無料電子ブック

学習

jpa

Free unaffiliated eBook created from
Stack Overflow contributors.

#jpa

1: jpaをいめる

JPAは、Java Persistence APIです.Java Persistence APIは、Javaオブジェクトとそれらのをリレーショナル・データベースにマッピングするをします。これは、オブジェクト・リレーショナル・マッパーORMとばれます。これは、よりレベルのJDBCのまたはです。これは、Javaのアプローチをし、なオブジェクトグラフをするがあるにもです。

JPAはではありません。これにはプロバイダがですを。のJPA 2.1のは、 EclipseLink 「のがであることをする」というのJPA 2.1のリファレンスですです。 Hibernate 、およびDataNucleusがまれます。

メタデータ

Javaオブジェクトとデータベーステーブルのマッピングは、 メタデータによってされます。 JPAプロバイダは、メタデータをしてしいデータベースをします。 JPAは、Javaクラスのをしてメタデータをします。

オブジェクト・リレーショナル・エンティティ・アーキテクチャ

エンティティアーキテクチャはのものでされます。

-
-
- コンテキスト
- エンティティマネージャファクトリ
- エンティティマネージャ

バージョン

バージョン	グループ	リリース
1.0	JSR-220	2006-11-06
2.0	JSR-317	20091210
2.1	JSR-338	2013-05-22

Examples

インストールまたはセットアップ

クラスパス

Eclipselink

Eclipselink APIとJPA APIをめるがあります。 Mavenのの

```
<dependencies>
  <dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>eclipselink</artifactId>
    <version>2.6.3</version>
  </dependency>
  <dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>javax.persistence</artifactId>
    <version>2.1.1</version>
  </dependency>
  <!-- ... -->
</dependencies>
```

Hibernate-coreがです。 Mavenのの

```
<dependencies>
  <dependency>
    <!-- requires Java8! -->
    <!-- as of 5.2, hibernate-entitymanager is merged into hibernate-core -->
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.1.Final</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.1-api</artifactId>
    <version>1.0.0</version>
  </dependency>
  <!-- ... -->
</dependencies>
```

DataNucleus

datanucleus-core、datanucleus-api-jpaおよびdatanucleus-rdbmsRDBMSデータストアをするがです。 Mavenのの

```
<dependencies>
  <dependency>
    <groupId>org.datanucleus</groupId>
    <artifactId>datanucleus-core</artifactId>
    <version>5.0.0-release</version>
  </dependency>
  <dependency>
    <groupId>org.datanucleus</groupId>
    <artifactId>datanucleus-api-jpa</artifactId>
    <version>5.0.0-release</version>
  </dependency>
```

```
  <dependency>
    <groupId>org.datanucleus</groupId>
    <artifactId>datanucleus-rdbms</artifactId>
    <version>5.0.0-release</version>
  </dependency>
  <dependency>
    <groupId>org.datanucleus</groupId>
    <artifactId>javax.persistence</artifactId>
    <version>2.1.2</version>
  </dependency>
  <!-- ... -->
</dependencies>
```

の

JPAでは、CLASSPATHのルートにある`META-INF`にあるファイル*persistence.xml*をするがあります。このファイルには、JPAがなユニットのがまれています。

JPAはさらに、`META-INF`にかれたマッピングファイル*orm.xml*をできます。このマッピングファイルは、クラスがデータストアにどのようにマップされるかをするためにされ、JPAエンティティクラスでJavaアノテーションをする/です。

の**persistence.xml**の

Hibernateおよびめまれた**H2 DB**

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                             http://java.sun.com/xml/ns/persistence/persistence_2_1.xsd"
         version="2.1">

<persistence-unit name="persistenceUnit">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>

    <class>my.application.entities.MyEntity</class>

    <properties>
        <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
        <property name="javax.persistence.jdbc.url" value="jdbc:h2:data/myDB.db" />
        <property name="javax.persistence.jdbc.user" value="sa" />

        <!-- DDL change options -->
        <property name="javax.persistence.schema-generation.database.action" value="drop-and-
create"/>

        <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
        <property name="hibernate.flushMode" value="FLUSH_AUTO" />
    </properties>
</persistence-unit>
</persistence>
```

Eclipselinkおよびめまれた H2 DB

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                             http://java.sun.com/xml/ns/persistence/persistence_2_1.xsd"
         version="2.1">

<persistence-unit name="persistenceUnit">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

    <class>my.application.entities.MyEntity</class>

    <properties>
        <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
        <property name="javax.persistence.jdbc.url" value="jdbc:h2:data/myDB.db"/>
        <property name="javax.persistence.jdbc.user" value="sa"/>

        <!-- Schema generation : drop and create tables -->
        <property name="javax.persistence.schema-generation.database.action" value="drop-and-
create-tables" />
    </properties>
</persistence-unit>

</persistence>
```

DataNucleusおよびめみ H2 DB

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                             http://java.sun.com/xml/ns/persistence/persistence_2_1.xsd"
         version="2.1">

<persistence-unit name="persistenceUnit">
    <provider>org.datanucleus.api.jpa.PersistenceProviderImpl</provider>

    <class>my.application.entities.MyEntity</class>

    <properties>
        <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
        <property name="javax.persistence.jdbc.url" value="jdbc:h2:data/myDB.db"/>
        <property name="javax.persistence.jdbc.user" value="sa"/>

        <!-- Schema generation : drop and create tables -->
        <property name="javax.persistence.schema-generation.database.action" value="drop-and-
create-tables" />
    </properties>
</persistence-unit>

</persistence>
```

こんにちは

なハロ―ワ―ルドをするためのなコンポ―ネントをすべててみましょう。

1. するJPA 2.1のをする
2. `persistence-unit`するデ―タベ―スへのをする
3. エンティティをします。
4. DAOデ―タアクセスオブジェクトをしてエンティティをする
5. アプリケ―ションのテスト

Mavenをうには、このがです。

