used to set the access type. If you set @AccessType(FIELD) then Field wise access will occur. If you set @AccessType(PROPERTY) then Property wise access will occur.

--@JoinColumn
---This annotation is used to specify an entity association or entity collection. This is used in many- to-one and one-to-many associations.

--@UniqueConstraint
---This annotation is used to specify the field, unique constraint for primary or secondary table.

--@ColumnResult
---This annotation references the name of a column in the SQL query using select clause.

--@ManyToMany
---This annotation is used to define a many-to-many relationship between the join Tables.

--@ManyToOne
---This annotation is used to define a many-to-one relationship between the join Tables.

--@OneToMany
---This annotation is used to define a one-to-many relationship between the join Tables.

--@OneToOne
---This annotation is used to define a one-to-one relationship between the join Tables.

--@NamedQueries
---This annotation is used for specifying list of named queries.

--@NamedQuery
---This annotation is used for specifying a Query using static name.

8)Java Bean Standard

- Java class, encapsulates the instance values and behaviors into a single unit called object.
- Java Bean is a temporary storage and reusable component or an object.
- It is a serializable class which has default constructor and getter & setter methods to initialize the instance attributes individually.

9)Bean Conventions

- Bean contains the default constructor or a file that contains serialized instance.
- Therefore, a bean can instantiate the bean.
- The properties of a bean can be segregated into Boolean properties and non-Boolean properties.
- Non-Boolean property contains getter and setter methods.
- Boolean property contains setter and is method.
- Getter method of any property should start with small lettered 'get' (java method convention) and continued with a field name that starts with capital letter.
- E.g. the field name is 'salary' therefore the getter method of this field is 'getSalary ()'.
- Setter method of any property should start with small lettered 'set' (java method convention), continued with a field name that starts with capital letter and the argument value to set to field.
- E.g. the field name is 'salary' therefore the setter method of this field is 'setSalary (double sal)'.
- For Boolean property, is method to check if it is true or false. E.g. the Boolean property 'empty', the is method of this field is 'isEmpty ()'.

4. Lab

1)JPA Project 생성

- In Package Explorer > right-click > New > Other > JPA > JPA Project
- Project name : Demo
- Target runtime : jdk 1.8.0_162 > Next
- Platform : Generic 2.1
- Type : User Library > Download library...
- Download Library : EclipseLink 2.5.2 > Next > Check I accpet... > Finish
- Finish

-Open Perspective

2)Adding H2 database connector to Project

- Go to Project properties > Java Build Path by right click on it.
- Click on Add External Jars.
- Select C:\Program Files (x86)\H2\bin\h2-1.4.197.jar > 열기
- Apply and Close

3)Entity Managers

- The main modules for this example are as follows:

Model or POJO

Employee.java

Persistence

Persistence.xml

Service

CreatingEmployee.java

UpdatingEmployee.java

FindingEmployee.java

DeletingEmployee.java

4)Creating Entities

- src/com.javasoft package 생성 > right-click > New > Class
- com.javasoft.Employee.java

```
package com.javasoft;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
```

```
import javax.persistence.Table;
```

```
@Entity
```

```
@Table
```

```
public class Employee {
```

```
    @Id
```

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

```
    private int eid;
```

```
    private String ename;
```

```
    private double salary;
```

```
    private String deg;
```

```
    public Employee(int eid, String ename, double salary, String deg) {
```

```
        this.eid = eid;
```

```
        this.ename = ename;
```

```
        this.salary = salary;
```

```
        this.deg = deg;
```

```
    }
```

```
    public Employee( ) {}
```

```
    public int getEid( ) {
```

```
        return eid;
```

```
    }
```

```
    public void setEid(int eid) {
```

```
        this.eid = eid;
```

```
    }
```

```
    public String getEname( ) {
```

```
        return ename;
```

```
    }
```

```
    public void setEname(String ename) {
```

```
        this.ename = ename;
```

```
    }
```

```

354
355     public double getSalary( ) {
356         return salary;
357     }
358
359     public void setSalary(double salary) {
360         this.salary = salary;
361     }
362
363     public String getDeg( ) {
364         return deg;
365     }
366
367     public void setDeg(String deg) {
368         this.deg = deg;
369     }
370
371     @Override
372     public String toString() {
373         return "Employee [eid=" + eid + ", ename=" + ename + ", salary=" + salary + ",
374             deg=" + deg + "]";
375     }
376

```

- In the above code, we have used @Entity annotation to make this POJO class as entity.
- Before going to next module we need to create database for relational entity, which will register the database in persistence.xml file.

5)src/META-INF/persistence.xml

- This module plays a crucial role in the concept of JPA.
- In this xml file we will register the database and specify the entity class.
- In the above shown package hierarchy, persistence.xml under JPA Content package is as follows:

```

385 <?xml version="1.0" encoding="UTF-8"?>
386 <persistence version="2.1"
387     xmlns="http://xmlns.jcp.org/xml/ns/persistence"
388     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
389     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
390         http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
391     <persistence-unit name="Demo" transaction-type="RESOURCE_LOCAL">
392         <class>com.javasoft.Employee</class>
393         <properties>
394             <property name="javax.persistence.jdbc.url"
395                 value="jdbc:h2:tcp://localhost/~/test" />
396             <property name="javax.persistence.jdbc.driver"
397                 value="org.h2.Driver" />
398             <property name="javax.persistence.jdbc.user" value="sa" />
399             <!-- <property name="javax.persistence.jdbc.password" value="" /> -->
400             <property name="eclipselink.logging.level" value="FINE" />
401             <property name="eclipselink.ddl-generation" value="create-tables" />
402         </properties>
403     </persistence-unit>
404 </persistence>

```

- In the above xml, <persistence-unit> tag is defined with specific name for JPA persistence.
- The <class> tag defines entity class with package name.
- The <properties> tag defines all the properties, and <property> tag defines each property such as database registration, URL specification, username, and password.
- These are the EclipseLink properties.
- This file will configure the database.

6)Persistence Operations

- Persistence operations are used against database and they are load and store operations.
- In a business component all the persistence operations fall under service classes.
- In the above shown package hierarchy, create a package named 'com.javasoft.service', under 'src' (source) package.
- All the service classes named as CreateEmployee.java, UpdateEmployee.java,

FindEmployee.java, and DeleteEmployee.java. comes under the given package as follows:

-src/com.javasoft.service package 생성

7)Create Employee

-Creating an Employee class named as CreateEmployee.java as follows:

```
package com.javasoft.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import com.javasoft.Employee;

public class CreateEmployee {
    public static void main( String[ ] args ) {
        EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo" );

        EntityManager entitymanager = emfactory.createEntityManager( );
        entitymanager.getTransaction( ).begin( );

        Employee employee = new Employee( );
        employee.setEid( 1201 );
        employee.setEname( "Gopal" );
        employee.setSalary( 40000 );
        employee.setDeg( "Technical Manager" );

        entitymanager.persist( employee );
        entitymanager.getTransaction( ).commit( );

        entitymanager.close( );
        emfactory.close( );
    }
}
```

-In the above code the createEntityManagerFactory () creates a persistence unit by providing the same unique name which we provide for persistence-unit in persistent.xml file.

-The entitymanagerfactory object will create the entitymanger instance by using createEntityManager () method.

-The entitymanager object creates entitytransaction instance for transaction management.

-By using entitymanager object, we can persist entities into database.

-After compilation and execution of the above program you will get notifications from eclipselink library on the console panel of eclipse IDE.

-For result, open the MySQL workbench and type the following queries.

```
SELECT * FROM employee
```

-The effected database table named employee will be shown in a tabular format as follows:

Eid	Ename	Salary	Deg
1201	Gopal	40000.0	Technical Manager

8)Update Employee

-src/com.javasoft.service/UpdateEmployee.java

```
package com.javasoft.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import com.javasoft.Employee;

public class UpdateEmployee {
    public static void main( String[ ] args ) {
        EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo" );

        EntityManager entitymanager = emfactory.createEntityManager( );
        entitymanager.getTransaction( ).begin( );
```



```

479         Employee employee = entitymanager.find( Employee.class, 1201 );
480
481         //before update
482         System.out.println( employee );
483         employee.setSalary( 46000 );
484         entitymanager.getTransaction( ).commit( );
485
486         //after update
487         System.out.println( employee );
488         entitymanager.close();
489         emfactory.close();
490     }
491 }

```

-The salary of employee, 1201 is updated to 46000.

9)Find Employee

-src/com.javasoft.service/FindEmployee.java

```

498     package com.javasoft.service;
499
500     import javax.persistence.EntityManager;
501     import javax.persistence.EntityManagerFactory;
502     import javax.persistence.Persistence;
503
504     import com.javasoft.Employee;
505
506     public class FindEmployee {
507         public static void main( String[ ] args ) {
508             EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo" );
509             EntityManager entitymanager = emfactory.createEntityManager();
510             Employee employee = entitymanager.find( Employee.class, 1201 );
511
512             System.out.println("employee ID = " + employee.getId( ));
513             System.out.println("employee NAME = " + employee.getName( ));
514             System.out.println("employee SALARY = " + employee.getSalary( ));
515             System.out.println("employee DESIGNATION = " + employee.getDeg( ));
516         }
517     }
518
519     employee ID = 1201
520     employee NAME = Gopal
521     employee SALARY = 46000.0
522     employee DESIGNATION = Technical Manager

```

10)Delete Employee

-src/com.javasoft.service/DeleteEmployee.java

```

526     package com.javasoft.service;
527
528     import javax.persistence.EntityManager;
529     import javax.persistence.EntityManagerFactory;
530     import javax.persistence.Persistence;
531
532     import com.javasoft.Employee;
533
534     public class DeleteEmployee {
535         public static void main( String[ ] args ) {
536             EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo" );
537             EntityManager entitymanager = emfactory.createEntityManager( );
538             entitymanager.getTransaction( ).begin( );
539
540
541             Employee employee = entitymanager.find( Employee.class, 1201 );
542             entitymanager.remove( employee );
543             entitymanager.getTransaction( ).commit( );
544             entitymanager.close( );
545             emfactory.close( );

```

```
546     }  
547 }  
548  
549
```

550 5. JPQL

551 1)Java Persistence Query language

- 552 -JPQL is Java Persistence Query Language defined in JPA specification.
- 553 -It is used to create queries against entities to store in a relational database.
- 554 -JPQL is developed based on SQL syntax.
- 555 -But it won't affect the database directly.
- 556 -JPQL can retrieve information or data using SELECT clause, can do bulk updates using UPDATE clause and DELETE clause.
- 557 -EntityManager.createQuery() API will support for querying language.

558 559 2)Query Structure

- 560 -JPQL syntax is very similar to the syntax of SQL.
- 561 -Having SQL like syntax is an advantage because SQL is a simple structured query language and many developers are using it in applications.
- 562 -SQL works directly against relational database tables, records and fields, whereas JPQL works with Java classes and instances.
- 563 -For example, a JPQL query can retrieve an entity object rather than field result set from database, as with SQL.
- 564 -The JPQL query structure as follows.

```
565  
566     SELECT ... FROM ...  
567     [WHERE ...]  
568     [GROUP BY ... [HAVING ...]]  
569     [ORDER BY ...]  
570
```

- 571 -The structure of JPQL DELETE and UPDATE queries is simpler as follows.

```
572  
573     DELETE FROM ... [WHERE ...]  
574
```

```
575     UPDATE ... SET ... [WHERE ...]  
576
```

577 3)Scalar and Aggregate Functions

- 578 -Scalar functions returns resultant values based on input values.
- 579 -Aggregate functions returns the resultant values by calculating the input values.
- 580 -Follow the same example employee management used in previous chapters.
- 581 -Here we will go through the service classes using scalar and aggregate functions of JPQL.
- 582 -Let us assume the jpadb.employee table contains following records.

583	Eid	Ename	Salary	Deg
584	1201	Gopal	40000	Technical Manager
585	1202	Manisha	40000	Proof Reader
586	1203	Masthanvali	40000	Technical Writer
587	1204	Satish	30000	Technical Writer
588	1205	Krishna	30000	Technical Writer
589	1206	Kiran	35000	Proof Reader

590 591 4)src/com.javasoft.service/ScalarandAggregateFunctions.java

```
592  
593     package com.javasoft.service;  
594  
595     import java.util.List;  
596  
597     import javax.persistence.EntityManager;  
598     import javax.persistence.EntityManagerFactory;  
599     import javax.persistence.Persistence;  
600     import javax.persistence.Query;  
601  
602     public class ScalarandAggregateFunctions {  
603         public static void main( String[ ] args ) {  
604  
605             EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo" );  
606             EntityManager entitymanager = emfactory.createEntityManager();  
607  
608             //Scalar function
```

```

609         Query query = entityManager.createQuery("Select UPPER(e.ename) from Employee e");
610         List<String> list = query.getResultList();
611
612         for(String e:list) {
613             System.out.println("Employee NAME :"+e);
614         }
615
616         //Aggregate function
617         Query query1 = entityManager.createQuery("Select MAX(e.salary) from Employee e");
618         Double result = (Double) query1.getSingleResult();
619         System.out.println("Max Employee Salary : " + result);
620     }
621 }
622
623 Employee NAME :GOPAL
624 Employee NAME :MANISHA
625 Employee NAME :MASTHANVALI
626 Employee NAME :SATISH
627 Employee NAME :KRISHNA
628 Employee NAME :KIRAN
629

```

5)Between, And, Like Keywords

- 'Between', 'And', and 'Like' are the main keywords of JPQL.
- These keywords are used after Where clause in a query.
- src/com.javasoft.service/BetweenAndLikeFunctions.java

```

635     package com.javasoft.service;
636
637     import java.util.List;
638
639     import javax.persistence.EntityManager;
640     import javax.persistence.EntityManagerFactory;
641     import javax.persistence.Persistence;
642     import javax.persistence.Query;
643
644     import com.javasoft.Employee;
645
646     public class BetweenAndLikeFunctions {
647         public static void main( String[ ] args ) {
648
649             EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo" );
650             EntityManager entitymanager = emfactory.createEntityManager();
651
652             //Between
653             Query query = entityManager.createQuery( "Select e " + "from Employee e " + "where
e.salary " + "Between 30000 and 40000" );
654
655             List<Employee> list=(List<Employee>)query.getResultList( );
656
657             for( Employee e:list ){
658                 System.out.print("Employee ID : " + e.getId( ));
659                 System.out.println("\t Employee salary : " + e.getSalary( ));
660             }
661
662             //Like
663             Query query1 = entityManager.createQuery("Select e " + "from Employee e " + "where
e.ename LIKE 'M%'");
664
665             List<Employee> list1=(List<Employee>)query1.getResultList( );
666
667             for( Employee e:list1 ) {
668                 System.out.print("Employee ID :"+e.getId( ));
669                 System.out.println("\t Employee name :"+e.getEname( ));
670             }
671         }
672     }
673

```

```

674 Employee ID :1201 Employee salary :40000.0
675 Employee ID :1202 Employee salary :40000.0
676 Employee ID :1203 Employee salary :40000.0
677 Employee ID :1204 Employee salary :30000.0
678 Employee ID :1205 Employee salary :30000.0
679 Employee ID :1206 Employee salary :35000.0
680 [EL Fine]: sql: 2018-06-21
12:23:05.205--ServerSession(403716510)--Connection(445288316)--Thread(Thread[main,5,mai
n])--SELECT EID, DEG, ENAME, SALARY FROM EMPLOYEE WHERE ENAME LIKE ?

