Hibernate & JPA Introduction

JDBC Disadvantages

- 1. Need to write DB dependent SQL queries.
- 2. Need to create tables and manage relationship.
- 3. Need to manage exception handling, connection pooling etc.

ORM(Object Relational Mapping)

JAVA TYPE	DATABASE TYPE
CLASS	TABLE
DATA MEMBER	COLUMN
ID FIELD	PRIMARY KEY
CLASS INSTANCE VARIABLE	DATABASE TABLE RECORD

Frameworks which implemented ORM concepts such as Hibernate, JPA, Toplink, Ibatis etc.

Hibernate

- 1. Open Source and Light Weight
- 2. ORM framework
- 3. Supports Build-in JDBC connection pool and support 3rd party also.
- 4. Allows Direct Sql/Native queries
- 5. Supports relationship between objects
- 6. Supports in-built caching

Hibernate Advantage:

- 1. Automatically creates the tables and SQL queries.
- 2. Automatically generate the JDBC connectivity code internally.

Terminology in ORM

1. **Entity** : Object which needs to be persisted.
2. **EntityManager** : Interface which allows to make connection and manage entities.
3. **PersistentContext**: Place where entities are managed
4. **Managed Entity** : Entity associated with a Persistent Context, detect and sync with DB.
5. **Unmanaged Entity** : Entity not associated with a Persistent Context.

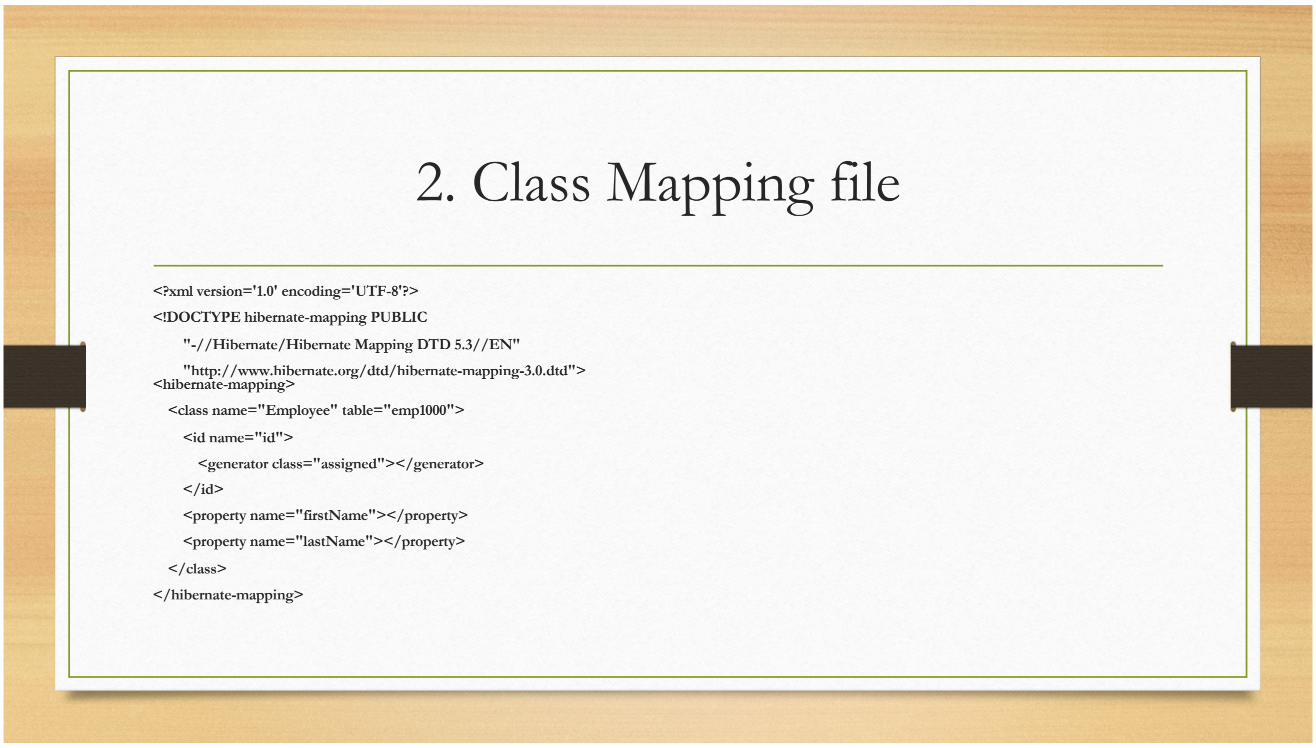
Hibernate Steps:

- Need to create 4 file to create a Hibernate Project.