```
<dependencies>

    <!-- JPA is a spec, I'll use the implementation with HIBERNATE -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>5.2.6.Final</version>
    </dependency>

    <!-- JDBC Driver, use in memory DB -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.193</version>
    </dependency>

</dependencies>
```

パ―シスタンスユニット

resourcesフォルダで、 `persistence.xml`というファイルをするがあります。それをするもなはのようなものです

```
<persistence-unit name="hello-jpa-pu" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <properties>
        <!-- ~ = relative to current user home directory -->
        <property name="javax.persistence.jdbc.url" value="jdbc:h2:./test.db"/>
        <property name="javax.persistence.jdbc.user" value=""/>
        <property name="javax.persistence.jdbc.password" value=""/>
        <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
        <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
        <property name="hibernate.show_sql" value="true"/>

        <!-- This create automatically the DDL of the database's table -->
        <property name="hibernate.hbm2ddl.auto" value="create-drop"/>

    </properties>
</persistence-unit>
```

エンティティをする

はクラス`Biker`をする

```
package it.hello.jpa.entities;


import javax.persistence.*;
import java.io.Serializable;
import java.util.Date;
import java.util.List;

@Entity
@Table(name = "BIKER")
public class Biker implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(name = "bikerName")
    private String name;

    @Column(unique = true, updatable = false)
    private String battleName;

    private Boolean beard;

    @Temporal(TemporalType.DATE)
    private Date birthday;

    @Temporal(TemporalType.TIME)
    private Date registrationDate;

    @Transient // --> this annotiation make the field transient only for JPA
    private String criminalRecord;

    public Long getId() {
        return this.id;
    }

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

    public String getName() {
        return this.name;
    }

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

    public String getBattleName() {
        return battleName;
    }

    public void setBattleName(String battleName) {
```

```
        this.battleName = battleName;
    }

    public Boolean getBeard() {
        return this.beard;
    }

    public void setBeard(Boolean beard) {
        this.beard = beard;
    }

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    public Date getRegistrationDate() {
        return registrationDate;
    }

    public void setRegistrationDate(Date registrationDate) {
        this.registrationDate = registrationDate;
    }

    public String getCriminalRecord() {
        return criminalRecord;
    }

    public void setCriminalRecord(String criminalRecord) {
        this.criminalRecord = criminalRecord;
    }
}
```

DAOをする

```
package it.hello.jpa.business;

import it.hello.jpa.entities.Biker;

import javax.persistence.EntityManager;
import java.util.List;

public class MotorcycleRally {

    public Biker saveBiker(Biker biker) {
        EntityManager em = EntityManagerUtil.getEntityManager();
        em.getTransaction().begin();
        em.persist(biker);
        em.getTransaction().commit();
        return biker;
    }

}
```

EntityManagerUtilはシングルトンです

```
package it.hello.jpa.utils;

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

public class EntityManagerUtil {

    // USE THE SAME NAME IN persistence.xml!
    public static final String PERSISTENCE_UNIT_NAME = "hello-jpa-pu";

    private static EntityManager entityManager;

    private EntityManagerUtil() {
    }

    public static EntityManager getEntityManager() {
        if (entityManager == null) {
            // the same in persistence.xml
            EntityManagerFactory emFactory =
Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);


            return emFactory.createEntityManager();
        }
        return entityManager;
    }
}
```

アプリケーションのテスト

パッケージit.hello.jpa.test;

パブリッククラスTestJpa {

```
@Test
public void insertBiker() {
    MotorcycleRally crud = new MotorcycleRally();

    Biker biker = new Biker();
    biker.setName("Manuel");
    biker.setBeard(false);

    biker = crud.saveBiker(biker);

    Assert.assertEquals(biker.getId(), Long.valueOf(1L));
}
```

}

はのようになります。

it.hello.jpa.test.TestJpa Hibernateドロップテーブル BIKERもしあればHibernateもし

hibernate_sequenceがすればhibernateシーケンスをするhibernate_sequenceは1ずつでまるHibernatecreate table BIKERid bigint not null、battleName varchar255 BattleName、beard、birthday、bikerName、registrationDateBibernateBIKER BattleName、beard、beard、bikerNameをします。 2019、2017 11:00:02 PM orgHibernate.jpa.internal.util.LogHelper logPersistenceUnitInformation INFO HHH000204PersistenceUnitInfoをするhello- jpa-pu ...]

テスト1、0、エラー0、スキップ0

オンラインでjpaをいめるをむ https://riptutorial.com/ja/jpa/topic/2125/jpaをいめる

2: 11マッピング

パラメーター

@TableGenerator	ジェネレータがつかるジェネレータとテーブルをします。
@GeneratedValue	をし、ジェネレータのをします。
@OneToOne	とのの11のをします。ここで、はのです
mappedBy	このはのにあります。これにより、

Examples

との11の

とのに11のがあるとします。

Employee.java

```
@Entity
public class Employee {

    @TableGenerator(name = "employee_gen", table = "id_gen", pkColumnName = "gen_name",
valueColumnName = "gen_val", allocationSize = 100)
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "employee_gen")
    private int idemployee;
    private String firstname;
    private String lastname;
    private String email;

    @OneToOne
    @JoinColumn(name = "iddesk")
    private Desk desk;