```

```

681 bind => [M%]
682 Employee ID :1202 Employee name :Manisha
683 Employee ID :1203 Employee name :Masthanvali
684

```

6)Ordering

-To Order the records in JPQL we use ORDER BY clause.
 -Ordering.java

```

688 package com.javasoft.service;
689
690 import java.util.List;
691
692 import javax.persistence.EntityManager;
693 import javax.persistence.EntityManagerFactory;
694 import javax.persistence.Persistence;
695 import javax.persistence.Query;
696
697 import com.javasoft.Employee;
698
699 public class Ordering {
700     public static void main( String[ ] args ) {
701         EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo" );
702         EntityManager entitymanager = emfactory.createEntityManager();
703
704         //Between
705         Query query = entitymanager.createQuery( "Select e " + "from Employee e " + "ORDER
706         BY e.ename ASC" );
707
708         List<Employee> list = (List<Employee>)query.getResultList( );
709
710         for( Employee e:list ) {
711             System.out.print("Employee ID :" + e.getId( ));
712             System.out.println("\t Employee Name :" + e.getEname( ));
713         }
714     }
715 }
716

```

```

717 Employee ID :1201 Employee Name :Gopal
718 Employee ID :1206 Employee Name :Kiran
719 Employee ID :1205 Employee Name :Krishna
720 Employee ID :1202 Employee Name :Manisha
721 Employee ID :1203 Employee Name :Masthanvali
722 Employee ID :1204 Employee Name :Satish
723

```

7)Named Queries

-A @NamedQuery annotation is defined as a query with a predefined unchangeable query string.
 -Instead of dynamic queries, usage of named queries may improve code organization by separating the JPQL query strings from POJO.
 -It also passes the query parameters rather than embedding literals dynamically into the query string and results in more efficient queries.
 -First of all, add @NamedQuery annotation to the Employee entity class named Employee.java.

```

728 @Entity
729 @Table
730 @NamedQuery(query = "Select e from Employee e where e.eid = :id", name = "find
731 employee by id")
732 public class Employee {
733
734

```

-src/com.javasoft.service/NamedQueries.java

```
package com.javasoft.service;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

import com.javasoft.Employee;

public class NamedQueries {
    public static void main( String[ ] args ) {

        EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo" );
        EntityManager entitymanager = emfactory.createEntityManager();
        Query query = entitymanager.createNamedQuery("find employee by id");

        query.setParameter("id", 1204);
        List<Employee> list = query.getResultList( );

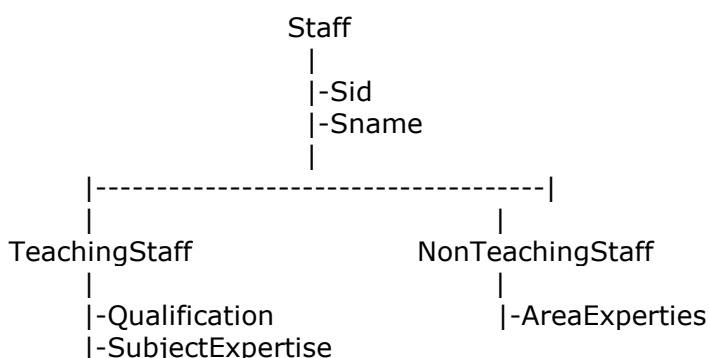
        for( Employee e:list ){
            System.out.print("Employee ID : " + e.getId( ));
            System.out.println("\t Employee Name : " + e.getName( ));
        }
    }
}
```

Employee ID :1204 Employee Name :Satish

6. Advanced Mappings

1)Inheritance Strategies

- Inheritance is the core concept of object oriented language, therefore we can use inheritance relationships or strategies between entities.
- JPA support three types of inheritance strategies such as SINGLE_TABLE, JOINED_TABLE, and TABLE_PER_CONCRETE_CLASS.
- Let us consider an example of Staff, TeachingStaff, NonTeachingStaff classes and their relationships as follows:



2)Single Table strategy

- Single-Table strategy takes all classes fields (both super and sub classes) and map them down into a single table known as SINGLE_TABLE strategy.
- Here discriminator value plays key role in differentiating the values of three entities in one table.
- Let us consider the above example, TeachingStaff and NonTeachingStaff are the sub classes of class Staff.
- Remind the concept of inheritance (is a mechanism of inheriting the properties of super class by sub class) and therefore sid, sname are the fields which belongs to both TeachingStaff and NonTeachingStaff.
- Create a JPA project.
- All the modules of this project as follows:
- Creating Entities

```

795 --src/com.javasoft.entity package 생성
796 --src/com.javasoft.entity.Staff.java
797
798 package com.javasoft.entity;
799
800 import java.io.Serializable;
801
802 import javax.persistence.DiscriminatorColumn;
803 import javax.persistence.Entity;
804 import javax.persistence.GeneratedValue;
805 import javax.persistence.GenerationType;
806 import javax.persistence.Id;
807 import javax.persistence.Inheritance;
808 import javax.persistence.InheritanceType;
809 import javax.persistence.Table;
810
811 @Entity
812 @Table
813 @Inheritance( strategy = InheritanceType.SINGLE_TABLE )
814 @DiscriminatorColumn( name = "type" )
815
816 public class Staff implements Serializable {
817     @Id
818     @GeneratedValue( strategy = GenerationType.AUTO )
819
820     private int sid;
821     private String sname;
822
823     public Staff( int sid, String sname ) {
824         this.sid = sid;
825         this.sname = sname;
826     }
827
828     public Staff( ) {
829         super( );
830     }
831
832     public int getSid( ) {
833         return sid;
834     }
835
836     public void setSid( int sid ) {
837         this.sid = sid;
838     }
839
840     public String getSname( ) {
841         return sname;
842     }
843
844     public void setSname( String sname ) {
845         this.sname = sname;
846     }
847 }

```

-In the above code @DiscriminatorColumn specifies the field name (type) and the values of it shows the remaining (Teaching and NonTeachingStaff) fields.

-Create a subclass (class) to Staff class named TeachingStaff.java under the com.javasoft.entity package.

-The TeachingStaff Entity class is shown as follows:

```

854 package com.javasoft.entity;
855
856 import javax.persistence.DiscriminatorValue;
857 import javax.persistence.Entity;
858
859 @Entity

```

```

860 @DiscriminatorValue( value="TS" )
861 public class TeachingStaff extends Staff {
862     private String qualification;
863     private String subjectexpertise;
864
865     public TeachingStaff( int sid, String sname, String qualification,String subjectexpertise ) {
866         super( sid, sname );
867         this.qualification = qualification;
868         this.subjectexpertise = subjectexpertise;
869     }
870
871     public TeachingStaff( ) {
872         super( );
873     }
874
875     public String getQualification( ){
876         return qualification;
877     }
878
879     public void setQualification( String qualification ){
880         this.qualification = qualification;
881     }
882
883     public String getSubjectexpertise( ) {
884         return subjectexpertise;
885     }
886
887     public void setSubjectexpertise( String subjectexpertise ){
888         this.subjectexpertise = subjectexpertise;
889     }
890 }

```

-Create a subclass (class) to Staff class named NonTeachingStaff.java under the com.javasoft.entity package.

-The NonTeachingStaff Entity class is shown as follows:

```

893 package com.javasoft.entity;
894
895 import javax.persistence.DiscriminatorValue;
896 import javax.persistence.Entity;
897
898 @Entity
899 @DiscriminatorValue( value = "NS" )
900 public class NonTeachingStaff extends Staff{
901     private String areaexpertise;
902
903     public NonTeachingStaff( int sid, String sname, String areaexpertise ) {
904         super( sid, sname );
905         this.areaexpertise = areaexpertise;
906     }
907
908     public NonTeachingStaff( ) {
909         super( );
910     }
911
912     public String getAreaexpertise( ) {
913         return areaexpertise;
914     }
915
916     public void setAreaexpertise( String areaexpertise ){
917         this.areaexpertise = areaexpertise;
918     }
919 }
920
921 }
922

```

-META-INF/persistence.xml

-Persistence.xml file contains the configuration information of database and registration information of entity classes.

-The xml file is shown as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="Demo"
    transaction-type="RESOURCE_LOCAL">
    <class>com.javasoft.entity.Staff</class>
    <class>com.javasoft.entity.NonTeachingStaff</class>
    <class>com.javasoft.entity.TeachingStaff</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:h2:tcp://localhost/~/test" />
      <property name="javax.persistence.jdbc.driver"
        value="org.h2.Driver" />
      <property name="javax.persistence.jdbc.user" value="sa" />
      <!-- <property name="javax.persistence.jdbc.password" value=""/> -->
      <property name="eclipselink.logging.level" value="FINE" />
      <property name="eclipselink.ddl-generation" value="create-tables" />
    </properties>
  </persistence-unit>
</persistence>
```

-Service class

```
--Service classes are the implementation part of business component.
--Create a package under 'src' package named 'com.javasoft.service'.
--Create a class named SaveClient.java under the given package to store Staff,
TeachingStaff, and NonTeachingStaff class fields.
--The SaveClient class is shown as follows:
```

```
package com.javasoft.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import com.javasoft.entity.NonTeachingStaff;
import com.javasoft.entity.TeachingStaff;

public class SaveClient {
    public static void main( String[ ] args ) {

        EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo" );
        EntityManager entitymanager = emfactory.createEntityManager( );
        entitymanager.getTransaction( ).begin( );

        //Teaching staff entity
        TeachingStaff ts1=new TeachingStaff(1,"Gopal","MSc MEd","Maths");
        TeachingStaff ts2=new TeachingStaff(2, "Manisha", "BSc BEd", "English");

        //Non-Teaching Staff entity
        NonTeachingStaff nts1=new NonTeachingStaff(3, "Satish", "Accounts");
        NonTeachingStaff nts2=new NonTeachingStaff(4, "Krishna", "Office Admin");

        //storing all entities
        entitymanager.persist(ts1);
        entitymanager.persist(ts2);
        entitymanager.persist(nts1);
        entitymanager.persist(nts2);

        entitymanager.getTransaction().commit();

        entitymanager.close();
        emfactory.close();
    }
}
```



```

990     }
991 }
992

```

993 --After compilation and execution of the above program you will get notifications in the console panel of Eclipse IDE.

994 --The output in a tabular format is shown as follows:

```

995
996     Sid  Type   Sname   Areaexpertise  Qualification  Subjectexpertise
997     1    TS    Gopal   null           MSC MED       Maths
998     2    TS    Manisha null           BSC BED       English
999     3    NS    Satish  Accounts       null          null
1000     4    NS    Krishna Office Admin   null          null
1001

```

1002 --Finally you will get single table which contains all three class's fields and differs with discriminator column named 'Type' (field).

1003 3)Joined table Strategy

1004 -Joined table strategy is to share the referenced column which contains unique values to join the table and make easy transactions.

1005 -Let us consider the same example as above.

1006 -Create a JPA Project.

1007 -All the project modules shown as follows:

1008 -Creating Entities

1009 --Create a package named 'com.javasoft.entity' under 'src' package.

1010 --Create a new java class named Staff.java under given package.

1011 --The Staff entity class is shown as follows:

```

1012
1013     package com.javasoft.entity;
1014
1015     import java.io.Serializable;
1016
1017     import javax.persistence.Entity;
1018     import javax.persistence.GeneratedValue;
1019     import javax.persistence.GenerationType;
1020     import javax.persistence.Id;
1021     import javax.persistence.Inheritance;
1022     import javax.persistence.InheritanceType;
1023     import javax.persistence.Table;
1024
1025     @Entity
1026     @Table
1027     @Inheritance( strategy = InheritanceType.JOINED )
1028     public class Staff implements Serializable {
1029         @Id
1030         @GeneratedValue( strategy = GenerationType.AUTO )
1031
1032         private int sid;
1033         private String sname;
1034
1035         public Staff( int sid, String sname ) {
1036             super( );
1037             this.sid = sid;
1038             this.sname = sname;
1039         }
1040
1041         public Staff( ) {
1042             super( );
1043         }
1044
1045         public int getSid( ) {
1046             return sid;
1047         }
1048
1049         public void setSid( int sid ) {
1050             this.sid = sid;
1051         }
1052     }
1053

```

```

1054         public String getSname( ) {
1055             return sname;
1056         }
1057
1058         public void setSname( String sname ) {
1059             this.sname = sname;
1060         }
1061     }
1062

```

--Create a subclass (class) to Staff class named TeachingStaff.java under the com.javasoft.entity package.

--The TeachingStaff Entity class is shown as follows:

```

1064         package com.javasoft.entity;
1065
1066         import javax.persistence.Entity;
1067         import javax.persistence.PrimaryKeyJoinColumn;
1068
1069         @Entity
1070         @PrimaryKeyJoinColumn(referencedColumnName="sid")
1071         public class TeachingStaff extends Staff{
1072             private String qualification;
1073             private String subjectexpertise;
1074
1075             public TeachingStaff( int sid, String sname, String qualification,String subjectexpertise
1076             ) {
1077                 super( sid, sname );
1078                 this.qualification = qualification;
1079                 this.subjectexpertise = subjectexpertise;
1080             }
1081
1082             public TeachingStaff( ) {
1083                 super( );
1084             }
1085
1086             public String getQualification( ){
1087                 return qualification;
1088             }
1089
1090             public void setQualification( String qualification ){
1091                 this.qualification = qualification;
1092             }
1093
1094             public String getSubjectexpertise( ) {
1095                 return subjectexpertise;
1096             }
1097
1098             public void setSubjectexpertise( String subjectexpertise ){
1099                 this.subjectexpertise = subjectexpertise;
1100             }
1101         }
1102

```

--Create a subclass (class) to Staff class named NonTeachingStaff.java under the com.javasoft.entity package.

--The NonTeachingStaff Entity class is shown as follows:

```

1104         package com.javasoft.entity;
1105
1106         import javax.persistence.Entity;
1107         import javax.persistence.PrimaryKeyJoinColumn;
1108
1109         @Entity
1110         @PrimaryKeyJoinColumn(referencedColumnName="sid")
1111         public class NonTeachingStaff extends Staff {
1112             private String areaexpertise;
1113
1114             public NonTeachingStaff( int sid, String sname, String areaexpertise ) {

```

```

1118         super( sid, sname );
1119         this.areaexpertise = areaexpertise;
1120     }
1121
1122     public NonTeachingStaff( ) {
1123         super( );
1124     }
1125
1126     public String getAreaexpertise( ) {
1127         return areaexpertise;
1128     }
1129
1130     public void setAreaexpertise( String areaexpertise ) {
1131         this.areaexpertise = areaexpertise;
1132     }
1133 }
1134 -META-INF/persistence.xml
1135
1136 <?xml version="1.0" encoding="UTF-8"?>
1137 <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
1138     <persistence-unit name="Demo1" transaction-type="RESOURCE_LOCAL">
1139         <class>com.javasoft.entity.Staff</class>
1140         <class>com.javasoft.entity.NonTeachingStaff</class>
1141         <class>com.javasoft.entity.TeachingStaff</class>
1142         <properties>
1143             <property name="javax.persistence.jdbc.url"
1144                 value="jdbc:h2:tcp://localhost/~/test" />
1145             <property name="javax.persistence.jdbc.driver"
1146                 value="org.h2.Driver" />
1147             <property name="javax.persistence.jdbc.user" value="sa" />
1148             <!-- <property name="javax.persistence.jdbc.password" value=""/> -->
1149             <property name="eclipselink.logging.level" value="FINE" />
1150             <property name="eclipselink.ddl-generation" value="create-tables" />
1151         </properties>
1152     </persistence-unit>
1153 </persistence>
1154
1155 -Service class
1156 --Service classes are the implementation part of business component.
1157 --Create a package under 'src' package named 'com.javasoft.service'.
1158 --Create a class named SaveClient.java under the given package to store Staff,
TeachingStaff, and NonTeachingStaff class fields.
1159 --Then SaveClient class as follows:
1160
1161     package com.javasoft.service;
1162
1163     import javax.persistence.EntityManager;
1164     import javax.persistence.EntityManagerFactory;
1165     import javax.persistence.Persistence;
1166
1167     import com.javasoft.entity.NonTeachingStaff;
1168     import com.javasoft.entity.TeachingStaff;
1169
1170     public class SaveClient {
1171         public static void main( String[ ] args ) {
1172             EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Demo1"
);
1173             EntityManager entitymanager = emfactory.createEntityManager( );
1174             entitymanager.getTransaction( ).begin( );
1175
1176             //Teaching staff entity
1177             TeachingStaff ts1 = new TeachingStaff(1,"Gopal","MSc MEd","Maths");
1178             TeachingStaff ts2 = new TeachingStaff(2, "Manisha", "BSc BEd", "English");
1179

```

```

1180 //Non-Teaching Staff entity
1181 NonTeachingStaff nts1 = new NonTeachingStaff(3, "Satish", "Accounts");
1182 NonTeachingStaff nts2 = new NonTeachingStaff(4, "Krishna", "Office Admin");
1183
1184 //storing all entities
1185 entitymanager.persist(ts1);
1186 entitymanager.persist(ts2);
1187 entitymanager.persist(nts1);
1188 entitymanager.persist(nts2);
1189
1190 entitymanager.getTransaction().commit();
1191 entitymanager.close();
1192 emfactory.close();
1193 }
1194 }
1195
1196

```

-After compilation and execution of the above program you will get notifications in the console panel of Eclipse IDE.

-Here three tables are created and the result of staff table in a tabular format is shown as follows:

Sid	Dtype	Sname
1	TeachingStaff	Gopal
2	TeachingStaff	Manisha
3	NonTeachingStaff	Satish
4	NonTeachingStaff	Krishna

-The result of TeachingStaff table in a tabular format is shown as follows:

Sid	Qualification	Subjectexpertise
1	MSC MED	Maths
2	BSC BED	English

-In the above table sid is the foreign key (reference field form staff table).

-The result of NonTeachingStaff table in tabular format is shown as follows:

Sid	Areaexpertise
3	Accounts
4	Office Admin

-Finally the three tables are created using their fields respectively and SID field is shared by all three tables.

-In staff table SID is primary key, in remaining (TeachingStaff and NonTeachingStaff) tables SID is foreign key.

4)Table per class strategy

-Table per class strategy is to create a table for each sub entity.

-The staff table will be created but it will contain null records.

-The field values of Staff table must be shared by TeachingStaff and NonTeachingStaff tables.

-Let us consider the same example as above.

-All modules of this project are shown as follows:

-Creating Entities

--Create a package named 'com.javasoft.entity' under 'src' package.

--Create a new java class named Staff.java under given package.

--The Staff entity class is shown as follows:

```

package com.javasoft.entity;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

```

```
1244 @Entity
1245 @Table
1246 @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
```