1. Pojo Class
2. Class Mapping file
3. Hibernate Configuration file
4. Client code to activate Hibernate framework

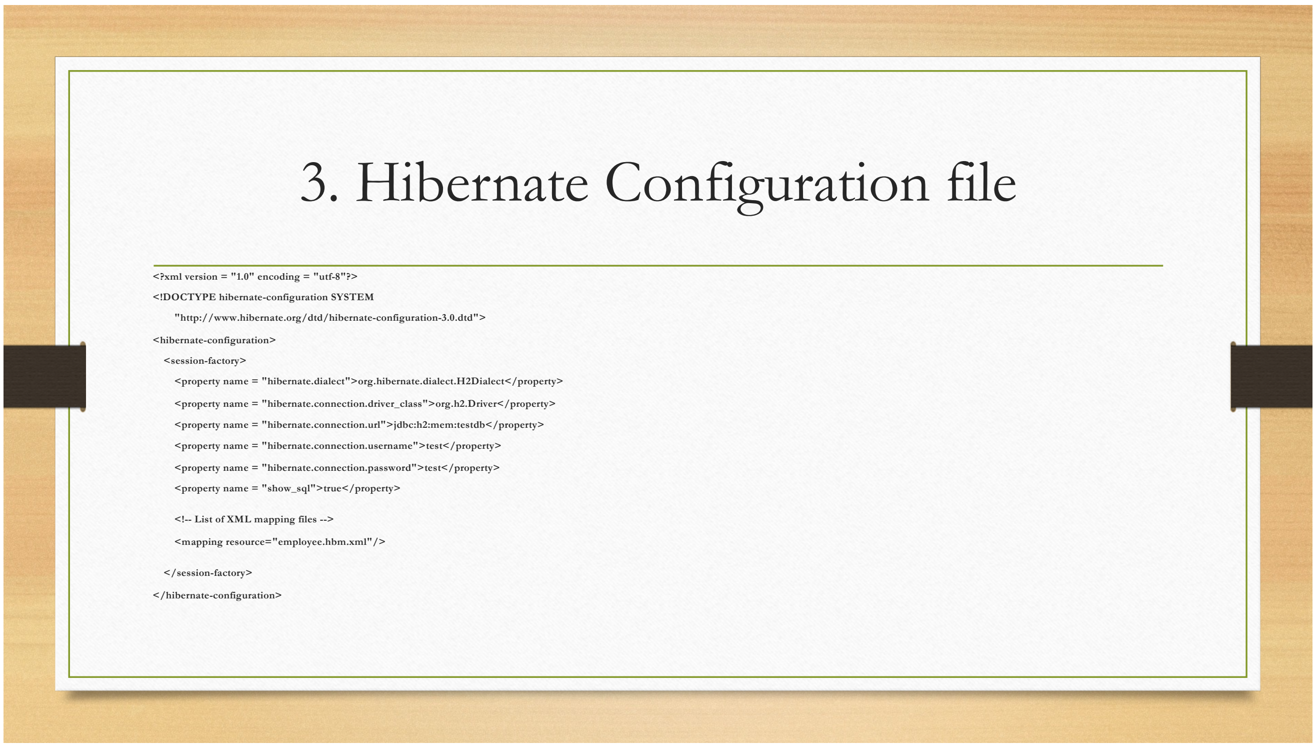
1. Pojo Class

```
public class Employee {
   private int id;
   private String firstName,lastName;
     //Getter & Setters
 }
```



2. Class Mapping file

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Employee" table="emp1000">
      <id name="id">
         <generator class="assigned"></generator>
      </id>
      <property name="firstName"></property>
      <property name="lastName"></property>
   </class>
</hibernate-mapping>
```



3. Hibernate Configuration file

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <property name = "hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
      <property name = "hibernate.connection.driver_class">org.h2.Driver</property>
      <property name = "hibernate.connection.url">jdbc:h2:mem:testdb</property>
      <property name = "hibernate.connection.username">test</property>
      <property name = "hibernate.connection.password">test</property>
      <property name = "show_sql">true</property>

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

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

4. Client code to activate Hibernate