    // getters and setters
}
```

Desk.java

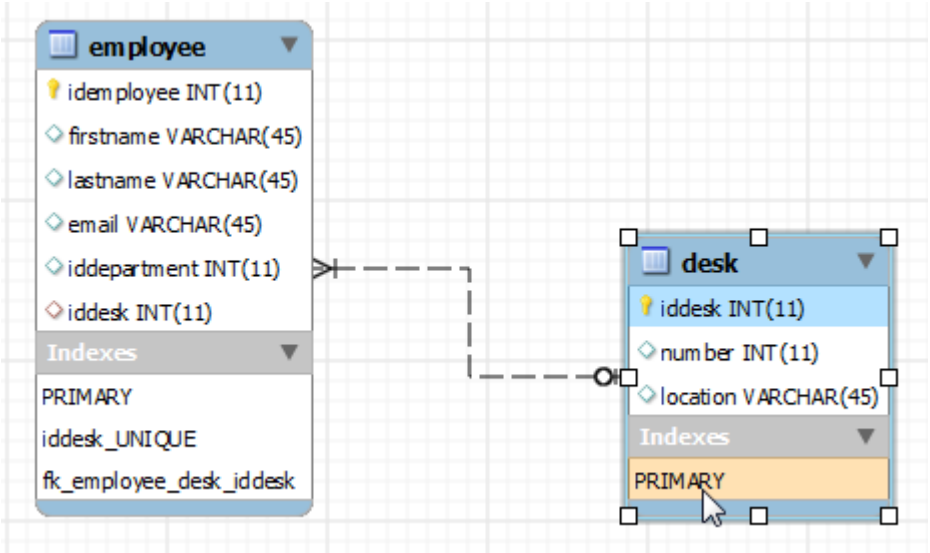
```
@Entity
public class Desk {

    @TableGenerator(table = "id_gen", name = "desk_gen", pkColumnName = "gen_name",
valueColumnName = "gen_value", allocationSize = 1)
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "desk_gen")
```

```
    private int iddesk;
    private int number;
    private String location;
    @OneToOne(mappedBy = "desk")
    private Employee employee;

    // getters and setters
}
```

テストコード

```
/* Create EntityManagerFactory */
    EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("JPAExamples");

    /* Create EntityManager */
    EntityManager em = emf.createEntityManager();

    Employee employee;

    employee = new Employee();
    employee.setFirstname("pranil");
    employee.setLastname("gilda");
    employee.setEmail("sdfsdf");

    Desk desk = em.find(Desk.class, 1); // retrieves desk from database
    employee.setDesk(desk);

    em.persist(employee);

    desk = em.find(Desk.class, 1); // retrieves desk from database
    desk.setEmployee(employee);
    System.out.println(desk.getEmployee());
```

データベースダイアグラムをにします。



- **@JoinColumn**アノテーションは、ジョイン･コラムをむテーブルにマップされるエンティティのマッピングをいます。の。の、Employeeテーブルには**Join**カラムがあり、 **@JoinColumn**はEmployeeエンティティのDeskフィールドにあります。
- **mappedBy**は、ののエンティティの**@OneToOne**でするがあります。すなわち、データベ

—スのアスペクトでをしないエンティティ。たちの、デスクはのエンティティです。

なはここにあります

オンラインで11マッピングをむ https://riptutorial.com/ja/jpa/topic/6474/11マッピング

3: 1の

パラ メーター

@TableGenerator	ジェネレータがつかるジェネレータとテーブルをします。
@GeneratedValue	をし、ジェネレータのをします。
@ManyToOne	とのに1のをします。
@OneToManymappedBy = "department"	Employeeエンティティの@ManyToOneアノテーションをにすることによって、EmployeeとDepartmentののをします。

Examples

1の

1のマッピングは、に、1マッピングのなるです。1マッピングのためにったじをりげます。

Employee.java

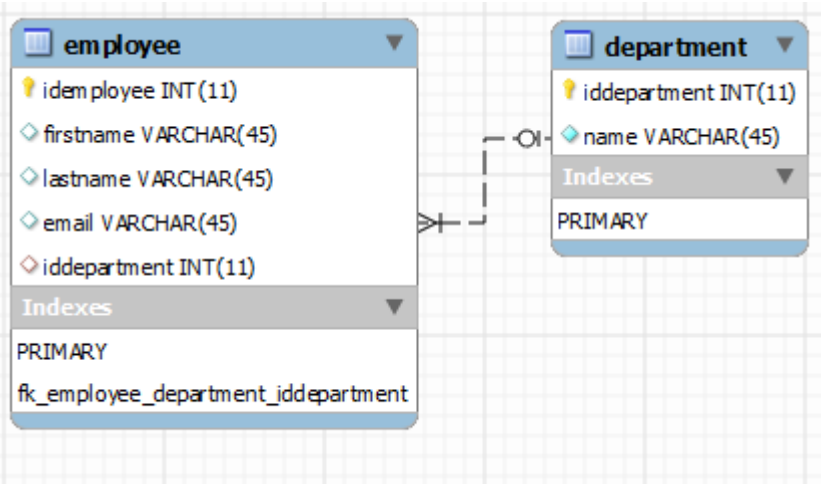
```
@Entity
public class Employee {

    @TableGenerator(name = "employee_gen", table = "id_gen", pkColumnName = "gen_name",
valueColumnName = "gen_val", allocationSize = 100)
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "employee_gen")
    private int idemployee;
    private String firstname;
    private String lastname;
    private String email;

    @ManyToOne
    @JoinColumn(name = "iddepartment")
    private Department department;

    // getters and setters
}
```

Department.java

```
@Entity
public class Department {

    @TableGenerator(table = "id_gen", pkColumnName = "gen_name", valueColumnName = "gen_val",
name = "department_gen", allocationSize = 1)
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "department_gen")
    private int iddepartment;
    private String name;

    @OneToMany(mappedBy = "department")
    private List<Employee> employees;