```
1247
1248 public class Staff implements Serializable {
1249
1250     @Id
1251     @GeneratedValue(strategy = GenerationType.AUTO)
1252
1253     private int sid;
1254     private String sname;
1255
1256     public Staff(int sid, String sname) {
1257         this.sid = sid;
1258         this.sname = sname;
1259     }
1260
1261     public Staff() {
1262         super();
1263     }
1264
1265     public int getSid() {
1266         return sid;
1267     }
1268
1269     public void setSid(int sid) {
1270         this.sid = sid;
1271     }
1272
1273     public String getSname() {
1274         return sname;
1275     }
1276
1277     public void setSname(String sname) {
1278         this.sname = sname;
1279     }
1280 }
1281
```

```
1282 --Create a subclass (class) to Staff class named TeachingStaff.java under the
com.javasoft.entity package.
```

```
1283 --The TeachingStaff Entity class is shown as follows:
```

```
1284
1285 package com.javasoft.entity;
1286
1287 import javax.persistence.Entity;
1288
1289 @Entity
1290 public class TeachingStaff extends Staff {
1291     private String qualification;
1292     private String subjectexpertise;
1293
1294     public TeachingStaff( int sid, String sname, String qualification, String subjectexpertise ) {
1295         super( sid, sname );
1296         this.qualification = qualification;
1297         this.subjectexpertise = subjectexpertise;
1298     }
1299
1300     public TeachingStaff( ) {
1301         super( );
1302     }
1303
1304     public String getQualification( ){
1305         return qualification;
1306     }
1307
1308     public void setQualification( String qualification ) {
1309         this.qualification = qualification;
1310     }
1311 }
```

```

1310     }
1311
1312     public String getSubjectexpertise( ) {
1313         return subjectexpertise;
1314     }
1315
1316     public void setSubjectexpertise( String subjectexpertise ){
1317         this.subjectexpertise = subjectexpertise;
1318     }
1319 }
1320
1321 --Create a subclass (class) to Staff class named NonTeachingStaff.java under the
com.javasoft.entity package.

```

```

1322 --The NonTeachingStaff Entity class is shown as follows:

```

```

1323
1324     package com.javasoft.entity;
1325
1326     import javax.persistence.Entity;
1327
1328     @Entity
1329     public class NonTeachingStaff extends Staff {
1330         private String areaexpertise;
1331
1332         public NonTeachingStaff( int sid, String sname, String areaexpertise ) {
1333             super( sid, sname );
1334             this.areaexpertise = areaexpertise;
1335         }
1336
1337         public NonTeachingStaff( ) {
1338             super( );
1339         }
1340
1341         public String getAreaexpertise( ) {
1342             return areaexpertise;
1343         }
1344
1345         public void setAreaexpertise( String areaexpertise ) {
1346             this.areaexpertise = areaexpertise;
1347         }
1348     }
1349

```

```

1350 -META-INF/persistence.xml

```

```

1351
1352     <?xml version="1.0" encoding="UTF-8"?>
1353     <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
1354         <persistence-unit name="Demo1" transaction-type="RESOURCE_LOCAL">
1355             <class>com.javasoft.entity.Staff</class>
1356             <class>com.javasoft.entity.NonTeachingStaff</class>
1357             <class>com.javasoft.entity.TeachingStaff</class>
1358             <properties>
1359                 <property name="javax.persistence.jdbc.url"
1360                     value="jdbc:h2:tcp://localhost/~/test" />
1361                 <property name="javax.persistence.jdbc.driver"
1362                     value="org.h2.Driver" />
1363                 <property name="javax.persistence.jdbc.user" value="sa" />
1364                 <!-- <property name="javax.persistence.jdbc.password" value=""/> -->
1365                 <property name="eclipselink.logging.level" value="FINE" />
1366                 <property name="eclipselink.ddl-generation" value="create-tables" />
1367             </properties>
1368         </persistence-unit>
1369     </persistence>
1370

```

```

1371 -Service class

```

```

1372     --Service classes are the implementation part of business component.

```

--Create a package under 'src' package named 'com.javasoft.service'.
--Create a class named SaveClient.java under the given package to store Staff, TeachingStaff, and NonTeachingStaff class fields.
--The SaveClient class is shown as follows:

```
package com.javasoft.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import com.javasoft.entity.NonTeachingStaff;
import com.javasoft.entity.TeachingStaff;

public class SaveClient {
    public static void main( String[ ] args ) {
        EntityManagerFactory emfactory = Persistence.createEntityManagerFactory(
            "Eclipselink_JPA" );
        EntityManager entitymanager = emfactory.createEntityManager( );
        entitymanager.getTransaction( ).begin( );

        //Teaching staff entity
        TeachingStaff ts1 = new TeachingStaff(1,"Gopal","MSc MEd","Maths");
        TeachingStaff ts2 = new TeachingStaff(2, "Manisha", "BSc BEd", "English");

        //Non-Teaching Staff entity
        NonTeachingStaff nts1 = new NonTeachingStaff(3, "Satish", "Accounts");
        NonTeachingStaff nts2 = new NonTeachingStaff(4, "Krishna", "Office Admin");

        //storing all entities
        entitymanager.persist(ts1);
        entitymanager.persist(ts2);
        entitymanager.persist(nts1);
        entitymanager.persist(nts2);

        entitymanager.getTransaction().commit();
        entitymanager.close();
        emfactory.close();
    }
}
```

--After compilation and execution of the above program you will get notifications in the console panel of Eclipse IDE.

--Here the three tables are created and the Staff table contains null records.

--The result of TeachingStaff in a tabular format is shown as follows:

Sid	Qualification	Sname	Subjectexpertise
1	MSC MED	Gopal	Maths
2	BSC BED	Manisha	English

--The above table TeachingStaff contains fields of both Staff and TeachingStaff Entities.

--The result of NonTeachingStaff in a tabular format is shown as follows:

Sid	Areaexpertise	Sname
3	Accounts	Satish
4	Office Admin	Krishna

--The above table NonTeachingStaff contains fields of both Staff and NonTeachingStaff Entities.

7. Entity Relationships

1)This chapter takes you through the relationships between Entities.

2)Generally the relations are more effective between tables in the database.

3)Here the entity classes are treated as relational tables (concept of JPA), therefore the relationships between Entity classes are as follows:

@ManyToOne Relation

@OneToMany Relation
@OneToOne Relation
@ManyToMany Relation

4)@ManyToOne Relation

- Where one entity (column or set of columns) is/are referenced with another entity (column or set of columns) which contain unique values.
- In relational databases these relations are applicable by using foreign key/primary key between tables.
- Let us consider an example of relation between Employee and Department entities.
- In unidirectional manner, i.e.from Employee to Department, Many-To-One relation is applicable.
- That means each record of employee contains one department id, which should be a primary key in Department table.
- Here in the Employee table, Department id is foreign Key.
- Create a JPA project in eclipse IDE named JPA_MTO.
- All the modules of this project are shown as follows:

-Creating Entities

- Create a package named 'com.javasoft.entity' under 'src' package.
- Create a class named Department.java under given package.
- The class Department entity is shown as follows:

```
package com.javasoft.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)

    private int id;
    private String name;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

- Create the second entity in this relation - Employee entity class named Employee.java under 'com.javasoft.entity' package.
- The Employee entity class is shown as follows:

```
package com.javasoft.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

@Entity
```



```

1499 public class Employee {
1500     @Id
1501     @GeneratedValue(strategy = GenerationType.AUTO)
1502
1503     private int eid;
1504     private String ename;
1505     private double salary;
1506     private String deg;
1507
1508     @ManyToOne
1509     private Department department;
1510
1511     public Employee(int eid, String ename, double salary, String deg) {
1512         super();
1513         this.eid = eid;
1514         this.ename = ename;
1515         this.salary = salary;
1516         this.deg = deg;
1517     }
1518
1519     public Employee() {
1520         super();
1521     }
1522
1523     public int getEid() {
1524         return eid;
1525     }
1526
1527     public void setEid(int eid) {
1528         this.eid = eid;
1529     }
1530
1531     public String getEname() {
1532         return ename;
1533     }
1534
1535     public void setEname(String ename) {
1536         this.ename = ename;
1537     }
1538
1539     public double getSalary() {
1540         return salary;
1541     }
1542
1543     public void setSalary(double salary) {
1544         this.salary = salary;
1545     }
1546
1547     public String getDeg() {
1548         return deg;
1549     }
1550
1551     public void setDeg(String deg) {
1552         this.deg = deg;
1553     }
1554
1555     public Department getDepartment() {
1556         return department;
1557     }
1558
1559     public void setDepartment(Department department) {
1560         this.department = department;
1561     }
1562 }
1563

```

-Persistence.xml

--Persistence.xml file is required to configure the database and the registration of entity

classes.

--Persistence.xml will be created by the eclipse IDE while creating a JPA Project.

--The configuration details are user specifications.

--The persistence.xml file is shown as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name = "JPA_MTO" transaction-type = "RESOURCE_LOCAL">
    <class>com.javasoft.entity.Employee</class>
    <class>com.javasoft.entity.Department</class>

    <properties>
      <property name = "javax.persistence.jdbc.url" value = "jdbc:h2:
tcp://localhost/~/test"/>
      <property name = "javax.persistence.jdbc.user" value = "sa"/>
      <property name = "javax.persistence.jdbc.driver" value="org.h2.Driver"/>
      <property name = "eclipselink.logging.level" value = "FINE"/>
      <property name = "eclipselink.ddl-generation" value = "create-tables"/>
    </properties>
  </persistence-unit>
</persistence>
```

-Service Classes

--This module contains the service classes, which implements the relational part using the attribute initialization.

--Create a package under 'src' package named 'com.javasoft.service'.

--The DAO class named ManyToOne.java is created under given package.

--The DAO class is shown as follows:

```
package com.javasoft.service;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import com.javasoft.entity.Department;
import com.javasoft.entity.Employee;

public class ManyToOne {
    public static void main(String[] args) {

        EntityManagerFactory emfactory =
            Persistence.createEntityManagerFactory("JPA_MTO");
        EntityManager entitymanager = emfactory.createEntityManager();
        entitymanager.getTransaction().begin();

        // Create Department Entity
        Department department = new Department();
        department.setName("Development");

        // Store Department
        entitymanager.persist(department);

        // Create Employee1 Entity
        Employee employee1 = new Employee();
        employee1.setEname("Satish");
        employee1.setSalary(45000.0);
        employee1.setDeg("Technical Writer");
        employee1.setDepartment(department);

        // Create Employee2 Entity
        Employee employee2 = new Employee();
        employee2.setEname("Krishna");
```

```

1626     employee2.setSalary(45000.0);
1627     employee2.setDeg("Technical Writer");
1628     employee2.setDepartment(department);
1629
1630     // Create Employee3 Entity
1631     Employee employee3 = new Employee();
1632     employee3.setEname("Masthanali");
1633     employee3.setSalary(50000.0);
1634     employee3.setDeg("Technical Writer");
1635     employee3.setDepartment(department);
1636
1637     // Store Employees
1638     entitymanager.persist(employee1);
1639     entitymanager.persist(employee2);
1640     entitymanager.persist(employee3);
1641
1642     entitymanager.getTransaction().commit();
1643     entitymanager.close();
1644     emfactory.close();
1645 }
1646 }
1647

```

-After compilation and execution of the above program you will get notifications in the console panel of Eclipse IDE.

-In this example two tables are created.

-Pass the following query in H2Database interface and the result of Department table in a tabular format is shown as follows in the query:

```
Select * from department;
```

Id	Name
1	Development

-Pass the following query in H2Database interface and the result of Employee table in a tabular format is shown as follows in the query:

```
Select * from employee;
```

Eid	Deg	Ename	Salary	Department_Id
2	Technical Writer	Satish	45000	1
3	Technical Writer	Krishna	45000	1
4	Technical Writer	Masthan Wali	50000	1

-In the above table Department_Id is the foreign key (reference field) from Department table.

5)@OneToMany Relation

-In this relationship each row of one entity is referenced to many child records in other entity.

-The important thing is that child records cannot have multiple parents.

-In a one-to-many relationship between Table A and Table B, each row in Table A is linked to 0, 1 or many rows in Table B.

-Let us consider the above example.

-If Employee and Department is in a reverse unidirectional manner, relation is Many-To-One relation.

-Create a JPA project in eclipse IDE named JPA_OTM.

-All the modules of this project are shown as follows:

-Creating Entities

--Create a package named 'com.javasoft.entity' under 'src' package.

--Create a class named Department.java under given package.

--The class Department entity is shown as follows:

```

package com.javasoft.entity;

import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;

```

```

1688 import javax.persistence.GenerationType;
1689 import javax.persistence.Id;
1690 import javax.persistence.OneToMany;
1691
1692 @Entity
1693 public class Department {
1694     @Id
1695     @GeneratedValue(strategy = GenerationType.AUTO)
1696
1697     private int id;
1698     private String name;
1699
1700     @OneToMany(targetEntity = Employee.class)
1701     private List employeeelist;
1702
1703     public int getId() {
1704         return id;
1705     }
1706
1707     public void setId(int id) {
1708         this.id = id;
1709     }
1710
1711     public String getName() {
1712         return name;
1713     }
1714
1715     public void setName(String deptName) {
1716         this.name = deptName;
1717     }
1718
1719     public List getEmployeeelist() {
1720         return employeeelist;
1721     }
1722
1723     public void setEmployeeelist(List employeeelist) {
1724         this.employeeelist = employeeelist;
1725     }
1726 }

```

--Create the second entity in this relation - Employee entity class named Employee.java under 'com.javasoft.entity' package.

--The Employee entity class is shown as follows:

```

1731 package com.javasoft.entity;
1732
1733 import javax.persistence.Entity;
1734 import javax.persistence.GeneratedValue;
1735 import javax.persistence.GenerationType;
1736 import javax.persistence.Id;
1737 import javax.persistence.ManyToOne;
1738
1739 @Entity
1740 public class Employee {
1741     @Id
1742     @GeneratedValue(strategy = GenerationType.AUTO)
1743
1744     private int eid;
1745     private String ename;
1746     private double salary;
1747     private String deg;
1748
1749     public Employee(int eid, String ename, double salary, String deg) {
1750         super();
1751         this.eid = eid;
1752         this.ename = ename;
1753         this.salary = salary;

```

```

1754         this.deg = deg;
1755     }
1756
1757     public Employee() {
1758         super();
1759     }
1760
1761     public int getEid() {
1762         return eid;
1763     }
1764
1765     public void setEid(int eid) {
1766         this.eid = eid;
1767     }
1768
1769     public String getEname() {
1770         return ename;
1771     }
1772
1773     public void setEname(String ename) {
1774         this.ename = ename;
1775     }
1776
1777     public double getSalary() {
1778         return salary;
1779     }
1780
1781     public void setSalary(double salary) {
1782         this.salary = salary;
1783     }
1784
1785     public String getDeg() {
1786         return deg;
1787     }
1788
1789     public void setDeg(String deg) {
1790         this.deg = deg;
1791     }
1792 }
1793

```

-Persistence.xml

- Persistence.xml file is required to configure the database and the registration of entity classes.
- Persistence.xml will be created by the eclipse IDE while creating a JPA Project.
- The configuration details are user specifications.
- The persistence.xml file is shown as follows:

```

1800 <?xml version="1.0" encoding="UTF-8"?>
1801 <persistence version="2.1"
1802     xmlns="http://xmlns.jcp.org/xml/ns/persistence"
1803     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1804     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
1805         http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
1806     <persistence-unit name="JPA_OTM" transaction-type="RESOURCE_LOCAL">
1807         <class>com.javasoft.entity.Employee</class>
1808         <class>com.javasoft.entity.Department</class>
1809
1810         <properties>
1811             <property name="javax.persistence.jdbc.url" value="jdbc:h2:
1812                 tcp://localhost/~/test" />
1813             <property name="javax.persistence.jdbc.user" value="sa" />
1814             <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
1815             <property name="eclipselink.logging.level" value="FINE" />
1816             <property name="eclipselink.ddl-generation" value="create-tables" />
1817         </properties>
1818     </persistence-unit>

```

</persistence>

-Service Classes

- This module contains the service classes, which implements the relational part using the attribute initialization.
- Create a package under 'src' package named 'com.javasoft.service'.
- The DAO class named OneToMany.java is created under given package.
- The DAO class is shown as follows:

```
package com.javasoft.service;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import com.javasoft.entity.Department;
import com.javasoft.entity.Employee;

public class OneToMany {
    public static void main(String[] args) {

        EntityManagerFactory emfactory =
            Persistence.createEntityManagerFactory("JPA_OTM");
        EntityManager entitymanager = emfactory.createEntityManager();
        entitymanager.getTransaction().begin();

        // Create Employee1 Entity
        Employee employee1 = new Employee();
        employee1.setEname("Satish");
        employee1.setSalary(45000.0);
        employee1.setDeg("Technical Writer");

        // Create Employee2 Entity
        Employee employee2 = new Employee();
        employee2.setEname("Krishna");
        employee2.setSalary(45000.0);
        employee2.setDeg("Technical Writer");