```
import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import java.io.File;


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

   //public static void main(String [] args){

      SessionFactory factory;

      try {

         factory = new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();

      } catch (Throwable ex) {}
      Session session = factory.openSession();

      Transaction tx = session.beginTransaction();
      Employee e1=new Employee();

      e1.setId(1);

      e1.setFirstName("Test");

      e1.setLastName("User");

      session.save(e1);

      tx.commit();

      session.close();

      factory.close();

   }

}
```

Entity Object States

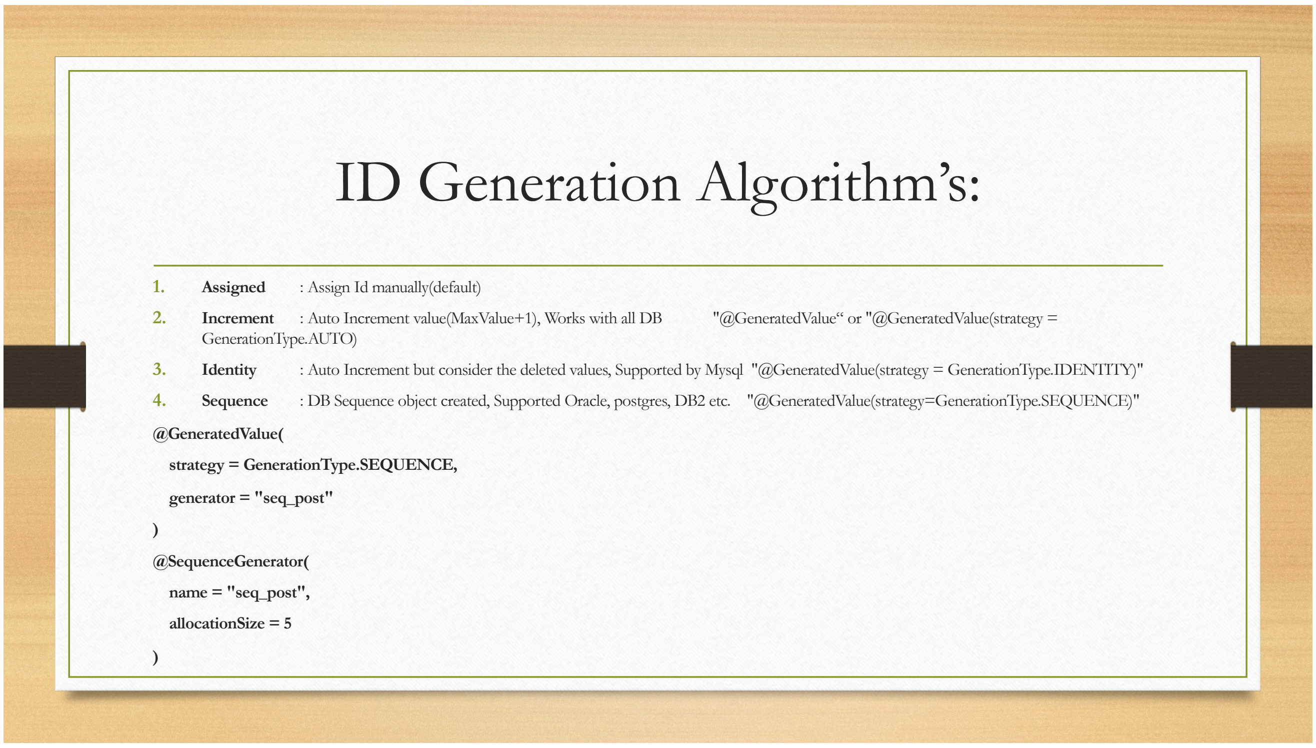
1. **Transient State** : When object is created. Do not have id and do not represent Db table record.
2. **Persistent State** : When object is created. It have id assigned and represent Db table record(with Sync with Db). Stored in perstintence context.