    // getters and setters
}
```

このはのようにデータベースにされます。



jpaの1マッピングについてえておくべき2つのポイントがあります。

- くの、のがのです。はそのでされます。
- 1のマッピングはのであるため、mappedByをでするがあります。

なはここですることができます

オンラインで1のをむ https://riptutorial.com/ja/jpa/topic/6529/1の

4: エンティティの

エンティティのの

キーは、のキーはキーをのテーブルでする1つのにすることができます。

キーとそれがするプライマリキーのとはじでなければなりません。

キーは、あるテーブルのからのテーブルのへのをします。

Examples

エンティティリレーションシップにおける

エンティティリレーションシップにおける

はのタイプのものです

- **11** エンティティインスタンスは、のエンティティの1つのインスタンスにしています。
- エンティティインスタンスは、のエンティティののインスタンスにけることができます。
- **1** エンティティののインスタンスは、のエンティティののインスタンスにけることができます。
- エンティティインスタンスは、にのインスタンスにけることができます。

11マッピング

11のマッピングは、11のをつのエンティティへののけをします。このマッピングは、するプロパティまたはフィールドの`@OneToOne`アノテーションをします。

*`Vehicle`*および*`ParkingPlace`*エンティティ。

1マッピング

エンティティインスタンスは、のエンティティののインスタンスにけることができます。

1 `@OneToMany`リレーションシップは、するプロパティまたはフィールドで`@OneToMany`アノテーションをします。

`mappedBy`は、するエンティティのManyToOneによってされたをするためにです。

```
@OneToMany(mappedBy="attribute")
```

1のけでは、エンティティのコレクションをマップするがあります。

1マッピング

1のマッピングは、ソースエンティティターゲットエンティティをするのに`@ManyToOne`アノテーションをけることによってされます。

`@JoinColumn(name="FK_name")`は、のforeingキーをディスクにきします。

マッピング

エンティティインスタンスは、いにのインスタンスにけることができます。

リレーションシップは、するプロパティまたはフィールドで`@ManyToMany`アノテーションをします。

3つのテーブルをして2つのエンティティタイプジョインテーブルをけるがあります。

@JoinTableの

JPAでのをマッピングする、 `@JoinTable`アノテーションをして、 `@JoinTable`キーのにされるテーブルのコンフィグレーションをできます。

```
@Entity
public class EntityA {
   @Id
   @Column(name="id")
   private long id;
   [...]
   @ManyToMany
   @JoinTable(name="table_join_A_B",
              joinColumns=@JoinColumn(name="id_A"), referencedColumnName="id"
              inverseJoinColumns=@JoinColumn(name="id_B", referencedColumnName="id"))
   private List<EntityB> entitiesB;
   [...]
}

@Entity
public class EntityB {
   @Id
   @Column(name="id")
   private long id;
   [...]
}
```

エンティティAによりEntityBにのをするからる。このでは、 `entitiesB`フィールドを、々はテーブルのテーブルがあることをする@JoinTableアノテーションを`table_join_A_B`によってされる、`id_A`び`id_B` 、それぞれEntityAのテーブルとEntityBのテーブルのカラム`id`をするキー。 `(id_A,id_B)`は、 `table_join_A_B`テーブルのプライマリキーに`table_join_A_B`ます。

オンラインでエンティティのをむ https://riptutorial.com/ja/jpa/topic/6305/エンティティの

5: なクラスのの

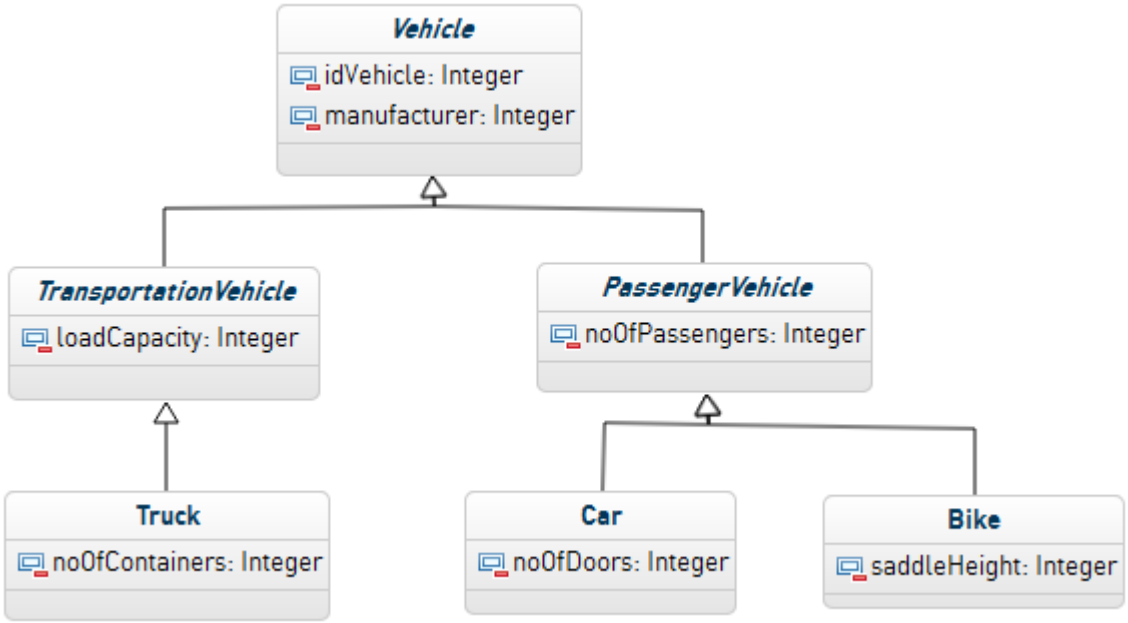
- Vehicle、TransportationVehicle、PassengerVehicleはクラスであり、データベースにはのテーブルはありません。
- Truck、Car、Bikeはなクラスであるため、するにマップされます。これらのテーブルには、するテーブルがデータベースにないため、@MappedSuperClassでがけられたクラスのすべてのフィールドをめるがあります。
- したがって、Truckテーブルには、TransportationVehicleおよびVehicleからされたフィールドをするがあります。
- に、CarとBikeには、PassengerVehicleとVehicleからしたフィールドをするがあります。

なはここにあります

Examples

なクラスのの

にすように、のをりげます。



Vehicle.java