        // Create Employee3 Entity
        Employee employee3 = new Employee();
        employee3.setEname("Masthanvali");
        employee3.setSalary(50000.0);
        employee3.setDeg("Technical Writer");

        // Store Employee
        entitymanager.persist(employee1);
        entitymanager.persist(employee2);
        entitymanager.persist(employee3);

        // Create Employeeelist
        List<Employee> emplist = new ArrayList();
        emplist.add(employee1);
        emplist.add(employee2);
        emplist.add(employee3);

        // Create Department Entity
        Department department = new Department();
        department.setName("Development");
        department.setEmployeeelist(emplist);

        // Store Department
        entitymanager.persist(department);

        entitymanager.getTransaction().commit();
```

```

1883         entitymanager.close();
1884         emfactory.close();
1885     }
1886 }
1887
1888
1889

```

-After compilation and execution of the above program you will get notifications in the console panel of Eclipse IDE.

-In this example two tables are created.

-Pass the following query in H2Database interface and the result of Department_employee table in a tabular format is shown as follows in the query:

```
SELECT * FROM department_employee;
```

Department_Id	Employee_Eid
54	51
54	52
54	53

-In the above table, department_id and employee_id fields are the foreign keys (reference fields) from department and employee tables.

-Pass the following query in H2Database interface and the result of department table in a tabular format is shown as follows in the query:

```
Select * from department;
```

Id	Name
54	Development

-Pass the following query in H2Database interface and the result of employee table in a tabular format is shown as follows in the query:

```
Select * from employee;
```

Eid	Deg	Ename	Salary
51	Technical Writer	Satish	45000
52	Technical Writer	Krishna	45000
53	Technical Writer	Masthanvali	50000

6)OneToOne Relation

-In One-To-One relationship, one item can belong to only one other item.

-It means each row of one entity is referred to one and only one row of another entity.

-Let us consider the above example.

-Employee and Department in a reverse unidirectional manner, the relation is One-To-One relation.

-It means each employee belongs to only one department.

-Create a JPA project in eclipse IDE named JPA_OT0.

-All the modules of this project are shown as follows:

-Creating Entities

--Create a package named 'com.javasoft.entity' under 'src' package.

--Create a class named Department.java under given package.

--The class Department entity is shown as follows:

```

package com.javasoft.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String name;
}

```

```

1944         public int getId() {
1945             return id;
1946         }
1947
1948         public void setId(int id) {
1949             this.id = id;
1950         }
1951
1952         public String getName() {
1953             return name;
1954         }
1955
1956         public void setName(String deptName) {
1957             this.name = deptName;
1958         }
1959     }
1960 }

```

--Create the second entity in this relation - Employee entity class named Employee.java under 'com.javasoft.entity' package.

--The Employee entity class is shown as follows:

```

1964     package com.javasoft.entity;
1965
1966     import javax.persistence.Entity;
1967     import javax.persistence.GeneratedValue;
1968     import javax.persistence.GenerationType;
1969     import javax.persistence.Id;
1970     import javax.persistence.OneToOne;
1971
1972     @Entity
1973     public class Employee {
1974         @Id
1975         @GeneratedValue(strategy = GenerationType.AUTO)
1976         private int eid;
1977         private String ename;
1978         private double salary;
1979         private String deg;
1980
1981         @OneToOne
1982         private Department department;
1983
1984         public Employee(int eid, String ename, double salary, String deg) {
1985             super();
1986             this.eid = eid;
1987             this.ename = ename;
1988             this.salary = salary;
1989             this.deg = deg;
1990         }
1991
1992         public Employee() {
1993             super();
1994         }
1995
1996         public int getEid() {
1997             return eid;
1998         }
1999
2000         public void setEid(int eid) {
2001             this.eid = eid;
2002         }
2003
2004         public String getEname() {
2005             return ename;
2006         }
2007     }
2008
2009

```



```

2010     public void setName(String ename) {
2011         this.ename = ename;
2012     }
2013
2014     public double getSalary() {
2015         return salary;
2016     }
2017
2018     public void setSalary(double salary) {
2019         this.salary = salary;
2020     }
2021
2022     public String getDeg() {
2023         return deg;
2024     }
2025
2026     public void setDeg(String deg) {
2027         this.deg = deg;
2028     }
2029
2030     public Department getDepartment() {
2031         return department;
2032     }
2033
2034     public void setDepartment(Department department) {
2035         this.department = department;
2036     }
2037 }
2038
2039

```

2040 -Persistence.xml

- 2041 --Persistence.xml file is required to configure the database and the registration of entity classes.
- 2042 --Persistence.xml will be created by the eclipse IDE while creating a JPA Project.
- 2043 --The configuration details are user specifications.
- 2044 --The persistence.xml file is shown as follows:

```

2045     <?xml version="1.0" encoding="UTF-8"?>
2046     <persistence version="2.1"
2047         xmlns="http://xmlns.jcp.org/xml/ns/persistence"
2048         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2049         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
2050             http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
2051         <persistence-unit name="JPA_OTO" transaction-type="RESOURCE_LOCAL">
2052             <class>com.javasoft.entity.Employee</class>
2053             <class>com.javasoft.entity.Department</class>
2054
2055             <properties>
2056                 <property name="javax.persistence.jdbc.url" value="jdbc:h2:
2057                     tcp://localhost/~/test" />
2058                 <property name="javax.persistence.jdbc.user" value="sa" />
2059                 <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
2060                 <property name="eclipselink.logging.level" value="FINE" />
2061                 <property name="eclipselink.ddl-generation" value="create-tables" />
2062             </properties>
2063         </persistence-unit>
2064     </persistence>
2065

```

2066 -Service Classes

- 2067 --This module contains the service classes, which implements the relational part using the attribute initialization.
- 2068 --Create a package under 'src' package named 'com.javasoft.service'.
- 2069 --The DAO class named OneToOne.java is created under given package.
- 2070 --The DAO class is shown as follows:

```

2071     package com.javasoft.service;
2072

```

```

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import com.javasoft.entity.Department;
import com.javasoft.entity.Employee;

public class OneToOne {
    public static void main(String[] args) {

        EntityManagerFactory emfactory =
            Persistence.createEntityManagerFactory("JPA_OTO");
        EntityManager entitymanager = emfactory.createEntityManager();
        entitymanager.getTransaction().begin();

        // Create Department Entity
        Department department = new Department();
        department.setName("Development");

        // Store Department
        entitymanager.persist(department);

        // Create Employee Entity
        Employee employee = new Employee();
        employee.setEname("Satish");
        employee.setSalary(45000.0);
        employee.setDeg("Technical Writer");
        employee.setDepartment(department);

        // Store Employee
        entitymanager.persist(employee);

        entitymanager.getTransaction().commit();
        entitymanager.close();
        emfactory.close();
    }
}

```

-After compilation and execution of the above program you will get notifications in the console panel of Eclipse IDE.

-In this example two tables are created.

-Pass the following query in H2Database interface and the result of Department table in a tabular format is shown as follows in the query:

```
SELECT * FROM department;
```

Id	Name
101	Development

-Pass the following query in H2Database interface and the result of employee table in a tabular format is shown as follows in the query:

```
Select * from employee
```

Eid	Deg	Ename	Salary	Department_id
102	Technical Writer	Satish	45000	101

7)ManyToMany Relation

-Many-To-Many relationship is where one or more rows from one entity are associated with more than one row in other entity.

-Let us consider an example of relation between Class and Teacher entities.

-In bidirectional manner, both Class and Teacher have Many-To-One relation.

-That means each record of Class is referred by Teacher set (teacher ids), which should be primary keys in Teacher table and stored in Teacher_Class table and vice versa. Here, Teachers_Class table contains both foreign Key fields.

-Create a JPA project in eclipse IDE named JPA_MTM.

-All the modules of this project are shown as follows:

-Creating Entities

--Create a package named 'com.javasoft.entity' under 'src' package.

--Create a class named Clas.java under given package.

--The class Cals entity is shown as follows:

```
package com.javasoft.entity;

import java.util.Set;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

@Entity
public class Clas {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)

    private int cid;
    private String cname;

    @ManyToMany(targetEntity = Teacher.class)
    private Set teacherSet;

    public Clas() {
        super();
    }

    public Clas(int cid, String cname, Set teacherSet) {
        super();
        this.cid = cid;
        this.cname = cname;
        this.teacherSet = teacherSet;
    }

    public int getCid() {
        return cid;
    }

    public void setCid(int cid) {
        this.cid = cid;
    }

    public String getCname() {
        return cname;
    }

    public void setCname(String cname) {
        this.cname = cname;
    }

    public Set getTeacherSet() {
        return teacherSet;
    }

    public void setTeacherSet(Set teacherSet) {
        this.teacherSet = teacherSet;
    }
}
```

--Create the second entity in this relation - Teacher entity class named Teacher.java under

'com.javasoft.entity' package.

--The Teacher entity class is shown as follows:

```
package com.javasoft.entity;

import java.util.Set;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

@Entity
public class Teacher {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int tid;
    private String tname;
    private String subject;

    @ManyToMany(targetEntity = Clas.class)
    private Set clasSet;

    public Teacher() {
        super();
    }

    public Teacher(int tid, String tname, String subject, Set clasSet) {
        super();
        this.tid = tid;
        this.tname = tname;
        this.subject = subject;
        this.clasSet = clasSet;
    }

    public int getTid() {
        return tid;
    }

    public void setTid(int tid) {
        this.tid = tid;
    }

    public String getTname() {
        return tname;
    }

    public void setTname(String tname) {
        this.tname = tname;
    }

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    public Set getClasSet() {
        return clasSet;
    }

    public void setClasSet(Set clasSet) {
        this.clasSet = clasSet;
    }
}
```

```
2266     }  
2267 }  
2268
```

2269 -Persistence.xml

2270 --Persistence.xml file is required to configure the database and the registration of entity classes.

2271 --Persistence.xml will be created by the eclipse IDE while creating a JPA Project.

2272 --The configuration details are user specifications.

2273 --The persistence.xml file is shown as follows:

```
2274  
2275 <?xml version="1.0" encoding="UTF-8"?>  
2276 <persistence version="2.1"  
2277     xmlns="http://xmlns.jcp.org/xml/ns/persistence"  
2278     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
2279     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence  
2280     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">  
2281     <persistence-unit name="JPA_MTM" transaction-type="RESOURCE_LOCAL">  
2282         <class>com.javasoft.entity.Clas</class>  
2283         <class>com.javasoft.entity.Teacher</class>  
2284  
2285         <properties>  
2286             <property name="javax.persistence.jdbc.url" value="jdbc:h2:  
2287             tcp://localhost/~/test" />  
2288             <property name="javax.persistence.jdbc.user" value="sa" />  
2289             <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />  
2290             <property name="eclipselink.logging.level" value="FINE" />  
2291             <property name="eclipselink.ddl-generation" value="create-tables" />  
2292         </properties>  
2293     </persistence-unit>  
2294 </persistence>
```

2295 -Service Classes

2296 --This module contains the service classes, which implements the relational part using the attribute initialization.

2297 --Create a package under 'src' package named 'com.javasoft.service'.

2298 --The DAO class named ManyToMany.java is created under given package.

2299 --The DAO class is shown as follows:

```
2300  
2301 package com.javasoft.service;  
2302  
2303 import java.util.HashSet;  
2304 import java.util.Set;  
2305  
2306 import javax.persistence.EntityManager;  
2307 import javax.persistence.EntityManagerFactory;  
2308 import javax.persistence.Persistence;  
2309  
2310 import com.javasoft.entity.Clas;  
2311 import com.javasoft.entity.Teacher;  
2312  
2313 public class ManyToMany {  
2314     public static void main(String[] args) {  
2315  
2316         EntityManagerFactory emfactory =  
2317             Persistence.createEntityManagerFactory("JPA_MTM");  
2318         EntityManager entitymanager = emfactory.createEntityManager();  
2319         entitymanager.getTransaction().begin();  
2320  
2321         // Create Clas Entity  
2322         Clas clas1 = new Clas(0, "1st", null);  
2323         Clas clas2 = new Clas(0, "2nd", null);  
2324         Clas clas3 = new Clas(0, "3rd", null);  
2325  
2326         // Store Clas  
2327         entitymanager.persist(clas1);  
2328         entitymanager.persist(clas2);
```

```

2328         entityManager.persist(clas3);
2329
2330         // Create Clas Set1
2331         Set<Clas> classSet1 = new HashSet();
2332         classSet1.add(clas1);
2333         classSet1.add(clas2);
2334         classSet1.add(clas3);
2335
2336         // Create Clas Set2
2337         Set<Clas> classSet2 = new HashSet();
2338         classSet2.add(clas3);
2339         classSet2.add(clas1);
2340         classSet2.add(clas2);
2341
2342         // Create Clas Set3
2343         Set<Clas> classSet3 = new HashSet();
2344         classSet3.add(clas2);
2345         classSet3.add(clas3);
2346         classSet3.add(clas1);
2347
2348         // Create Teacher Entity
2349         Teacher teacher1 = new Teacher(0, "Satish", "Java", classSet1);
2350         Teacher teacher2 = new Teacher(0, "Krishna", "Adv Java", classSet2);
2351         Teacher teacher3 = new Teacher(0, "Masthanvali", "DB2", classSet3);
2352
2353         // Store Teacher
2354         entityManager.persist(teacher1);
2355         entityManager.persist(teacher2);
2356         entityManager.persist(teacher3);
2357
2358         entityManager.getTransaction().commit();
2359         entityManager.close();
2360         emfactory.close();
2361     }
2362 }

```

-After compilation and execution of the above program you will get notifications in the console panel of Eclipse IDE.

-In this example three tables are created.

-Pass the following query in H2Database interface and the result of teacher_clas table in a tabular format is shown as follows in the query:

```
Select * from teacher_clas;
```

Teacher _tid	Classet_cid
154	151
155	151
156	151
154	152
155	152
156	152
154	153
155	153
156	153

-In the above table teacher_tid is the foreign key from teacher table, and classet_cid is the foreign key from class table.

-Therefore different teachers are allotted to different class.

-Pass the following query in H2Database interface and the result of teacher table in a tabular format is shown as follows in the query:

```
Select * from teacher;
```

Tid	Subject	Tname
154	Java	Satish
155	Adv Java	Krishna
156	DB2	Masthanvali

-Pass the following query in H2Database interface and the result of clas table in a tabular format is shown as follows in the query:

Select * from clas;

cid	Cname
151	1st
152	2nd
153	3rd

8. Criteria API

- 1)The Criteria API is a predefined API used to define queries for entities.
- 2)It is the alternative way of defining a JPQL query.
- 3)These queries are type-safe, and portable and easy to modify by changing the syntax.
- 4)Similar to JPQL it follows abstract schema (easy to edit schema) and embedded objects.
- 5)The metadata API is mingled with criteria API to model persistent entity for criteria queries.
- 6)The major advantage of the criteria API is that errors can be detected earlier during compile time.
- 7)String based JPQL queries and JPA criteria based queries are same in performance and efficiency.
- 8)History of criteria API

-The criteria API is included into all versions of JPA therefore each step of criteria API is notified in the specifications of JPA.

-In JPA 2.0, the criteria query API, standardization of queries are developed.

-In JPA 2.1, Criteria update and delete (bulk update and delete) are included.

9)Criteria Query Structure

-The Criteria API and the JPQL are closely related and are allowed to design using similar operators in their queries.

-It follows javax.persistence.criteria package to design a query.

-The query structure means the syntax criteria query.

-The following simple criteria query returns all instances of the entity class in the data source.