   Ex: save(), persist(), update(), merge(), get(), load() etc.
3. **Detached State** : When object is created. Have id but do not represent Db table record. When session is closed or record is deleted from Db. Ex: delete(), evict(), clear()

JPA

- Standard Specification which gave set of Interfaces implemented by standard ORM frameworks like Hibernate, Spring Data, Toplinks etc.

JPA Annotations(javax.persistence)

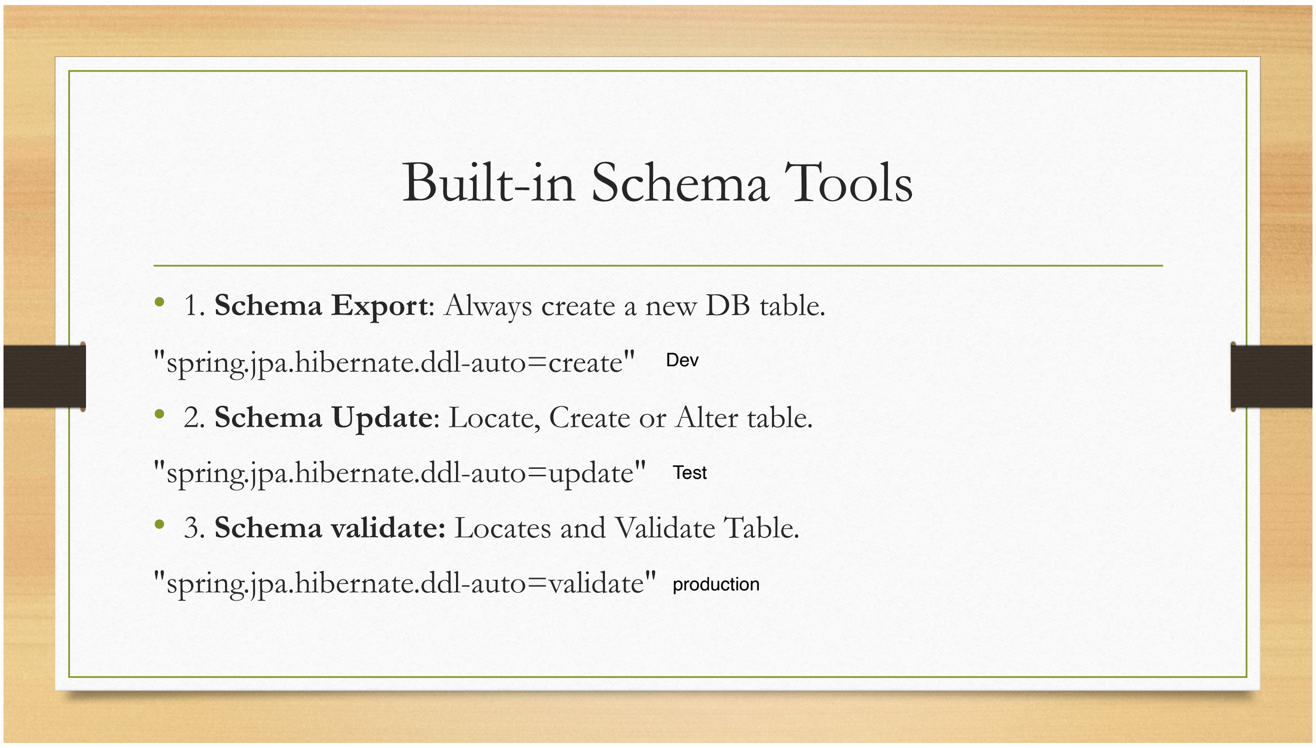
1. @Id
2. @Column
3. @Table
4. @Entity
5. @Transient
6. @Temporal(TemporalType.DATE)



ID Generation Algorithm's:

1. **Assigned** : Assign Id manually(default)
2. **Increment** : Auto Increment value(MaxValue+1), Works with all DB "@GeneratedValue" or "@GeneratedValue(strategy = GenerationType.AUTO)
3. **Identity** : Auto Increment but consider the deleted values, Supported by Mysql "@GeneratedValue(strategy = GenerationType.IDENTITY)"
4. **Sequence** : DB Sequence object created, Supported Oracle, postgres, DB2 etc. "@GeneratedValue(strategy=GenerationType.SEQUENCE)"