```
package com.thejavageek.jpa.entities;

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.TableGenerator;
```

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Vehicle {

    @TableGenerator(name = "VEHICLE_GEN", table = "ID_GEN", pkColumnName = "GEN_NAME",
valueColumnName = "GEN_VAL", allocationSize = 1)
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "VEHICLE_GEN")
    private int idVehicle;
    private String manufacturer;

    public int getIdVehicle() {
        return idVehicle;
    }

    public void setIdVehicle(int idVehicle) {
        this.idVehicle = idVehicle;
    }

    public String getManufacturer() {
        return manufacturer;
    }

    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }

}
```

TransportationVehilcle.java

```
package com.thejavageek.jpa.entities;

import javax.persistence.MappedSuperclass;

@MappedSuperclass
public abstract class TransportationVehicle extends Vehicle {

    private int loadCapacity;

    public int getLoadCapacity() {
        return loadCapacity;
    }

    public void setLoadCapacity(int loadCapacity) {
        this.loadCapacity = loadCapacity;
    }

}
```

Truck.java

```
package com.thejavageek.jpa.entities;

import javax.persistence.Entity;

@Entity
public class Truck extends TransportationVehicle {
```

```
    private int noOfContainers;

    public int getNoOfContainers() {
        return noOfContainers;
    }

    public void setNoOfContainers(int noOfContainers) {
        this.noOfContainers = noOfContainers;
    }

}
```

PassengerVehicle.java

```
package com.thejavageek.jpa.entities;

import javax.persistence.MappedSuperclass;

@MappedSuperclass
public abstract class PassengerVehicle extends Vehicle {

    private int noOfpassengers;

    public int getNoOfpassengers() {
        return noOfpassengers;
    }

    public void setNoOfpassengers(int noOfpassengers) {
        this.noOfpassengers = noOfpassengers;
    }

}
```

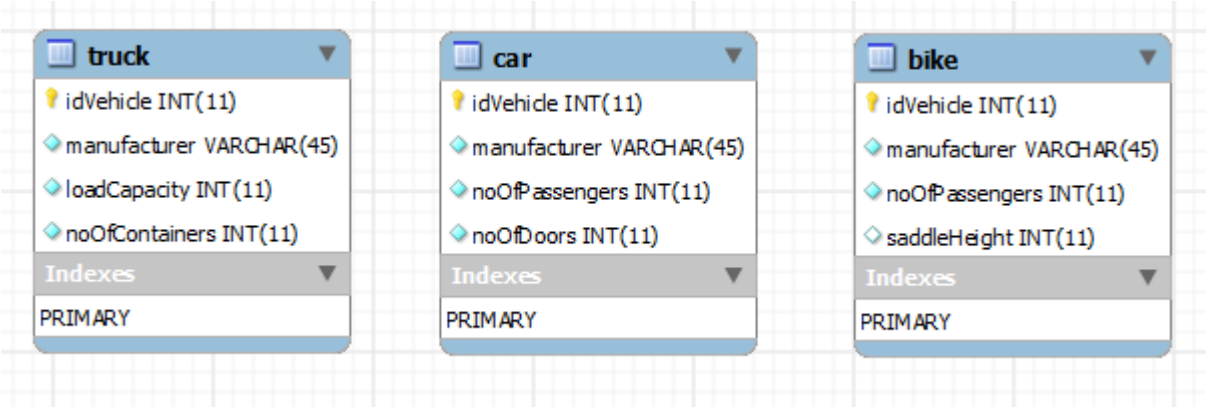
Car.java

```
package com.thejavageek.jpa.entities;

import javax.persistence.Entity;

@Entity
public class Car extends PassengerVehicle {

    private int noOfDoors;

    public int getNoOfDoors() {
        return noOfDoors;
    }

    public void setNoOfDoors(int noOfDoors) {
        this.noOfDoors = noOfDoors;
    }

}
```

Bike.java

```
package com.thejavageek.jpa.entities;

import javax.persistence.Entity;

@Entity
public class Bike extends PassengerVehicle {

    private int saddleHeight;

    public int getSaddleHeight() {
        return saddleHeight;
    }

    public void setSaddleHeight(int saddleHeight) {
        this.saddleHeight = saddleHeight;
    }

}
```

データベースのはのようになります



オンラインでなクラスののをむ https://riptutorial.com/ja/jpa/topic/6255/なクラスのの

6: テーブル

パラメーター

@	されるストラテジのタイプをします。
@DiscriminatorColumn	エンティティにりてられたのIDにづいてなるエンティティをするためにされるデータベースのをします
@MappedSuperClass	マップされたスーパークラスはではなく、そのサブクラスのをするためにのみされます。にJavaクラスは@MapperSuperClassでマークされています
@DiscriminatorValue	@DiscriminatorColumnでされたでされた。このは、エンティティのタイプをするのにちます