```
EntityManager em = ...;
CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<Entity class> cq = cb.createQuery(Entity.class);
Root<Entity> from = cq.from(Entity.class);

cq.select(Entity);
TypedQuery<Entity> q = em.createQuery(cq);
List<Entity> allitems = q.getResultList();
```

-The query demonstrates the basic steps to create a criteria.

--EntityManager instance is used to create a CriteriaBuilder object.

--CriteriaQuery instance is used to create a query object.

--This query object's attributes will be modified with the details of the query.

--CriteriaQuery.from method is called to set the query root.

--CriteriaQuery.select is called to set the result list type.

--TypedQuery<T> instance is used to prepare a query for execution and specifying the type of the query result.

--getResultList method on the TypedQuery<T> object to execute a query.

--This query returns a collection of entities, the result is stored in a List.

10)Example of criteria API

-Let us consider the example of employee database.

-Let us assume the jpadb.employee table contains following records:

Eid	Ename	Salary	Deg
401	Gopal	40000	Technical Manager
402	Manisha	40000	Proof reader
403	Masthanvali	35000	Technical Writer
404	Satish	30000	Technical writer
405	Krishna	30000	Technical Writer
406	Kiran	35000	Proof reader

-Create a JPA Project in the eclipse IDE named JPA__Criteria.

-All the modules of this project are shown as follows:

-Creating Entities

--Create a package named com.javasoft.entity under 'src' package.

--Create a class named Employee.java under given package.

--The class Employee entity is shown as follows:

```
package com.javasoft.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)

    private int eid;
    private String ename;
    private double salary;
    private String deg;

    public Employee(int eid, String ename, double salary, String deg) {
        super();
        this.eid = eid;
        this.ename = ename;
        this.salary = salary;
        this.deg = deg;
    }

    public Employee() {
        super();
    }

    public int getEid() {
        return eid;
    }

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

    public String getEname() {
        return ename;
    }

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

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public String getDeg() {
        return deg;
    }

    public void setDeg(String deg) {
        this.deg = deg;
    }
}
```



```

2521
2522         @Override
2523         public String toString() {
2524             return "Employee [eid = " + eid + ", ename = " + ename + ", salary = " + salary +
                ", deg = " + deg + "]\n";
2525         }
2526     }
2527

```

-Persistence.xml

```

2529     --Persistence.xml file is required to configure the database and the registration of entity
        classes.
2530     --Persistence.xml will be created by the eclipse IDE while cresting a JPA Project.
2531     --The configuration details are user specification.
2532     --The persistence.xml file is shown as follows:
2533

```

```

2534     <?xml version="1.0" encoding="UTF-8"?>
2535     <persistence version="2.1"
        xmlns="http://xmlns.jcp.org/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
        http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
2539
2540         <persistence-unit name="JPA_Criteria" transaction-type="RESOURCE_LOCAL">
2541
2542             <class>com.javasoft.entity.Employee</class>
2543             <properties>
2544                 <property name="javax.persistence.jdbc.url"
2545                     value="jdbc:h2:tcp://localhost/~/test" />
2546                 <property name="javax.persistence.jdbc.user" value="sa" />
2547                 <property name="javax.persistence.jdbc.driver"
2548                     value="org.h2.Driver" />
2549                 <property name="eclipselink.logging.level" value="FINE" />
2550                 <property name="eclipselink.ddl-generation"
2551                     value="create-tables" />
2552             </properties>
2553
2554         </persistence-unit>
2555     </persistence>
2556

```

-Service classes

```

2558     --This module contains the service classes, which implements the Criteria query part using
        the MetaData API initialization.
2559     --Create a package named 'com.javasoft.service'.
2560     --The class named CriteriaAPI.java is created under given package.
2561     --The DAO class is shown as follows:
2562

```

```

2563         package com.javasoft.service;
2564
2565         import java.util.List;
2566
2567         import javax.persistence.EntityManager;
2568         import javax.persistence.EntityManagerFactory;
2569         import javax.persistence.Persistence;
2570         import javax.persistence.TypedQuery;
2571         import javax.persistence.criteria.CriteriaBuilder;
2572         import javax.persistence.criteria.CriteriaQuery;
2573         import javax.persistence.criteria.Root;
2574
2575         import com.javasoft.entity.Employee;
2576
2577         public class CriteriaApi {
2578             public static void main(String[] args) {
2579
2580                 EntityManagerFactory emfactory = Persistence.createEntityManagerFactory(
                "JPA_Criteria" );
2581                 EntityManager entitymanager = emfactory.createEntityManager( );
2582                 CriteriaBuilder criteriaBuilder = entitymanager.getCriteriaBuilder();

```

```

2583 CriteriaQuery<Object> criteriaQuery = criteriaBuilder.createQuery();
2584 Root<Employee> from = criteriaQuery.from(Employee.class);
2585
2586 //select all records
2587 System.out.println("Select all records");
2588 CriteriaQuery<Object> select = criteriaQuery.select(from);
2589 TypedQuery<Object> typedQuery = entityManager.createQuery(select);
2590 List<Object> resultlist = typedQuery.getResultList();
2591
2592 for(Object o:resultlist) {
2593     Employee e = (Employee)o;
2594     System.out.println("EID : " + e.getId() + " Ename : " + e.getName());
2595 }
2596
2597 //Ordering the records
2598 System.out.println("Select all records by follow ordering");
2599 CriteriaQuery<Object> select1 = criteriaQuery.select(from);
2600 select1.orderBy(criteriaBuilder.asc(from.get("ename")));
2601 TypedQuery<Object> typedQuery1 = entityManager.createQuery(select);
2602 List<Object> resultlist1 = typedQuery1.getResultList();
2603
2604 for(Object o:resultlist1){
2605     Employee e=(Employee)o;
2606     System.out.println("EID : " + e.getId() + " Ename : " + e.getName());
2607 }
2608
2609 entityManager.close( );
2610 emfactory.close( );
2611 }
2612 }
2613

```

-After compilation and execution of the above program you will get output in the console panel of Eclipse IDE as follows:

```

2615
2616 Select All records
2617 EID : 401 Ename : Gopal
2618 EID : 402 Ename : Manisha
2619 EID : 403 Ename : Masthanvali
2620 EID : 404 Ename : Satish
2621 EID : 405 Ename : Krishna
2622 EID : 406 Ename : Kiran
2623 Select All records by follow Ordering
2624 EID : 401 Ename : Gopal
2625 EID : 406 Ename : Kiran
2626 EID : 405 Ename : Krishna
2627 EID : 402 Ename : Manisha
2628 EID : 403 Ename : Masthanvali
2629 EID : 404 Ename : Satish
2630
2631

```

9. 왜 ORM 인가?

1)보통 한국은 iBatis/MyBatis를 현업에서 대부분 사용하는데, query 자체에 집중하고 business logic을 query에 의존하게 되면 객체지향의 장점을 놓칠 수 있는 가능성이 크다.

2)예를 들어 User table과 Order table이 있다.

3)크게 복잡하지 않은 1(user):N(order) 관계의 table이다.

```

2636 tbl_user
2637 user_id INT
2638 username VARCHAR(20) PRIMARY KEY
2639 nick_name VARCHAR(20)
2640 address VARCHAR(100)
2641
2642 tbl_order
2643 order_id INT PRIMARY KEY
2644 user_id INT FOREIGN KEY REFERENCES tbl_user(user_id)
2645 order_name VARCHAR(45)
2646 note VARCHAR(45)
2647

```

4)즉 한명의 user는 여러 order를 가질 수 있고 order는 fk로 user_id를 가지고 있다.

5)보통 query를 이용해서 이 table을 객체로 만들때는

```
public class User {
    private int userId;
    private String username;
    private String nickName;
    private String address;
}

public class Order {
    private int orderId;
    private String orderName;
    private String note;
    private int userId;
}
```

6)이런 식의 DTO 또는 VO class를 만드는데 User가 자신이 생성한 Order data를 가져오기 위해서는

```
SELECT * FROM tbl_user u JOIN tbl_order o ON u.user_id = o.user_id
WHERE u.user_id = #{value}
```

7)또는

```
SELECT * FROM tbl_order WHERE user_id = #{value}
```

8)이렇게 query가 작성 된다.

9)여기서 주의해서 봐야될 부분은 바로 Order class에 userId 변수다.

10)이 변수는 Order를 생성한 User의 id를 가지고 있다.

11)즉 table 관점으로 본다면 userId 변수는 fk가 되는 것이다.

12>User 객체가 아니라 User의 pk만 가지고 있는 것이다.

13)그럼, ORM에서는 객체를 어떻게 구성할까?

```
@Entity(name = "tbl_user")
public class User {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer userId;
    private String username;
    private String nickName;
    private String address;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Order> orders;
}

@Entity(name = "tbl_order")
public class Order {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer orderId;
    private String orderName;
    private String note;
    private int price;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id")
    private User user;
}
```

14>User 객체는 Order List 객체를, Order 객체는 User 객체를 가지고 있다.

15)Table 관점으로 보나, 객체 관점으로 보나 pk, fk도 만족을 하며, 객체지향적으로 객체를 만든 것이다.

16)심지어 data를 조작하기 위한 query 자체도 신경쓰지 않는다.

17)Query 자체를 신경쓰지 않는다는 뜻은 실제 User 객체나 Order 객체를 table에서 조회해서 만드는 과정에 있어 위 예제와 같이 query를 작성해서 만들지 않는다는 뜻이다.

18)내부적으로는 query가 실행되고 실제로 우리는 그 query가 어떻게 만들어지고 실행되는지 정확히 인지하고 있어야 정확한 JPA 사용이 가능하다.

19)우리가 query를 작성하지 않는다는 뜻이지 query가 실제로 실행 안된다는 뜻은 아니다.

20)이렇게 객체지향적인 객체가 생성되면 우리 business logic 자체를 query에 집중하지 않고 Java code 자체에 집중

할 수 있다는 것이다.

21)Code에 business logic을 집중할 수 있다는 것은 많은 장점이 있는데 유지보수, testing, debugging, 객체지향과 같은 여러 장점을 갖을 수 있다.

22)물론 xBatis나 JDBC, Spring JDBC Template과 같은 query를 작성하는 기술을 사용해도 객체지향적인 객체 설계가 가능하지만 좀 더 쉽고 빠르게 쓸 수 있는 것이 ORM이다.

10. 영속성 컨텍스트(Persistence Context)

1)EntityManager를 하나 만들때 마다 생성되며 조회, 저장 시에 EntityManager는 영속성 context에 entity를 보관하게 된다.

2)Entity의 생명주기

-비영속(new/transient)

--영속성 context와 전혀 관계없는 상태 순수한 객체 상태, 아직 DB와 전혀 상관없는 상태이다.

--entityManager.persist()호출 이전

-영속(managed)

--영속성 context에 저장된 상태 entityManager.persist() 호출 이후의 상태.

--즉 영속 상태란, 영속성 context에 의해 관리된다는 의미이다

-준영속(detached)

--영속성 context에 저장되었다가 분리된 상태.

--detach(), close()를 호출한 상태

--entityManager.clear()를 호출해서 영속성 context를 초기화 해도 해당 entity는 준영속 상태로 유지된다

-삭제(removed)

--삭제된 상태

--entityManager.remove(entity)와 같이 영속성 context와 DB에서 삭제된 상태.

3)영속성 context의 특징

-영속성 context에는 @Id로 식별자가 꼭 존재해야 한다.

-영속성 context에서 DB로의 저장은 commit(flush)시에 수행된다.

-영속어 conext를 사용하면 1차 cache, 동일성 보장, transaction을 지원하는 쓰기 지연, 변경감지, 지연로딩 등의 장점이 존재한다.

4)Entity조회

-영속성 context 내부의 cache를 1차 cache라 부른다.

-내부에 Map형태의 정보를 저장하고 @Id로 mapping된 정보가 저장되어 있다.

-즉 1차 cache의 key는 식별자(@Id)값이다

-entityManager.find()를 호출하면 우선 1차 cache에서 검색한다.

-1차 cache에 없다면 DB를 검색하고 결과를 1차 cache에 저장 한다.

-이를 통해서 객체의 동일성을 보장 할 수 있다.

5)Entity 등록

-Entity Manager는 transaction을 commit하기 전까지 entity를 DB에 저장하지 않고 query 저장소에 INSERT SQL문을 차곡 차곡 쌓아 놓는다.

-그리고 commit시에 한번에 query를 보낸다.

-이것이 바로 transaction을 지원하는 쓰기 지연이다.

-이 기능을 잘 사용하면 모아둔 등록 query를 database에 한번에 전달해 성능을 최적화 할 수 있다.

6)Entity 수정

-Project가 커져가면서 수정 사항은 항상 늘어난다.

-SQL문으로 이러한 문제를 대처 하고 있다면 수정 사항이 생길때 마다 query를 추가 하거나 수정 해야 한다.

-JPA에서의 entity 수정은 entity manger에서 받아온 entity의 값을 변경해 줌으로써 가능 하다.

-변경사항의 DB적용은 commit 시에 저장 된다.

-수정 query문은 변경 사항만 가지고 만들어지지 않고 전체 field를 이용해서 만들어진단.

-이는 항상 동일한 query(값은 다르지만)가 만들어진다는 장점과 용량이 커질수 있다는 단점이 있다.

-용량문제가 critical하다면 hibernate의 DynamicUpdate 기능을 사용하는 방법도 있다(30개 이상의 column에서는 효과가 있다고 한다).

7)변경 감시

-Transaction을 commit하면 EntityManager 내부에서 flush()가 일어나고 entity와 snapshot을 비교해서 변경사항을 SQL query로 만들어 SQL 저장소에 보낸다.

-그리고 쓰기 지연 저장소의 SQL을 DB에 보내고 commit한다.

-변경 감시는 영속성 context가 관리하는 영속 상태의 entity에만 적용된다.

8)Entity 삭제

-삭제를 하려면 먼저 entity를 조회 해야 한다.

-조회된 entity를 넘겨주면 삭제가 된다.

-이때도 당연히 삭제 query는 쓰기 지연 SQL 저장소에 저장 되고 commit 시에 수행 된다.

9)Flush

- 영속성 context의 변경 내용을 DB에 반영하는 것을 말한다.
- 변경 감지가 동작해서 영속성 context에 있는 모든 entity를 snapshot과 비교해서 수정된 정보를 찾고 query로 만든다.
- 쓰기 지연 SQL 저장소의 query를 DB로 전송하다.
- Flush가 호출되는 시점은 크게 3가지가 있다.
 - 직접 호출
 - Transaction commit 시 자동 호출
 - JPQL query 실행시 자동 호출

10)Flush mode

- FlushModeType.AUTO (default)
 - commit이나 commit 시에 수행됨
- FlushModeType.COMMIT
 - commit시에만 수행

11. Entity Mapping

1)Mapping annotation 을 크게 4가지로 분류하면

- 객체와 table mapping : @Entity, @Table
- 기본 key mapping : @Id
- field와 column mapping : @column
- 연관관계 mapping : @ManyToOne, @JoinColumn

2)@Entity

- class에 @Entity를 붙여주면 JPA가 Entity로서 관리 한다는 것을 의미한다.
- 속성
 - name
 - 다른 Entity와 충돌이 우려될 경우 이름을 바꿔준다.
 - 기본적으로는 Class명을 따른다.
- 주의사항
 - 기본 생성자 필수
 - final, enum, interface, inner-class 사용 못함
 - 저장 field에 final 사용 못함

3)@Table

- Entity와 mapping할 DB Table을 지정 한다.
- 속성
 - name
 - mapping할 table 이름, 기본은 Entity 이름을 사용한다.
 - catalog
 - catalog 기능이 있는 DB에서 catalog를 매핑
 - schema
 - schema 기능이 있는 DB에서 schema를 매핑
 - uniqueConstraints
 - DDL 생성 시에 unique 제약조건을 만든다.

4) Database schema 자동 생성

- 자동으로 schema를 생성하는 기능은 아래 값을 설정 함으로써 가능하다.

```
<property name="hibernate.hbm2ddl.auto" value="create-drop"/>
```

-value

- create
 - 기존 Table Drop + 생성
- create-drop
 - create후 종료시 drop까지 실행
- update
 - 변경된 내용만 수정한다.
 - 이건 JPA spec에는 없고 hibernate에만 있는 설정 이다.
- validate
 - 기존 DB Table정보와 비교해서 차이가 있다면 경고하고 application을 실행 하지 않는다.
 - 이건 JPA spec에는 없고 hibernate에만 있는 설정 이다.
- none
 - 설정이 없거나 유효하지 않은 값을 설정하면 기능을 사용하지 않게된다.

5)기본 key mapping

- primary key를 설정하는 것을 말한다.