```
@GeneratedValue(
   strategy = GenerationType.SEQUENCE,
   generator = "seq_post"
)
@SequenceGenerator(
   name = "seq_post",
   allocationSize = 5
)
```



Built-in Schema Tools

- 1. **Schema Export**: Always create a new DB table.

"spring.jpa.hibernate.ddl-auto=create" Dev

- 2. **Schema Update**: Locate, Create or Alter table.

"spring.jpa.hibernate.ddl-auto=update" Test

- 3. **Schema validate:** Locates and Validate Table.

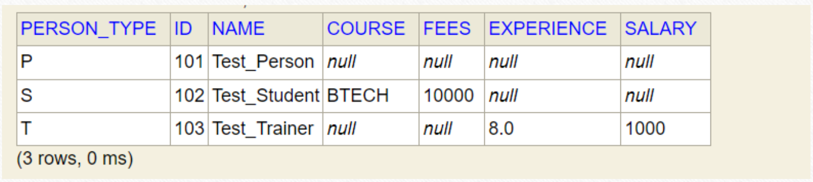
"spring.jpa.hibernate.ddl-auto=validate" production

Inheritance Mapping

1. **Table-per-class** : Single Common Table with a discriminator column
2. **Table-per-subclass*** : Each class uses its own DB table with a reference column between child and parent. Parent holds the primary key.
3. **Table-per-concrete-class** :Each class use their own sepearate tables. child have additional fields of parent also.

1. Table-per-class

- Single Common Table with a discriminator column.
- Person(id,name), Student(course,fees), Trainer(experience,salary)
- Person(Parent)-> Student and Trainer(Sub classes)

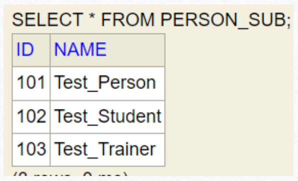


PERSON_TYPE	ID	NAME	COURSE	FEES	EXPERIENCE	SALARY
P	101	Test_Person	*null*	*null*	*null*	*null*
S	102	Test_Student	BTECH	10000	*null*	*null*
T	103	Test_Trainer	*null*	*null*	8.0	1000

(3 rows, 0 ms)

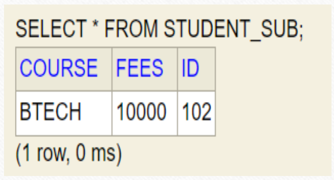
2. Table-per-subclass

- Each class uses its own DB table with a reference column between child and parent. Parent holds the primary key.
- Person(id,name), Student(course,fees), Trainer(experience,salary)
- Person(Parent)-> Student and Trainer(Sub classes)



```
SELECT * FROM PERSON_SUB;
```

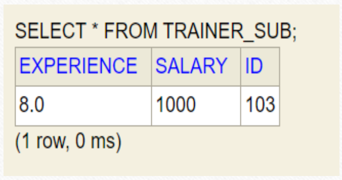
ID	NAME
101	Test_Person
102	Test_Student
103	Test_Trainer



```
SELECT * FROM STUDENT_SUB;
```

COURSE	FEES	ID
BTECH	10000	102

(1 row, 0 ms)



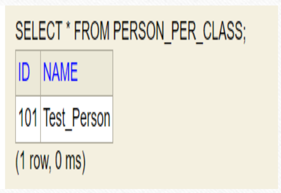
```
SELECT * FROM TRAINER_SUB;
```

EXPERIENCE	SALARY	ID
8.0	1000	103

(1 row, 0 ms)

3. Table-per-concrete-class

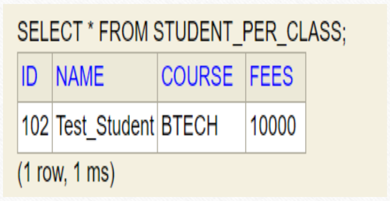
- Each class use their own seperate tables. child have additional fields of parent also.
- Person(id,name), Student(course,fees), Trainer(experience,salary)
- Person(Parent)-> Student and Trainer(Sub classes)



```
SELECT * FROM PERSON_PER_CLASS;
```

ID	NAME
101	Test_Person

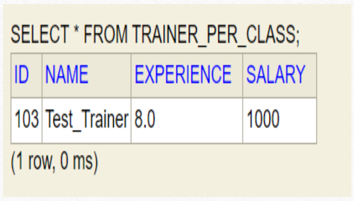
(1 row, 0 ms)