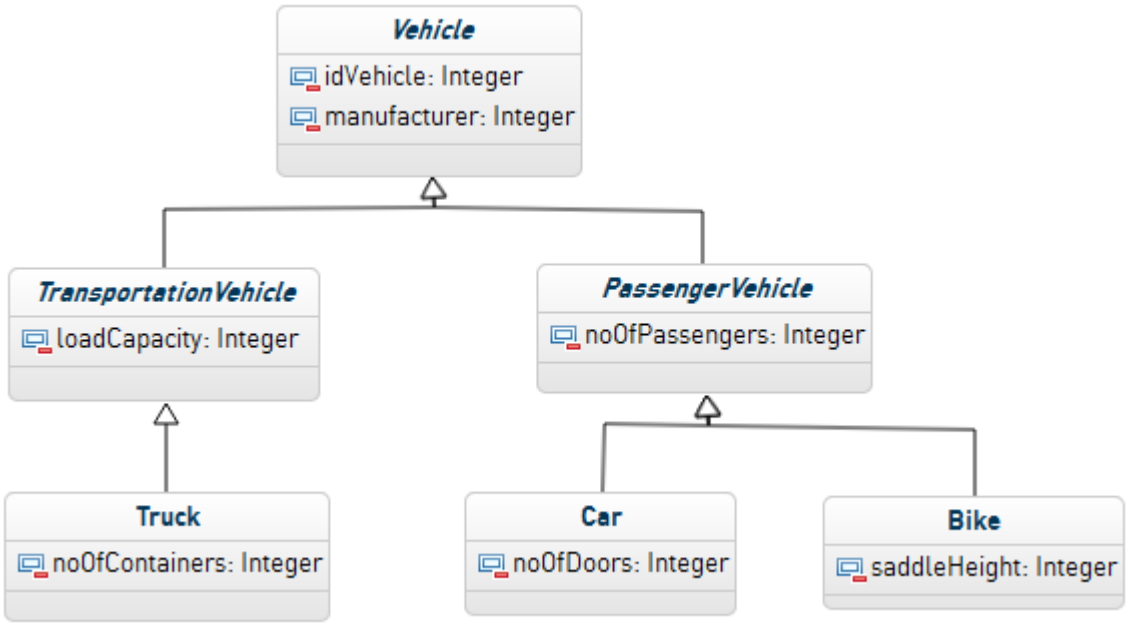
テーブルのは、エンティティのとになをとしないで、くのがNULLであるがあり、それらのためのデータがないためデータベースをにすることです。

なとはここにあります

Examples

テーブルの

のテーブルのためにVehicleのなをることができます。



なクラス

```
package com.thejavageek.jpa.entities;

import javax.persistence.DiscriminatorColumn;
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;
import javax.persistence.TableGenerator;

@Entity
@Table(name = "VEHICLE")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "VEHICLE_TYPE")
public abstract class Vehicle {

    @TableGenerator(name = "VEHICLE_GEN", table = "ID_GEN", pkColumnName = "GEN_NAME",
valueColumnName = "GEN_VAL", allocationSize = 1)
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "VEHICLE_GEN")
    private int idVehicle;
    private String manufacturer;

    public int getIdVehicle() {
        return idVehicle;
    }

    public void setIdVehicle(int idVehicle) {
        this.idVehicle = idVehicle;
    }

    public String getManufacturer() {
        return manufacturer;
    }
```

```
    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }

}
```

TransportableVehicle.javaパッケージcom.thejavageek.jpa.entities;

import javax.persistence.MappedSuperclass;

```
@MappedSuperclass
public abstract class TransportationVehicle extends Vehicle {

    private int loadCapacity;

    public int getLoadCapacity() {
        return loadCapacity;
    }

    public void setLoadCapacity(int loadCapacity) {
        this.loadCapacity = loadCapacity;
    }

}
```

PassengerVehicle.java

```
package com.thejavageek.jpa.entities;

import javax.persistence.MappedSuperclass;

@MappedSuperclass
public abstract class PassengerVehicle extends Vehicle {

    private int noOfpassengers;

    public int getNoOfpassengers() {
        return noOfpassengers;
    }

    public void setNoOfpassengers(int noOfpassengers) {
        this.noOfpassengers = noOfpassengers;
    }

}
```

Truck.java

```
package com.thejavageek.jpa.entities;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue(value = "Truck")
public class Truck extends TransportationVehicle{
```

```
    private int noOfContainers;

    public int getNoOfContainers() {
        return noOfContainers;
    }

    public void setNoOfContainers(int noOfContainers) {
        this.noOfContainers = noOfContainers;
    }

}
```

Bike.java

```
package com.thejavageek.jpa.entities;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue(value = "Bike")
public class Bike extends PassengerVehicle {

    private int saddleHeight;

    public int getSaddleHeight() {
        return saddleHeight;
    }

    public void setSaddleHeight(int saddleHeight) {
        this.saddleHeight = saddleHeight;
    }

}
```

Car.java

```
package com.thejavageek.jpa.entities;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue(value = "Car")
public class Car extends PassengerVehicle {

    private int noOfDoors;

    public int getNoOfDoors() {
        return noOfDoors;
    }

    public void setNoOfDoors(int noOfDoors) {
        this.noOfDoors = noOfDoors;
    }

}
```

テストコード

```
/* Create EntityManagerFactory */
EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("AdvancedMapping");

/* Create EntityManager */
EntityManager em = emf.createEntityManager();
EntityTransaction transaction = em.getTransaction();
transaction.begin();

Bike cbr1000rr = new Bike();
cbr1000rr.setManufacturer("honda");
cbr1000rr.setNoOfpassengers(1);
cbr1000rr.setSaddleHeight(30);
em.persist(cbr1000rr);

Car avantador = new Car();
avantador.setManufacturer("lamborghini");
avantador.setNoOfDoors(2);
avantador.setNoOfpassengers(2);
em.persist(avantador);

Truck truck = new Truck();
truck.setLoadCapacity(100);
truck.setManufacturer("mercedes");
truck.setNoOfContainers(2);
em.persist(truck);

transaction.commit();
```

オンラインでテーブルをむ https://riptutorial.com/ja/jpa/topic/6277/テーブル

7: なマッピング

パラメーター

`@Id`	エンティティのキーとしてフィールド/をマークします。
`@Basic`	されたフィールドをタイプとしてマップすることをマークします。これはプリミティブとそのラッパー、 `String` 、 `Date` 、 `Calendar` され`Calendar` 。パラメタがえられていない、にははオプションですが、をにするためにはスタイルがいでしょう。
`@Transient`	なものとしてマークされたフィールドは、の`transient`キーワードとに、のためにされません。