```

2839 -직접할당
2840     --em.persist()를 호출하기전에 사용자가 직접 ID를 설정하는 것을 말한다.
2841
2842     Board board = new Board();
2843     board.setId("board1");
2844     em.persist(board);
2845
2846 -자동생성
2847     --IDENTITY
2848     ---기본 key의 생성을 DB에 위임하는것을 말한다.
2849     ---MySQL의 AUTO_INCREMENT와 같은것을 말한다.
2850     ---@GeneratedValue(strategy = GenerationType.IDENTITY) 로 설정 가능
2851
2852     --SEQUENCE
2853     ---유일한 값을 순서대로 생성하는 특별한 database object를 이용하는 방법을 말한다.
2854     ---@GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
2855         "BOARD_SEQ_GENERATOR") 로 설정 가능
2856
2857     @Entity
2858     @SequenceGenerator(
2859         name = "BOARD_SEQ_GENERATOR",
2860         sequenceName = "BOARD_SEQ", // 실제 DB의 Sequence Name
2861         initialValue = 1, allocationSize = 1 )
2862     public class Board {
2863         ...
2864         @Id
2865         @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
2866             "BOARD_SEQ_GENERATOR")
2867         private Long id;
2868         ...
2869     }
2870
2871     --TABLE
2872     ---key 생성 전용 Table을 만들어서 이를 SEQUENCE 처럼 사용하는 것을 말한다.
2873
2874     @Entity
2875     @TableGenerator(
2876         name = "BOARD_SEQ_GENERATOR",
2877         table = "MY_SEQUENCE", // 실제 DB의 Table name
2878         pkColumnName = "BOARD_SEQ", allocationSize = 1 )
2879     public class Board {
2880         ...
2881         @Id
2882         @GeneratedValue(strategy = GenerationType.TABLE, generator =
2883             "BOARD_SEQ_GENERATOR")
2884         private Long id;
2885         ...
2886     }
2887
2888     --AUTO
2889     ---DB종류에 따라 JPA가 알맞은 것을 선택하는것을 의미한다.
2890     ---Oracle의 경우 SEQUENCE, MySQL의 경우 IDENTITY를 선택하게 된다.
2891     ---또하나의 장점은 DB종류가 바뀌어도 source를 수정하지 않아도 된다는 것이다.
2892     ---@GeneratedValue(strategy = GenerationType.AUTO) 로 설정 가능

```

6)field와 column mapping : reference

```

2892 -field와 column mapping에 사용되는 annotation으로는 @Column, @Enumerated, @Temporal, @Lob,
2893     @Transient, @Access 가 있다.
2894
2895 -@Column
2896     --객체 field를 table column과 mapping해주는 가장 대표적인 annotation이다.
2897     --name, nullable이 가장 많이 사용되고 나머지는 많이 사용되지는 않는다.
2898     --속성
2899         ---name
2900             ----mapping할 table column 이름, 기본은 객체의 field 이름을 사용한다.
2901         ---insertable
2902             ----거의 사용안됨, entity 저장시 이 field도 저장하라는 의미로 기본은 true이다.
2903             ----단 false로 하면 Readonly일때 사용가능 하다.

```

```

2902      ---updateable
2903          ----거의 사용안됨, entity 수정시 이 field도 수정하라는 의미로 기본은 true이다.
2904          ----단 false로 하면 Readonly일때 사용가능 하다.
2905      ---table
2906          ----거의 사용안됨, 하나의 entity를 두 개 이상의 table에 mapping할때 사용
2907      ---nullable
2908          ----false로 설정하면 DDL생성시에 "NOT NULL" 제약조건을 추가해 준다.
2909      ---unique
2910          ----@Table 의 uniqueConstraints와 같지만 한 column에 대해서 적용할때는 간단하게 이것을 이용가능
2911          ----단 여러 column을 사용할때는 @Table의 uniqueConstraints를 사용해야 한다.
2912      ---columnDefinition
2913          ----사용자가 직접 column의 정보를 입력해준다.
2914      ---length
2915          ----문자 길이에 대한 제약조건을 준다.
2916          ----String 타입에만 적용되며 기본값은 255이다.
2917      ---precision, scale
2918          ----BigDecimal 타입에서 사용된다.
2919          ----precision은 소수점을 포함한 전체 지릿수, scale은 소수 자리수를 의미.
2920          ----float, double에는 해당 되지 않는다.
2921
2922      -@Enumerated
2923          --enum type을 mapping할 때 사용한다.
2924          --속성
2925              ---name
2926                  ----EnumType.ORDINAL
2927                      -----enum의 순서를 DB에 저장, 이값이 Default이다.
2928                      -----숫자로 저장되므로 data 크기가 작아지고 빠르다.
2929                      -----enum의 순서를 변경할 수 없는 단점이 있다.
2930                  ----EnumType.STRING
2931                      -----enum이름을 DB에 저장
2932                      -----문자로 저장되므로 data 크기가 커지고 느리다. 하지만 enum의 순서와 상관없이 사용가능해 진다.
2933                      -----Default는 ORDINAL이지만 STRING을 더 추천 한다.
2934
2935      -@Temporal
2936          --날짜 type을 mapping할 때 사용 속성을 입력 하지 않으면 Java의 Date과 가장 유사한 Timestamp로 저장
2937          --된다(H2, Oracle, PostgreSQL).
2938          --하지만 이는 DB의 종류에 따라 Datetime으로 저장되기도 한다(MySQL).
2939          --속성
2940              ---value
2941                  ----TemporalType.DATE
2942                      -----2013-01-23 와 같은 날짜 타입
2943                  ----TemporalType.TIME
2944                      -----11:23:18 과 같은 시간 타입
2945                  ----TemporalType.TIMESTAMP
2946                      -----2013-01-23 11:23:18 과 같이 DB의 Timestamp 타입과 매핑
2947
2948      -@Lob
2949          --@Lob는 별도의 속성은 없다.
2950          --대신 mapping을 문자열이면 CLOB, 그외의 type에는 BLOB으로 매핑한다.
2951          ---CLOB : String, char[], java.sql.CLOB
2952          ---BLOB : byte[], java.sql.BLOB
2953
2954      -@Transient
2955          --이 field는 매핑하지 말라는 의미이다.
2956          --이는 임시로 중간 값을 저장하는 용도로 사용가능하다.
2957
2958      -@Access
2959          --JPA가 entity data에 접근하는 방식
2960          --@Access를 설정하지 않으면 @Id의 설정위치에 따라서 접근 방식이 결정 된다.
2961          --@Id가 필드에 붙어 있으면 FIELD접근 방식을 의미하므로 Getter가 없어도 된다.
2962          --@Id가 property에 있으면 PROPERTY 접근 방식을 의미하게 된다.
2963          --하지만 이 두가지 방식을 섞어서 사용도 가능하다.
2964          ---AccessType.FIELD
2965              ----field 접근
2966              ----Private이어도 접근 가능하다.
2967          ---AccessType.PROPERTY
2968              ----property 접근

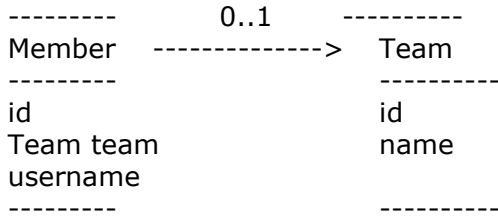
```

----접근자(Getter)를 이용한다.

12. 연관관계 mapping 기초

1) 단방향 연관관계

- 연관관계중에서는 다대일(N:1)을 가정 먼저 이해 하고 넘어가야 한다.
- 회원과 팀의 관계가 그에 해당 한다.
- 회원과 팀이 있다.
- 회원은 하나의 팀에만 속할 수 있다.



-객체 연관관계

- Member 객체는 Member.team field로 Team객체와 연관 관계를 맺고 있고 이는 단방향 관계이다.
- 즉 Member는 team 필드로 팀을 알수 있지만 Team은 Member를 알아 낼 수 없다.

-테이블 연관관계

- 반면 table은 외래 key를 이용해서 양방향으로 조회 할 수 있다.
- 즉, 외래 key를 이용해서 서로 JOIN해주면 양방향의 관계를 알아 낼 수 있다.

-객체 관계 매핑

@Entity

```
public class Member {
    @Id
    @Column(name = "MEMBER_ID")
    private String id;

    private String username;

    @ManyToOne
    @JoinColumn(name = "TEAM_ID")
    private Team team;

    // Getter, Setter...
}
```

@Entity

```
public class Team {
    @Id
    @Column(name = "TEAM_ID")
    private String id;

    private String name;

    // Getter, Setter...
}
```

-@ManyToOne

- 말 그대로 N:1의 관계를 라는 mapping 정보 이다.
- 속성
 - optional
 - 기본은 true인데 false로 설정하면 연관된 Entity가 꼭 있어야 한다.
 - fetch
 - FetchType.EAGER
 - FetchType.LAZY
 - cascade
 - 영속성 전이 기능을 사용
 - targetEntity
 - 연관된 entity의 type 정보를 설정


```

3035 -@OneToMany
3036
3037     @OneToMany
3038     private List<Member> members // generic으로 type 정보를 알수 있다.
3039
3040     @OneToMany(targetEntity=Member.class)
3041     private List member // generic이 없으면 type 정보를 알수 없다.
3042
3043 -@JoinColumn(name="TEAM_ID")
3044     --JoinColumn 은 외래 key를 mapping할 때 사용한다.
3045     --속성
3046         ---name
3047             ----mapping할 외래 key 이름
3048         ---referencedColumnName
3049             ----외래 key가 참조하는 대상 table의 column 이름
3050         ---foreignKey(DDL)
3051             ----외래 key 제약조건을 직접 설정하기위한 속성이다.
3052         ---unique
3053         ---nullable
3054         ---insertable
3055         ---updatable
3056         ---columnDefinition
3057         ---table
3058             ----@Column속성과 같다.
3059
3060
3061 -연관관계 사용
3062     --저장
3063         ---JPA에서 entity를 저장할 때 연관된 모든 entity는 영속 상태여야 한다.
3064
3065         Team team1 = new Team("team1", "팀1");
3066         em.persist(team1);
3067
3068         Member member1 = new Member("member1", "회원1");
3069         member1.setTeam(team1);
3070         em.persist(member1);
3071
3072         Member member2 = new Member("member2", "회원2");
3073         member2.setTeam(team1);
3074         em.persist(member2);
3075
3076     --조회
3077         ---연관관계가 있는 entity를 조회 하는 방법은 "객체 graph 탐색" 방법과 "객체지향 query 사용" 이 있다.
3078         ----객체 graph 탐색
3079
3080         Member member = em.find(Member.class, "member1");
3081         Team team = member.getTeam(); // <---- 이것이 객체 그래프 탐색이다.
3082
3083         ----객체지향 query 사용
3084             -----객체지향 query인 JPQL을 이용한 방법이다.
3085
3086         String jpql = "select m from Member m join m.team t where t.name = :teamName";
3087
3088         List<Member> resultList = em.createQuery(jpql, Member.class)
3089             .setParameter("teamName", "팀1")
3090             .getResultList();
3091
3092     -수정
3093         --em.update()와 같은 함수는 없다.
3094         --단순히 아래와 같이 처리하면 된다.
3095
3096         Team team2 = new Team("team2", "팀2");
3097         em.persist(team2);
3098
3099         Member member = em.find(Member.class, "member1");
3100         member.setTeam(team2);
3101

```

-연관 관계 제거
--아래와 같이 null로 처리해 주면 관계는 삭제 된다.

```
Member member = em.find(Member.class, "member1");  
member.setTeam(null);
```

-연관된 entity 삭제
--연관된 entity를 삭제 하려면 해당 entity를 사용하는 모든 entity의 연관 관계를 제거 해줘야 한다.

```
member1.setTeam(null);  
member2.setTeam(null);  
em.remove(team);
```

2) 양방향 연관관계

-단방향에서는 "회원" -> "팀" 으로만 확인이 가능했다면 양방향에서는 "팀" -> "회원" 으로도 확인이 가능해야 한다.
-RDB의 Table에서의 관계는 양방향과 단방향이 모두 동일하다.(FK로 서로 교차 검색이 가능하다)

```
@Entity  
public class Team {  
    @Id  
    @Column(name = "TEAM_ID")  
    private String id;  
  
    private String name;  
  
    // 추가  
    @OneToMany(mappedBy = "team")  
    private List<Member> members = new ArrayList<Member>();  
  
    // Getter, Setter...  
}
```

-mappedBy는 양방향 매핑을 할때 반대쪽에 매핑할 필드의 이름이다.

-연관관계의 주인

--mappedBy는 왜? 필요할까??

---객체에서 양방향 관계라는 것은 없다. -> 2개의 단방향 관계가 있는 것이다.

---mappedBy는 연관관계의 주인을 정해주는 작업이다.

---연관관계의 주인은 외래 key가 존재하는 entity가 된다.

---즉, 연관관계 주인 만이 연관관계를 갱신할 수 있고, 반대편(inverse, non-owning side)는 읽기만 가능하다.

-양방향 연관관계 저장 및 주의할 점

--예제에서는 Member.team 연관관계의 주인이므로 아래 코드의 내용에 주의 해줘야한다.

// 정상적인 연관관계의 설정은 Member 에서 이루어져야 한다.

```
Team team1 = new Team("team1", "팀1");  
em.persist(team1);
```

```
Member member2 = new Member("member2", "회원2");  
member.setTeam(team1);  
em.persist(member2);
```

// Team에서 연관관계를 업데이트 하려고 시도하는것은 무시된다.

```
Member member2 = new Member("member2", "회원2");  
em.persist(member2);
```

```
Team team1 = new Team("team1", "팀1");  
team1.getMembers().add(member2);  
em.persist(team1);
```

--하지만 순수 객체로 따져 보면 위 예제의 정상과 비정상 예제의 내용이 모두 실행되어야 정상적으로 동작하게 된다.

--그래서 Member의 setTeam을 아래와 같이 수정해 주면 순수 객체에서도 정상동작 하게 된다.

```
public void setTeam(Team team){  
    // 기존팀 관계 제거  
    // -> 영속성 컨텍스트가 새로 시작되면 문제가 없지만 영속성 컨텍스트를 계속 사용중이라면  
    // 문제가 발생 할 수 있다.
```

```

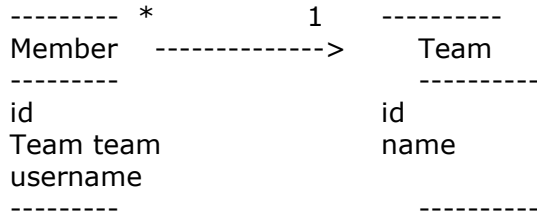
3169         if (this.team != null){
3170             this.team.getMembers().remove(this);
3171         }
3172         this.team = team;
3173         team.getMembers().add(this);
3174     }
3175
3176

```

13. 다양한 연관관계 mappinig

1) 다대일

-다대일 단방향 [N:1]



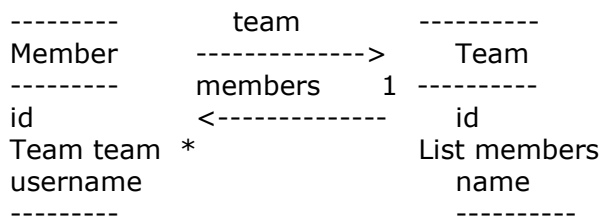
--Member에서는 @MenyToOne 으로 Team을 참조 할 수 있도록 해주고 Team에서는 별도의 설정을 하지 않는다.

```

3191         <Member>
3192         @Entity
3193         public class Member {
3194             @Id @GeneratedValue
3195             @Column(name = "MEMBER_ID")
3196             private Long id;
3197
3198             private String username;
3199
3200             @ManyToOne
3201             @JoinColumn(name = "TEAM_ID")
3202             private Team team;
3203         }
3204
3205         <Team>
3206         @Entity
3207         public class Team {
3208             @Id @GeneratedValue
3209             @Column(name = "TEAM_ID")
3210             private Long id;
3211
3212             private String name;
3213         }
3214

```

-다대일 양방향 [N:1, 1:N]



--Member에서는 @MenyToOne 으로 Team을 참조 할 수 있도록 해주고 Team에서는 @OneToMany 와 함께 mappedBy값을 설정 해서 연관관계의 주인이 Member table임을 알려준다.

--항상 1:N, N:1 관계에서는 항상 N쪽에 외래 key가 있다.

--여기에서는 N쪽인 Member table이 외래 key를 가지게 된다.

--그러므로 Member.team이 연관관계의 주인이 된다.

```

3230         <Member>
3231         @Entity
3232         public class Member {
3233             @Id @GeneratedValue
3234             @Column(name = "MEMBER_ID")

```

```

3235         private Long id;
3236
3237         private String username;
3238
3239         @ManyToOne
3240         @JoinColumn(name = "TEAM_ID")
3241         private Team team;
3242
3243         public void setTeam(Team team){
3244             this.team = team;
3245
3246             if(!team.getMembers().contains(this)){
3247                 team.getMembers().add(this);
3248             }
3249         }
3250     }
3251
3252     <Team>
3253     @Entity
3254     public class Team {
3255         @Id @GeneratedValue
3256         @Column(name = "TEAM_ID")
3257         private Long id;
3258
3259         @OneToMany(mappedBy = "team")
3260         private List<Member> members
3261
3262         private String name;
3263
3264         public void addMember(Member member){
3265             this.members.add(member);
3266             if(member.getTeam() != this){
3267                 member.setTeam(this);
3268             }
3269         }
3270     }
3271
3272
3273

```

2) 일대다

- 일대다 단방향 [N:1]

```

3276         ----- 1 * -----
3277         Team -----> Member
3278         ----- members -----
3279         id id
3280         name username
3281         List members
3282         -----
3283

```

3284 --보통은 자신이 mapping한 table의 외래 key를 가지는데 반해 여기서는 Member table에 외래 key가 있다.
3285 --일대다 관계에서는 꼭 JoinColumn을 선언 해줘야 한다.
3286 --안그러면 join table을 중간에 두는 방식으로 설정 된다.
3287 --일대다의 단점은 mapping한 객체가 관리하는 외래 key가 다른 table에 있으므로 Insert SQL한번으로 조작이
 끝나는것이 아닌 연관 테이블의 Update도 해줘야하는 문제가 있다.
 --그러므로, 일대다 보다는 다대일 단방향 mapping을 사용하자.

```

3288
3289
3290     <Team>
3291     @Entity
3292     public class Team {
3293         @Id @GeneratedValue
3294         @Column(name = "TEAM_ID")
3295         private Long id;
3296
3297         @OneToMany
3298         @JoinColumn(name = "TEAM_ID") // Member table의 TEAM_ID 를 나타냄
3299         private List<Member> members;
3300

```

```

3301         private String name;
3302     }
3303
3304     <Member>
3305     @Entity
3306     public class Member {
3307         @Id @GeneratedValue
3308         @Column(name = "MEMBER_ID")
3309         private Long id;
3310
3311         private String username;
3312     }
3313

```

3314 -일대다 단방향 [N:1]

- 3315 --일대다 양방향 mapping은 존재 하지 않는다.
- 3316 --대시 다대일 mapping을 사용해야 한다.
- 3317 --다시 말해 양방향 mapping에서는 @OneToMany는 연관관계의 주인이 될 수 없다는 것이다.

3320 3)일대일

- 3321 -양쪽이 서로 하나의 관계만을 가진다.
- 3322 -즉, 일대일 관계는 그 반대도 일대일 관계다.
- 3323 -주 table에 외래 key
- 3324 --JPA에서는 주 table에 외래 key가 있는것이 좀더 편리하게 mapping할 수 있게된다.
- 3325 -단방향
- 3326 --Member 와 Locker의 관계로 알아보자.
- 3327 --아래 모습은 다대일(N:1) 단방향 모습과 매우 흡사하다.