```
SELECT * FROM STUDENT_PER_CLASS;
```

ID	NAME	COURSE	FEES
102	Test_Student	BTECH	10000

(1 row, 1 ms)

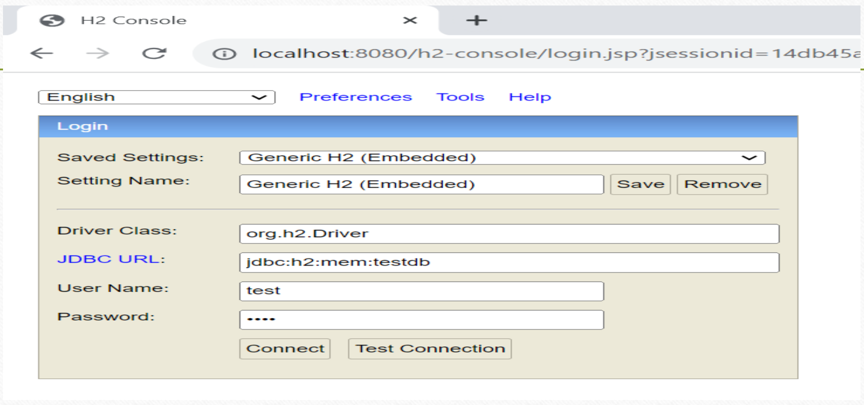


```
SELECT * FROM TRAINER_PER_CLASS;
```

ID	NAME	EXPERIENCE	SALARY
103	Test_Trainer	8.0	1000

(1 row, 0 ms)

H2 in-memory database console


H2 Console

localhost:8080/h2-console/login.jsp?jsessionid=14db45a

English | Preferences | Tools | Help

Login

Saved Settings:	Generic H2 (Embedded)
Setting Name:	Generic H2 (Embedded)
Driver Class:	org.h2.Driver
JDBC URL:	jdbc:h2:mem:testdb
User Name:	test
Password:	••••

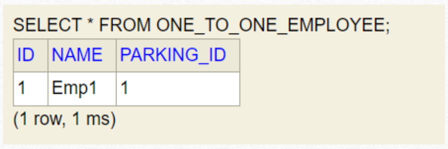
localhost:8080/h2-console/login.jsp

Association/Relationship Mapping

1. **One to One** : Eg: Employee and Parking Space
2. **One to Many** : Eg: Department and Employee
3. **Many to One** : Eg: Employee and Department
4. **Many to Many** : Eg: Employee and Project

1. One to One

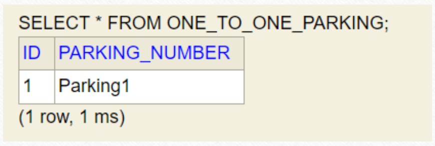
- Employee has a Parking.
- Parent class will have the reference column of child class id(unique).



SELECT * FROM ONE_TO_ONE_EMPLOYEE;

ID	NAME	PARKING_ID
1	Emp1	1

(1 row, 1 ms)



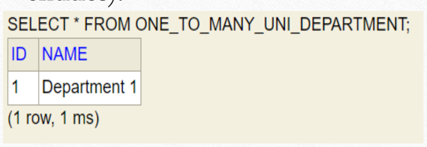
SELECT * FROM ONE_TO_ONE_PARKING;

ID	PARKING_NUMBER
1	Parking1

(1 row, 1 ms)

2. One to Many

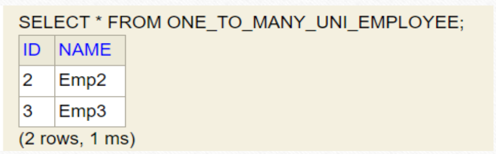
- Department has many Employee
- Separate Table will be needed to map the relationship(having primary key of both entities).



SELECT * FROM ONE_TO_MANY_UNI_DEPARTMENT;

ID	NAME
1	Department 1

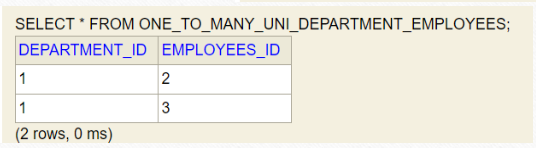
(1 row, 1 ms)



SELECT * FROM ONE_TO_MANY_UNI_EMPLOYEE;

ID	NAME
2	Emp2
3	Emp3

(2 rows, 1 ms)



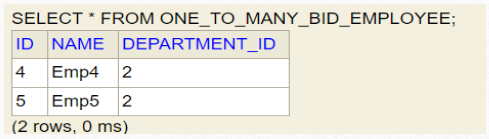
SELECT * FROM ONE_TO_MANY_UNI_DEPARTMENT_EMPLOYEES;

DEPARTMENT_ID	EMPLOYEES_ID
1	2
1	3

(2 rows, 0 ms)

3. Many to One

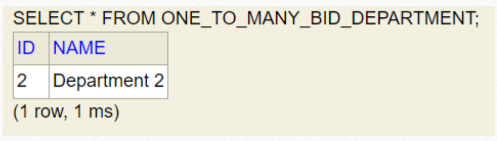
- Many Employee works in one Department
- Parent class will have the reference column of child class id(non-unique).