にデフォルトのコンストラクタ、つまりパラメータのないコンストラクタがです。なでは、コンストラクタがされていなかったので、Javaは1をしました。をつコンストラクタをするは、パラメータのないコンストラクタもずしてください。

Examples

になエンティティ

```
@Entity
class Note {
    @Id
    Integer id;

    @Basic
    String note;

    @Basic
    int count;
}
```

Getters、setterなどはにされていますが、JPAにはありません。

このJavaクラスは、のにマッピングされますデータベースにじて、Postgresマッピングのがあります。

```
CREATE TABLE Note (
  id integer NOT NULL,
  note text,
  count integer NOT NULL
)
```

JPAプロバイダをしてDDLをし、ここにすDDLとはなるDDLをするがありますが、があるりにはしません。 DDLのにらないことがです。

マッピングからのフィールドの

```
@Entity
class Note {
    @Id
    Integer id;

    @Basic
    String note;

    @Transient
    String parsedNote;

    String readParsedNote() {
        if (parsedNote == null) { /* initialize from note */ }
        return parsedNote;
    }
}
```

クラスにデータベースにきまれてはならないフィールドがなは、 `@Transient`としてマークします。データベースからのみり、フィールドは`null`になり`null`。

マッピングのと

Javaでは、 `Date`と`Calendar` 、そしての`LocalDate`と`LocalDateTime`な`Date`と`Calendar`があります。`Timestamp` 、 `Instant` 、 `ZonedLocalDateTime` 、 `ZonedLocalDateTime` -timeのタイプがあります。データベースでは、 `time` 、 `date` 、 `timestamp` とのをち`date` 、 `timestamp`ゾーンのにかかわらずがあります。

Java 8よりのと

プリJava-8タイプの`java.util.Date` 、 `java.util.Calendar` 、および`java.sql.Timestamp`のデフォルトマッピングはSQLの`timestamp`です。 `java.sql.Date`は`date`です。

```
@Entity
class Times {
    @Id
    private Integer id;

    @Basic
    private Timestamp timestamp;

    @Basic
    private java.sql.Date sqldate;

    @Basic
    private java.util.Date utildate;
```

```
    @Basic
    private Calendar calendar;
}
```

これは、のににします。

```
CREATE TABLE times (
    id integer not null,
    timestamp timestamp,
    sqldate date,
    utildate timestamp,
    calendar timestamp
)
```

これはではないかもしれません。たとえば、Javaの`Date`または`Calendar`は、のみをすためにのためにされることがよくあります。デフォルトのマッピングをするには、またはマッピングをにするために`@Temporal`アノテーションをします。

```
@Entity
class Times {
    @Id
    private Integer id;

    @Temporal(TemporalType.TIME)
    private Date date;

    @Temporal(TemporalType.DATE)
    private Calendar calendar;
}
```

のSQLテーブルはのとおりです。

```
CREATE TABLE times (
    id integer not null,
    date time,
    calendar date
)
```

1 `@Temporal`されたはDDLにします。 `@Basic`だけで`Date`の`date`のをつこともできます。

2 `Calendar`は`time`だけすることはできません。

Java 8のと

JPA 2.1では、Java 8でされている`java.time`のサポートはされていません `java.time` 2.1のは、にはベンダーではありますが、これらのタイプのサポートをしています。

DataNucleusの、これらのタイプは`@Temporal`でし、 `@Temporal`アノテーションをしてさまざまなマッピングのをします。

Hibernateの、Hibernate 5.2+をしている、 `@Basic`アノテーションをするだけですぐに`@Basic`ます。 Hibernate 5.0-5.1をしているは、`org.hibernate:hibernate-java8`をするがあります。されるマッピングは

- `LocalDate`に`date`
- `Instant` 、 `LocalDateTime` 、 `ZonedDateTime`を`timestamp` `ZonedDateTime`

ベンダーにしないは、がなJava 8の`java.time`のJPA 2.1 `AttributeConverter`をすることです。

シーケンスされたIDをつエンティティ

ここではクラスがあり、IDフィールド `userUid` にデータベースのSEQUENCEをしてがされるようにします。このSEQUENCEは`USER_UID_SEQ`と`USER_UID_SEQ`れ、DBAによってされるか、JPAプロバイダによってされます。

```
@Entity
@Table(name="USER")
public class User implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @SequenceGenerator(name="USER_UID_GENERATOR", sequenceName="USER_UID_SEQ")
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="USER_UID_GENERATOR")
    private Long userUid;

    @Basic
    private String userName;
}
```

オンラインでなマッピングをむ https://riptutorial.com/ja/jpa/topic/3691/なマッピング

8: 1マッピン グ

パラ メ―タ―

カラム	カラム
@TableGenerator	IDのためのテ―ブルジェネレ―タの
@GeneratedValue	フィ―ルドにされるがされたであることをします。
@Id	フィ―ルドをとしてする
@ManyToOne	との1のをします。このはくのでマ―クされています。のが1つのにしています。だからDepartmentはEmployeeエンティティの@ ManyToOneでがけられています。
@JoinColumn	エンティティのキ―をするデ―タベ―ステ―ブルのをします。

Examples

からへのManyToOneの

```
@Entity
public class Employee {

    @TableGenerator(name = "employee_gen", table = "id_gen", pkColumnName = "gen_name",
valueColumnName = "gen_val", allocationSize = 1)
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "employee_gen")
    private int idemployee;
    private String firstname;
    private String lastname;
    private String email;

    @ManyToOne
    @JoinColumn(name = "iddepartment")
    private Department department;

    // getters and setters
    // toString implementation
}
```

エンティティ

```
@Entity
public class Department {

    @Id
```

```
    private int iddepartment;
    private String name;

    // getters, setters and toString()
}
```