```

3329 ----- 1          1 -----
3330 Member -----> Locker
3331 -----
3332 id                      id
3333 Locker locker            name
3334 username
3335 -----
3336

```

```

3337 @Entity
3338 public class Member {
3339     @Id @GeneratedValue
3340     @Column(name = "MEMBER_ID")
3341     private Long id;
3342
3343     private String username;
3344
3345     @OneToOne
3346     @JoinColumn(name = "LOCKER_ID")
3347     private Locker locker;
3348 }
3349

```

```

3350 @Entity
3351 public class Locker {
3352     @Id @GeneratedValue
3353     @Column(name = "LOCKER_ID")
3354     private Long id;
3355
3356     private String name;
3357 }
3358

```

3359 -양방향

- 3360 --양방향의 주인이 정해 졌다.
- 3361 --Member table의 외래 key를 가지고 있으며 Member.locker가 연관관계의 주인이다.
- 3362 --Locker는 mappedBy를 이용해서 연관관계의 주인을 선언한다.

```

3364 ----- locker -----
3365 Member -----> Locker
3366 ----- 1 -----
3367 id          1          id

```

```

3368 Locker locker    <- - - - - name
3369 username        member    Member member
3370 -----

```

```

3371
3372 @Entity
3373 public class Member {
3374     @Id @GeneratedValue
3375     @Column(name = "MEMBER_ID")
3376     private Long id;
3377
3378     private String username;
3379
3380     @OneToOne
3381     @JoinColumn(name = "LOCKER_ID")
3382     private Locker locker;
3383 }
3384

```

```

3385 @Entity
3386 public class Locker {
3387     @Id @GeneratedValue
3388     @Column(name = "LOCKER_ID")
3389     private Long id;
3390
3391     private String name;
3392
3393     @OneToOne(mappedBy = "locker")
3394     private Member member;
3395 }
3396

```

```

3397 -대상 table에 외래 key
3398 --JPA에서는 주 table에 외래 key가 있는것이 좀더 편리하게 매핑할 수 있게된다.
3399 --단방향
3400     ---이는 JPA에서 지원 하지 않는다.
3401 --양방향
3402     ---대상 table인 Locker에서 외래 key를 관리하는 모습이다.
3403

```

```

3404 ----- locker -----
3405 Member    - - - - - > Locker
3406 ----- 1 -----
3407 id        1 id
3408 Locker locker <------ name
3409 username  member Member member
3410 -----

```

```

3411
3412 @Entity
3413 public class Member {
3414     @Id @GeneratedValue
3415     @Column(name = "MEMBER_ID")
3416     private Long id;
3417
3418     private String username;
3419
3420     @OneToOne(mappedBy = "member")
3421     private Locker locker;
3422 }
3423

```

```

3424 @Entity
3425 public class Locker {
3426     @Id @GeneratedValue
3427     @Column(name = "LOCKER_ID")
3428     private Long id;
3429
3430     private String name;
3431
3432     @OneToOne
3433     @JoinColumn(name = "MEMBER_ID")
3434     private Member member;

```

```
}
```

4)다대다

- 다대다는 한쪽에 외래 key를 두고 2개의 table로 관계를 표현 할 수가 없어서 중간에 연결 table을 추가 하게 된다.
- 하지만 객체의 관계는 2개의 객체로 표현이 가능하다.
- 다대다 관계인 "Member"와 "Product"로 살펴보자.
- 단방향
 - Member에서 Product를 @ManyToMany로 연결한다.
 - 중요한 점은 @JoinTable을 이용해서 연결 table을 설정해 주는 것이다.
 - MEMBER_PRODUCT table은 다대다 관계를 "일대다, 다대일" 관계로 풀어내기 위한 연결 table이다.

```
@Entity
```

```
public class Member {
```

```
    @Id @GeneratedValue
```

```
    @Column(name = "MEMBER_ID")
```

```
    private Long id;
```

```
    private String username;
```

```
    @ManyToMany
```

```
    @JoinTable(name = MEMBER_PRODUCT,
```

```
    joinColumns = @JoinColumn(name = "MEMBER_ID"),
```

```
    inverseJoinColumns = @JoinColumn(name = "PRODUCT_ID")
```

```
    private List<Product> products = new ArrayList<Product>();
```

```
}
```

```
@Entity
```

```
public class Product {
```

```
    @Id @GeneratedValue
```

```
    @Column(name = "PRODUCT_ID")
```

```
    private Long id;
```

```
    private String name;
```

```
}
```

```
public void save(){
```

```
    Product productA = new Product();
```

```
    productA.setId("ProductA");
```

```
    productA.setName("상품A");
```

```
    em.persist(productA);
```

```
    Member member1 = new Member();
```

```
    member1.setId("member1");
```

```
    member1.setUsername("회원1");
```

```
    member1.getProducts().add(ProductA); // 연관관계 설정
```

```
    em.persist(member1);
```

```
}
```

```
public void find(){
```

```
    Member member = em.find(Member.class, "member1");
```

```
    List<Product> products = member.getProducts();
```

```
    for(Product product : products){
```

```
        System.out.println("product.name = " + product.getName());
```

```
    }
```

```
}
```

-양방향

--단방향과 마찬가지로 양방향은 반대쪽도 @ManyToMany를 사용한다.

--그리고 양방향중 한곳에 mappedBy를 설정해서 연관관계 주인을 지정해 준다.

--mappedBy가 없는 쪽이 연관관계의 주인이다.

```
@Entity
```

```
public class Product {
```

```
    @Id
```

```
    private String id;
```

```
3502         @ManyToMany(mappedBy = "products")
3503         private List<Member> members;
3504
3505     ...
3506 }
3507
3508
```

3509 14. Lab : JavaSE 환경에서 JPA 설정 및 CRUD

3510 1)Project 구조

- 3511 -DB는 편의상 h2 DB 사용
- 3512 -tbl_user, tbl_order table을 예제로 사용

3513

3514 2)SQL

```
3515 CREATE TABLE tbl_user
3516 (
3517     user_id INT,
3518     username VARCHAR(20),
3519     nick_name VARCHAR(20),
3520     address VARCHAR(100),
3521     CONSTRAINT tbl_user_user_id_pk PRIMARY KEY(user_id)
3522 );
3523
3524 CREATE TABLE tbl_order
3525 (
3526     order_id INT,
3527     user_id INT,
3528     order_name VARCHAR(45),
3529     note VARCHAR(100),
3530     price INT,
3531     CONSTRAINT tbl_order_order_id_pk PRIMARY KEY(order_id),
3532     CONSTRAINT tbl_order_user_id_fk FOREIGN KEY(user_id)
3533     REFERENCES tbl_user(user_id)
3534 );
3535
```

3536 3)Eclipse, JPA J2SE Project

- 3537 -Package Explorer > right-click > New > Other > JPA > JPA Project > Next
- 3538 -Project name : JPADemo
- 3539 -Target runtime : jdk1.8.0_162
- 3540 -JPA version : 2.1
- 3541 -Configuration : Default Configuration for jdk1.8.0_162
- 3542 -Next > Next
- 3543 -Platform :Generic 2.1
- 3544 -Type : User Library > Download library...
- 3545 -Download Library : EclipseLink 2.5.2 > Next > Check I accpet... > Finish
- 3546 -Connection : Add connection... > Generic JDBC
- 3547 -Name : H2 JDBC > Next > New Driver Definition
- 3548 -Name/Type tab > Select Generic JDBC Driver > JAR List tab > Add JAR/Zip > C:\Program Files (x86)\H2\bin\h2-1.4.197.jar
- 3549 -Properties tab >
 - 3550 --Connection URL :jdbc:h2:tcp://localhost/~/test
 - 3551 --Database Name : test
 - 3552 --Driver Class : Click ... > Select [Browse for class] > org.h2.Driver
 - 3553 --User ID as sa
- 3554 -Test Connection > Finish
- 3555 -Finish
- 3556 -Open Perspective

3557

3558 4)Data Source Explorer view에 H2 database 등록할 경우

- 3559 -<https://ibytecode.com/blog/eclipse-dtp-configure-h2-datasource-using-data-source-explorer/>
- 3560 -Right click on Database Connections in Data Source Explorer -> New
- 3561 -From the main menu, select File -> New -> Other. Under Connection Profiles, select Connection Profile and click Next.
- 3562 -Select Generic JDBC from Connection Profile Types
 - 3563 --Name : H2 JDBC
 - 3564 --Click Next.
- 3565 -Now select an existing JDBC Driver and provide Connection Details in the "New Connection Profile" dialog.

-If this is the first installation of H2 Database Profile in your Eclipse workspace, you have to create a new driver definition by providing the location of the driver JAR and connection properties.

- Click on the New Driver Definition icon next to the Drivers combo box.
- Select driver version under Name/Type tab.
- Select [Generic JDBC Driver]
- Click on JAR list tab and add the location of the H2 database JAR file.
C:\Program Files (x86)\H2\bin\h2-1.4.197.jar
- Click on Properties tab and enter the properties and click on OK.
- Connection URL : jdbc:h2:tcp://localhost/~ /test
- Database Name : test
- Driver Class : Click ... > Select [Browse for class] > org.h2.Driver
- User ID as sa
(empty password) and click on OK.
- Click [Test Connection]
- Click Finish to create the connection profile.

5)H2Database Driver를 project build path에 등록

6)Entity(Domain)

- JPA에서 Entity는 하나의 테이블 객체를 표현한 것이라고 생각해 된다.
- @Entity가 테이블 정보이며 변수가 필드가 되는 것이다.

- src/com.javasoft.entity package
- com.javasoft.entity > right-click > New > Class
- Class name : User > Finish
- User.java

```
package com.javasoft.entity;
```

```
import java.io.Serializable;  
import java.util.List;
```

```
import javax.persistence.CascadeType;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.OneToMany;  
import javax.persistence.Table;
```

```
@Entity
```

```
@Table(name = "tbl_user")
```

```
public class User implements Serializable {
```

```
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private Integer user_id;
```

```
    private String username;
```

```
    private String nick_name;
```

```
    private String address;
```

```
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
```

```
    private List<Order> orders;
```

```
    public User() {}
```

```
    public User(Integer user_id, String username, String nick_name, String address) {
```

```
        this.user_id = user_id;
```

```
        this.username = username;
```

```
        this.nick_name = nick_name;
```

```
        this.address = address;
```

```
    }
```

```
    public Integer getUser_id() {
```

```
        return user_id;
```

```
    }
```

```
    public void setUser_id(Integer user_id) {
```

```

3631         this.user_id = user_id;
3632     }
3633
3634     public String getUsername() {
3635         return username;
3636     }
3637
3638     public void setUsername(String username) {
3639         this.username = username;
3640     }
3641
3642     public String getNick_name() {
3643         return nick_name;
3644     }
3645
3646     public void setNick_name(String nick_name) {
3647         this.nick_name = nick_name;
3648     }
3649
3650     public String getAddress() {
3651         return address;
3652     }
3653
3654     public void setAddress(String address) {
3655         this.address = address;
3656     }
3657
3658     public List<Order> getOrders() {
3659         return orders;
3660     }
3661
3662     public void setOrders(List<Order> orders) {
3663         this.orders = orders;
3664     }
3665
3666     public int totalPrice() {
3667         int totalPrice = 0;
3668         for (Order order : orders) {
3669             totalPrice += order.getPrice();
3670         }
3671         return totalPrice;
3672     }
3673
3674     @Override
3675     public int hashCode() {
3676         // TODO Auto-generated method stub
3677         return super.hashCode();
3678     }
3679
3680     @Override
3681     public boolean equals(Object obj) {
3682         // TODO Auto-generated method stub
3683         return super.equals(obj);
3684     }
3685 }

```

3688
3689 -com.javasoft.entity.Order.java

```

3690
3691     package com.javasoft.entity;
3692
3693     import javax.persistence.CascadeType;
3694     import javax.persistence.Entity;
3695     import javax.persistence.FetchType;
3696     import javax.persistence.GeneratedValue;
3697     import javax.persistence.GenerationType;

```

```

3698 import javax.persistence.Id;
3699 import javax.persistence.IdClass;
3700 import javax.persistence.JoinColumn;
3701 import javax.persistence.ManyToOne;
3702 import javax.persistence.Table;
3703
3704 @Entity
3705 @Table(name = "tbl_order")
3706 public class Order {
3707     @Id
3708     @GeneratedValue(strategy = GenerationType.AUTO)
3709     private Integer order_id;
3710     private String order_name;
3711     private String note;
3712     private int price;
3713
3714     @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
3715     @JoinColumn(name = "user_id")
3716     private User user;
3717
3718     public Order() {}
3719
3720     public Order(String order_name, String note, int price, User user) {
3721         this.order_name = order_name;
3722         this.note = note;
3723         this.price = price;
3724         this.user = user;
3725     }
3726
3727     public Integer getOrder_id() {
3728         return order_id;
3729     }
3730
3731     public void setOrder_id(Integer order_id) {
3732         this.order_id = order_id;
3733     }
3734
3735     public String getOrder_name() {
3736         return order_name;
3737     }
3738
3739     public void setOrder_name(String order_name) {
3740         this.order_name = order_name;
3741     }
3742
3743     public String getNote() {
3744         return note;
3745     }
3746
3747     public void setNote(String note) {
3748         this.note = note;
3749     }
3750
3751     public int getPrice() {
3752         return price;
3753     }
3754
3755     public void setPrice(int price) {
3756         this.price = price;
3757     }
3758
3759     @Override
3760     public String toString() {
3761         return "Order{" + "orderId=" + order_id + ", orderName='" + order_name + "\" + ",
3762             note='" + note + '\'' + "\n";
3763     }

```

7)src/META-INF/persistence.xml

```
-hibernate.connection.driver_class : DB Driver
-hibernate.connection.url : DB url 및 DB파일이 저장될 경로(h2 DB에 한함)
-hibernate.connection.user : username
-hibernate.show_sql : JPA 내부적으로 사용되는 쿼리를 log로 나타낼지 설정
-hibernate.hbm2ddl.auto : Entity에 의한 테이블 설정 (create-drop은 프로젝트 실행시 기존 테이블을 삭제하고
다시 생성한다. 즉 테스트하기 위한 초기화)
-구현체와 DB 종류별 설정 참고(https://gist.github.com/mortezaadi/8619433)
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

  <persistence-unit name="JPADemo" transaction-type="RESOURCE_LOCAL">
    <class>com.javasoft.entity.User</class>
    <class>com.javasoft.entity.Order</class>

    <properties>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:h2:tcp://localhost/~/test" />
      <property name="javax.persistence.jdbc.driver"
        value="org.h2.Driver" />
      <property name="javax.persistence.jdbc.user" value="sa" />
      <!-- <property name="javax.persistence.jdbc.password" value=""/> -->
      <property name="eclipselink.logging.level" value="FINE" />
      <property name="eclipselink.ddl-generation" value="create-tables" />
    </properties>
  </persistence-unit>
</persistence>
```

8)JPACRUDTest.java

```
-com.javasoft.service package 생성
-com.javasoft.service > right-click > New > Other > Java > JUnit > JUnit Test Case
-Select New JUnit 4 test
-Name : JPACRUDTest > Next >
-Select Perform the following action : Add JUnit 4 library to the build path > OK

-EntityManager : Entity의 Lifecycle과 persistence context, transaction을 관리한다.
-즉 insert, update, delete, select를 할 수 있다.
  --select : find(Class, Object);
  --insert : persist(Object);
  --update : merge(T);
  --delete : remove(Object);
-EntityManagerFactory : EntityManager를 생성하기 위한 클래스이며 persistence.xml 설정에 기반한다.

package com.javasoft.test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNull;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import com.javasoft.entity.Order;
```