```
SELECT * FROM ONE_TO_MANY_BID_EMPLOYEE;
```

ID	NAME	DEPARTMENT_ID
4	Emp4	2
5	Emp5	2

(2 rows, 0 ms)



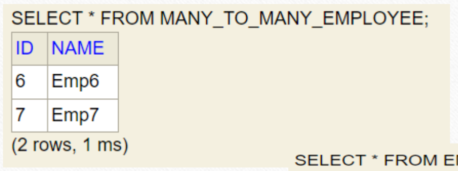
```
SELECT * FROM ONE_TO_MANY_BID_DEPARTMENT;
```

ID	NAME
2	Department 2

(1 row, 1 ms)

4. Many to Many

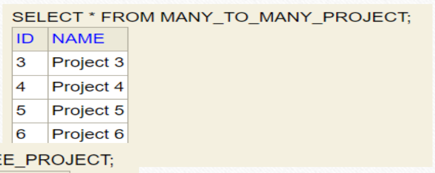
- Many **Employee** works in many **Project. (Separate Mapping tables needed)**



```
SELECT * FROM MANY_TO_MANY_EMPLOYEE;
```

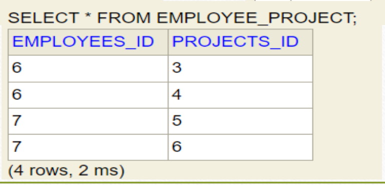
ID	NAME
6	Emp6
7	Emp7

(2 rows, 1 ms)



```
SELECT * FROM MANY_TO_MANY_PROJECT;
```

ID	NAME
3	Project 3
4	Project 4
5	Project 5
6	Project 6



```
SELECT * FROM EMPLOYEE_PROJECT;
```

EMPLOYEES_ID	PROJECTS_ID
6	3
6	4
7	5
7	6

(4 rows, 2 ms)

Passing Parameters in Query

- 1. Named Parameters

Ex: 1.1)

```
public interface EmployeeRepository extends CrudRepository<Employee, Long> {
 @Query("SELECT e FROM Employee e WHERE e.dept = :dept "
        + "AND e.salary < :topSalNum "
        + "ORDER BY e.salary DESC")
 List<Employee> findByDeptTopNSalaries(@Param("topSalNum") long topSalaryNum,
@Param("dept") String dept);
}
```

1. Named Parameters

- 1. 2)

```
TypedQuery<Employee> query = em.createQuery(
 "SELECT e FROM Employee e WHERE e.name = :name AND e.age = :empAge" , Employee.class);
String empName = "John Doe";
int empAge = 55;
List<Employee> employees = query
 .setParameter("name", empName)
 .setParameter("empAge", empAge)
 .getResultList();    //
```

1. Named Parameters

- 1.3)

```
TypedQuery<Employee> query = entityManager.createQuery(
  "SELECT e FROM Employee e WHERE e.empNumber IN (:numbers)" , Employee.class);
List<String> empNumbers = Arrays.asList("A123", "A124");
List<Employee> employees = query.setParameter("numbers", empNumbers).getResultList();
```

2. Positional Parameters

- 2.1)

```
TypedQuery<Employee> query = em.createQuery(
  "SELECT e FROM Employee e WHERE e.empNumber = ?1", Employee.class);
String empNumber = "A123";
Employee employee = query.setParameter(1, empNumber).getSingleResult();
```

2. Positional Parameters

- 2.2)

```
TypedQuery<Employee> query = entityManager.createQuery(
  "SELECT e FROM Employee e WHERE e.empNumber IN (?1)" , Employee.class);
List<String> empNumbers = Arrays.asList("A123", "A124");
List<Employee> employees = query.setParameter(1, empNumbers).getResultList();  //Passing Collection Value
```

Code Reference: https://github.com/aroopkumar/jpa-mapping-examples.git

THANK YOU.