```

3830 import com.javasoft.entity.User;
3831
3832 public class JPACRUDTest {
3833     private EntityManager entityManager;
3834     private EntityManagerFactory entityManagerFactory;
3835     private User user;
3836
3837     @Before
3838     public void setUp() throws Exception {
3839         entityManagerFactory = Persistence.createEntityManagerFactory("JPADemo");
3840         entityManager = entityManagerFactory.createEntityManager();
3841         entityManager.getTransaction().begin();
3842
3843         // fixture
3844         user = new User();
3845         user.setUsername("한지민");
3846         user.setNick_name("jimin");
3847         user.setAddress("서울");
3848         List<Order> orders = new ArrayList<>();
3849         for (int i = 0; i < 10; i++) {
3850             Order order = new Order("order" + i, "note" + i, i + 10, user);
3851             entityManager.persist(order);
3852             orders.add(order);
3853         }
3854         user.setOrders(orders);
3855         entityManager.persist(user);
3856         System.out.println("===== fixture =====\n" + user);
3857     }
3858
3859     @After
3860     public void after() {
3861         entityManager.getTransaction().commit();
3862         entityManager.close();
3863         entityManagerFactory.close();
3864     }
3865
3866     @Test
3867     public void select() {
3868         User findUser = entityManager.find(User.class, user.getUser_id());
3869         assertEquals(user.getUser_id(), findUser.getUser_id());
3870         assertEquals(user.getUsername(), findUser.getUsername());
3871         assertEquals(user.getNick_name(), findUser.getNick_name());
3872         assertEquals(user.getAddress(), findUser.getAddress());
3873         assertEquals(user.totalPrice(), 145);
3874
3875         // order
3876         assertEquals(user.getOrders().size(), 10);
3877     }
3878
3879     @Test
3880     public void update() {
3881         // update
3882         User updateUser = entityManager.find(User.class, user.getUser_id());
3883         updateUser.setNick_name("update nickName");
3884         updateUser.setAddress("update address");
3885
3886         entityManager.merge(updateUser);
3887
3888         // persistence Context Test
3889         assertEquals("update nickName", user.getNick_name());
3890         assertEquals("update address", user.getAddress());
3891
3892         // update Tests
3893         assertEquals("update nickName", updateUser.getNick_name());
3894         assertEquals("update address", updateUser.getAddress());
3895     }
3896

```

```

3897         @Test
3898         public void delete() {
3899             User getUser = entityManager.find(User.class, user.getUser_id());
3900             entityManager.remove(getUser);
3901             User deleteUser = entityManager.find(User.class, user.getUser_id());
3902             assertNull(deleteUser);
3903         }
3904     }
3905
3906

```

15. Lab : OneToOne Demo

1)Project 구조

- DB는 편의상 h2 DB 사용
- person, cellular table을 예제로 사용

2)SQL

```

3913 CREATE TABLE Person
3914 (
3915     id INT,
3916     cellular_id INT,
3917     name VARCHAR(45),
3918     CONSTRAINT person_id_pk PRIMARY KEY(id)
3919 );
3920
3921 CREATE TABLE Cellular
3922 (
3923     id INT,
3924     tel_number CHAR(13),
3925     CONSTRAINT cellular_id_pk PRIMARY KEY(id)
3926 );
3927

```

3)Eclipse, JPA J2SE Project

- Package Explorer > right-click > New > Other > JPA > JPA Project > Next
- Project name : OneToOneDemo
- Target runtime : jdk1.8.0_162
- JPA version : 2.1
- Configuration : Default Configuration for jdk1.8.0_162
- Next > Next
- Platform :Generic 2.1
- Type : User Library > Download library...
- Download Library : EclipseLink 2.5.2 > Next > Check I accpet... > Finish
- Connection : Add connection... > Generic JDBC
- Name : H2 JDBC > Next > New Driver Definition
- Name/Type tab > Select Generic JDBC Driver > JAR List tab > Add JAR/Zip > C:\Program Files (x86)\H2\bin\h2-1.4.197.jar
- Properties tab >
 - Connection URL :jdbc:h2:tcp://localhost/~/test
 - Database Name : test
 - Driver Class : Click ... > Select [Browse for class] > org.h2.Driver
 - User ID as sa
- Test Connection > Finish
- Finish
- Open Perspective

4)Data Source Explorer view에 H2 database 등록할 경우

- <https://ibytecode.com/blog/eclipse-dtp-configure-h2-datasource-using-data-source-explorer/>
- Right click on Database Connections in Data Source Explorer -> New
- From the main menu, select File -> New -> Other. Under Connection Profiles, select Connection Profile and click Next.
- Select Generic JDBC from Connection Profile Types
 - Name : H2 JDBC
 - Click Next.
- Now select an existing JDBC Driver and provide Connection Details in the "New Connection Profile" dialog.
- If this is the first installation of H2 Database Profile in your Eclipse workspace, you have to create a new driver definition by providing the location of the driver JAR and connection properties.

--Click on the New Driver Definition icon next to the Drivers combo box.
--Select driver version under Name/Type tab.
--Select [Generic JDBC Driver]
--Click on JAR list tab and add the location of the H2 database JAR file.
C:\Program Files (x86)\H2\bin\h2-1.4.197.jar
--Click on Properties tab and enter the properties and click on OK.
--Connection URL :jdbc:h2:tcp://localhost/~ /test
--Database Name : test
--Driver Class : Click ... > Select [Browse for class] > org.h2.Driver
--User ID as sa
(empty password) and click on OK.
--Click [Test Connection]
-Click Finish to create the connection profile.

5)H2Database Driver를 project build path에 등록

6)Entity(Domain)

-src/com.javasoft.entity package
-com.javasoft.entity > right-click > New > Class
-Class name : Person > Finish
-Person.java

```
package com.javasoft.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;

@Entity
public class Person {
    @Id
    @GeneratedValue
    private int id;

    private String name;

    @OneToOne
    @JoinColumn(name = "cellular_id")
    private Cellular cellular;

    public Person() {}

    public Person(String name, Cellular cellular) {
        this.name = name;
        this.cellular = cellular;
    }

    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;
    }

    public Cellular getCellular() {
        return cellular;
    }
```

```

4026     }
4027
4028     public void setCellular(Cellular cellular) {
4029         this.cellular = cellular;
4030     }
4031
4032     @Override
4033     public String toString() {
4034         return "Person{" +
4035             "id=" + id +
4036             ", name=" + name + "\" +
4037             ", cellular=" + cellular +
4038             "'}";
4039     }
4040 }

```

4042 -com.javasoft.entity.Cellular.java

```

4045     package com.javasoft.entity;
4046
4047     import javax.persistence.Column;
4048     import javax.persistence.Entity;
4049     import javax.persistence.GeneratedValue;
4050     import javax.persistence.Id;
4051
4052     @Entity
4053     public class Cellular {
4054         @Id
4055         @GeneratedValue
4056         private int id;
4057
4058         @Column(name="tel_number")
4059         private String number;
4060
4061         public Cellular() {}
4062
4063         public Cellular(String number) {
4064             this.number = number;
4065         }
4066
4067         public int getId() {
4068             return id;
4069         }
4070
4071         public void setId(int id) {
4072             this.id = id;
4073         }
4074
4075         public String getNumber() {
4076             return number;
4077         }
4078
4079         public void setNumber(String number) {
4080             this.number = number;
4081         }
4082
4083         @Override
4084         public String toString() {
4085             return "Cellular{" +
4086                 "id=" + id +
4087                 ", number=" + number +
4088                 "'}";
4089         }
4090     }

```


7)src/META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="OneToOneDemo" transaction-type="RESOURCE_LOCAL">

    <class>com.javasoft.entity.Person</class>
    <class>com.javasoft.entity.Cellular</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:h2:tcp://localhost/~/test" />
      <property name="javax.persistence.jdbc.driver"
        value="org.h2.Driver" />
      <property name="javax.persistence.jdbc.user" value="sa" />
      <!-- <property name="javax.persistence.jdbc.password" value=""/> -->
      <property name="eclipselink.logging.level" value="FINE" />
      <property name="eclipselink.ddl-generation" value="create-tables" />
    </properties>
  </persistence-unit>
</persistence>
```

8)OneToOneTest.java

```
-com.javasoft.service package 생성
-com.javasoft.service > right-click > New > Other > Java > JUnit > JUnit Test Case
-Select New JUnit 4 test
-Name : OneToOneTest > Next >
-Select Perform the following action : Add JUnit 4 library to the build path > OK
```

```
package com.javasoft.test;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import com.javasoft.entity.Cellular;
import com.javasoft.entity.Person;

public class OneToOneTest {
    private EntityManagerFactory entityManagerFactory;
    private EntityManager entityManager;

    @Test
    public void oneToOneTest() {
        Cellular cellular = new Cellular();
        cellular.setNumber("010-1234-5678");
        entityManager.persist(cellular);

        Person person = new Person();
        person.setName("한지민");
        person.setCellular(cellular);
        entityManager.persist(person);

        Assert.assertEquals(person.getCellular().getId(), cellular.getId());
    }

    @Before
    public void setUp() throws Exception {
```

```

4157         entityManagerFactory = Persistence.createEntityManagerFactory("OneToOneDemo");
4158         entityManager = entityManagerFactory.createEntityManager();
4159         entityManager.getTransaction().begin();
4160     }
4161
4162     @After
4163     public void after() {
4164         entityManager.getTransaction().commit();
4165         entityManager.close();
4166     }
4167 }

```

16. Lab : ManyToOne Demo

1)Project 구조

- DB는 편의상 h2 DB 사용
- tbl_User, tbl_Order table을 예제로 사용

2)SQL

```

4177 CREATE TABLE tbl_user
4178 (
4179     user_id INT,
4180     username VARCHAR(20),
4181     nick_name VARCHAR(20),
4182     address VARCHAR(100),
4183     CONSTRAINT tbl_user_user_id_pk PRIMARY KEY(user_id)
4184 );
4185
4186 CREATE TABLE tbl_order
4187 (
4188     order_id INT,
4189     user_id INT,
4190     order_name VARCHAR(45),
4191     note VARCHAR(100),
4192     price INT,
4193     CONSTRAINT tbl_order_order_id_pk PRIMARY KEY(order_id),
4194     CONSTRAINT tbl_order_user_id_fk FOREIGN KEY(user_id)
4195     REFERENCES tbl_user(user_id)
4196 );

```

3)Eclipse, JPA J2SE Project

- Package Explorer > right-click > New > Other > JPA > JPA Project > Next
- Project name : ManyToOneDemo
- Target runtime : jdk1.8.0_162
- JPA version : 2.1
- Configuration : Default Configuration for jdk1.8.0_162
- Next > Next
- Platform :Generic 2.1
- Type : User Library > Download library...
- Download Library : EclipseLink 2.5.2 > Next > Check I accpet... > Finish
- Open Perspective

4)H2Database Driver를 project build path에 등록

5)Entity(Domain)

- src/com.javasoft.entity package
- com.javasoft.entity > right-click > New > Class
- Class name : User > Finish
- User.java

```

4218     package com.javasoft.entity;
4219
4220     import java.util.ArrayList;
4221     import java.util.List;
4222
4223     import javax.persistence.CascadeType;

```

```
4224 import javax.persistence.Entity;
4225 import javax.persistence.GeneratedValue;
4226 import javax.persistence.GenerationType;
4227 import javax.persistence.Id;
4228 import javax.persistence.OneToMany;
4229 import javax.persistence.Table;
4230
4231 @Entity
4232 @Table(name = "tbl_user")
4233 public class User {
4234     @Id @GeneratedValue(strategy = GenerationType.AUTO)
4235     private Integer user_id;
4236     private String username;
4237     private String nick_name;
4238     private String address;
4239
4240     @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
4241     private List<Order> orders;
4242
4243     public User() {}
4244
4245     public User(Integer user_id, String username, String nick_name, String address) {
4246         this.user_id = user_id;
4247         this.username = username;
4248         this.nick_name = nick_name;
4249         this.address = address;
4250     }
4251
4252     public Integer getUser_id() {
4253         return user_id;
4254     }
4255
4256     public void setUser_id(Integer user_id) {
4257         this.user_id = user_id;
4258     }
4259
4260     public String getUsername() {
4261         return username;
4262     }
4263
4264     public void setUsername(String username) {
4265         this.username = username;
4266     }
4267
4268     public String getNick_name() {
4269         return nick_name;
4270     }
4271
4272     public void setNick_name(String nick_name) {
4273         this.nick_name = nick_name;
4274     }
4275
4276     public String getAddress() {
4277         return address;
4278     }
4279
4280     public void setAddress(String address) {
4281         this.address = address;
4282     }
4283
4284     public List<Order> getOrders() {
4285         return orders;
4286     }
4287
4288     public void setOrders(List<Order> orders) {
4289         this.orders = orders;
4290     }
4291 }
```

```

4291
4292     public boolean addOrder(Order order) {
4293         if(orders == null)
4294             orders = new ArrayList<>();
4295
4296         return this.orders.add(order);
4297     }
4298
4299     @Override
4300     public String toString() {
4301         return "User{" +
4302             "userId=" + this.user_id +
4303             ", username=" + this.username + "\" +
4304             ", nickName=" + this.nick_name + "\" +
4305             ", address=" + this.address + "\" +
4306             ", orders=" + orders +
4307             '}'';
4308     }
4309 }

```

-com.javasoft.entity.Order.java

```

4313     package com.javasoft.entity;
4314
4315     import javax.persistence.CascadeType;
4316     import javax.persistence.Entity;
4317     import javax.persistence.FetchType;
4318     import javax.persistence.GeneratedValue;
4319     import javax.persistence.GenerationType;
4320     import javax.persistence.Id;
4321     import javax.persistence.IdClass;
4322     import javax.persistence.JoinColumn;
4323     import javax.persistence.ManyToOne;
4324     import javax.persistence.Table;
4325
4326     @Entity
4327     @Table(name = "tbl_order")
4328     public class Order {
4329         @Id
4330         @GeneratedValue(strategy = GenerationType.AUTO)
4331         private Integer order_id;
4332         private String order_name;
4333         private String note;
4334         private int price;
4335
4336         @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
4337         @JoinColumn(name = "user_id")
4338         private User user;
4339
4340         public Order() {}
4341
4342         public Order(String order_name, String note, int price, User user) {
4343             this.order_name = order_name;
4344             this.note = note;
4345             this.price = price;
4346             this.user = user;
4347         }
4348
4349         public Integer getOrder_id() {
4350             return order_id;
4351         }
4352
4353         public void setOrder_id(Integer order_id) {
4354             this.order_id = order_id;
4355         }
4356     }
4357

```

```

4358     public String getOrder_name() {
4359         return order_name;
4360     }
4361
4362     public void setOrder_name(String order_name) {
4363         this.order_name = order_name;
4364     }
4365
4366     public String getNote() {
4367         return note;
4368     }
4369
4370     public void setNote(String note) {
4371         this.note = note;
4372     }
4373
4374     public int getPrice() {
4375         return price;
4376     }
4377
4378     public void setPrice(int price) {
4379         this.price = price;
4380     }
4381
4382     public User getUser() {
4383         return user;
4384     }
4385
4386     public void setUser(User user) {
4387         this.user = user;
4388     }
4389
4390     @Override
4391     public String toString() {
4392         return "Order{" + "orderId=" + order_id + ", orderName=" + order_name + "\" + ",
         note=" + note + '}' + "\n";
4393     }
4394 }
4395
4396

```

6)src/META-INF/persistence.xml

```

4397
4398
4399     <?xml version="1.0" encoding="UTF-8"?>
4400     <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="
4401         http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
4402         http://xmlns.jcp.org/xml/ns/persistence
4403         http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
4404         <persistence-unit name="ManyToOneDemo" transaction-type="RESOURCE_LOCAL">
4405
4406             <class>com.javasoft.entity.User</class>
4407             <class>com.javasoft.entity.Order</class>
4408             <properties>
4409                 <property name="javax.persistence.jdbc.url"
4410                     value="jdbc:h2:tcp://localhost/~/test" />
4411                 <property name="javax.persistence.jdbc.driver"
4412                     value="org.h2.Driver" />
4413                 <property name="javax.persistence.jdbc.user" value="sa" />
4414                 <!-- <property name="javax.persistence.jdbc.password" value=""/> -->
4415                 <property name="eclipselink.logging.level" value="FINE" />
4416                 <property name="eclipselink.ddl-generation" value="create-tables" />
4417             </properties>
4418         </persistence-unit>
4419     </persistence>
4420

```

7)ManyToOneTest.java

```

4420     -com.javasoft.service package 생성

```

-com.javasoft.service > right-click > New > Other > Java > JUnit > JUnit Test Case
-Select New JUnit 4 test
-Name : ManyToOneTest > Next >
-Select Perform the following action : Add JUnit 4 library to the build path > OK

```
package com.javasoft.test;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import com.javasoft.entity.Order;
import com.javasoft.entity.User;

public class ManyToOneTest {
    private EntityManagerFactory entityManagerFactory;
    private EntityManager entityManager;

    @Test
    public void oneToManyAndManyToOneTest() {
        Order order = new Order();
        order.setOrder_name("test order");
        order.setPrice(123);
        order.setNote("test note");
        User user = new User();
        user.setUsername("한지민");
        user.setNick_name("jimin");
        user.setAddress("서울");

        // relationship
        user.addOrder(order);
        order.setUser(user);
        entityManager.persist(user);

        Assert.assertEquals(user.getOrders().get(0).getOrder_id(), order.getOrder_id());
    }

    @Before
    public void setUp() throws Exception {
        entityManagerFactory = Persistence.createEntityManagerFactory("ManyToOneDemo");
        entityManager = entityManagerFactory.createEntityManager();
        entityManager.getTransaction().begin();
    }

    @After
    public void after() {
        entityManager.getTransaction().commit();
        entityManager.close();
    }
}
```