テストクラス

```
public class Test {

    public static void main(String[] args) {

        EntityManagerFactory emf = Persistence
                .createEntityManagerFactory("JPAExamples");
        EntityManager em = emf.createEntityManager();
        EntityTransaction txn = em.getTransaction();

        Employee employee = new Employee();
        employee.setEmail("someMail@gmail.com");
        employee.setFirstname("Prasad");
        employee.setLastname("kharkar");

        txn.begin();
        Department department = em.find(Department.class, 1);//returns the department named
vert
        System.out.println(department);
        txn.commit();

        employee.setDepartment(department);

        txn.begin();
        em.persist(employee);
        txn.commit();

    }

}
```

オンラインで1マッピングをむ https://riptutorial.com/ja/jpa/topic/6531/1マッピング

9: マッピング

き

ManyToManyマッピングは、エンティティのをします。エンティティのは、にのインスタンスにけることができ、 @ManyToManyアノテーションでされます。

エンティティのテーブルのキーができる@OneToManyとはなり、 ManyToManyはエンティティをにマップするテーブルがです。

パラメーター

@TableGenerator	GeneratedValueアノテーションにジェネレータがされたときに、でされるキージェネレータをします
@GeneratedValue	キーののをします。 Idアノテーションとに、エンティティまたはマップされたスーパークラスのプライマリキーのプロパティまたはフィールドにできます。
@ManyToMany	くのがのプロジェクトでできるように、とプロジェクトのエンティティのをします。
mappedBy="projects"	とプロジェクトののをする
@JoinColumn	のとみなされるエンティティをするのをします。
@JoinTable	キーをしてとプロジェクトのをするデータベースのテーブルをします。

- @TableGeneratorと@GeneratedValueは、jpaテーブルジェネレータをしたIDにされます。
- @ManyToManyアノテーションは、とプロジェクトエンティティのをします。
- @JoinTableは、joinテーブルjpaとしてするテーブルのをします.name = "employee_project"をしてEmployeeとProjectののマッピング。これは、データベーステーブルにのテーブルをするためのキーがまれていないため、jpaマッピングのをするがないためにわれます。
- @JoinColumnは、のとみなされるエンティティをするのをし、@inverseJoinColumnはののをします。 あなたはとみなされるようにのをぶことができます。ここではEmployeeをオーナーとしてしていますので、@JoinColumnはemployee_projectというジョインテーブルのidemployeeカラムをし、@InverseJoinColumnはjpaののidprojectをします。
- Projectエンティティの@ManyToManyアノテーションはのをしますので、mappedBy = projectsをしてEmployeeエンティティのフィールドをします。

なはここでできます

Examples

からプロジェクトへのマッピング

エンティティ。

```
package com.thejavageek.jpa.entities;

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.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.TableGenerator;

@Entity
public class Employee {

    @TableGenerator(name = "employee_gen", table = "id_gen", pkColumnName = "gen_name",
valueColumnName = "gen_val", allocationSize = 100)
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "employee_gen")
    private int idemployee;
    private String name;

    @ManyToMany(cascade = CascadeType.PERSIST)
    @JoinTable(name = "employee_project", joinColumns = @JoinColumn(name = "idemployee"),
inverseJoinColumns = @JoinColumn(name = "idproject"))
    private List<Project> projects;

    public int getIdemployee() {
        return idemployee;
    }

    public void setIdemployee(int idemployee) {
        this.idemployee = idemployee;
    }

    public String getName() {
        return name;
    }

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

    public List<Project> getProjects() {
        return projects;
    }

    public void setProjects(List<Project> projects) {
```

```
        this.projects = projects;
    }

}
```

プロジェクトエンティティ

```
package com.thejavageek.jpa.entities;

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.ManyToMany;
import javax.persistence.TableGenerator;

@Entity
public class Project {

    @TableGenerator(name = "project_gen", allocationSize = 1, pkColumnName = "gen_name",
valueColumnName = "gen_val", table = "id_gen")
    @Id
    @GeneratedValue(generator = "project_gen", strategy = GenerationType.TABLE)
    private int idproject;
    private String name;

    @ManyToMany(mappedBy = "projects", cascade = CascadeType.PERSIST)
    private List<Employee> employees;

    public int getIdproject() {
        return idproject;
    }

    public void setIdproject(int idproject) {
        this.idproject = idproject;
    }

    public String getName() {
        return name;
    }

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

    public List<Employee> getEmployees() {
        return employees;
    }

    public void setEmployees(List<Employee> employees) {
        this.employees = employees;
    }

}
```

テストコード

/ * EntityManagerFactoryをする* / EntityManagerFactory emf =.createEntityManagerFactory "JPAExamples";

```
    /* Create EntityManager */
    EntityManager em = emf.createEntityManager();

    EntityTransaction transaction = em.getTransaction();

    transaction.begin();

    Employee prasad = new Employee();
    prasad.setName("prasad kharkar");

    Employee harish = new Employee();
    harish.setName("Harish taware");

    Project ceg = new Project();
    ceg.setName("CEG");

    Project gtt = new Project();
    gtt.setName("GTT");

    List<Project> projects = new ArrayList<Project>();
    projects.add(ceg);
    projects.add(gtt);

    List<Employee> employees = new ArrayList<Employee>();
    employees.add(prasad);
    employees.add(harish);

    ceg.setEmployees(employees);
    gtt.setEmployees(employees);

    prasad.setProjects(projects);
    harish.setProjects(projects);

    em.persist(prasad);

    transaction.commit();
```

めみなアノテ―ションなしでキ―をする

あなたがっている

```
Role:
+-----------------------------+
| roleId | name | discription |
+-----------------------------+


Rights:
+-----------------------------+
| rightId | name | discription|
+-----------------------------+

rightrole
+------------------+
| roleId | rightId |
```

```
+------------------+
```

のシナリオでは、 `rightrole`テーブルにキーがあり、JPAユーザーにアクセスするには`Embeddable`アノテーションをしてエンティティをするがあります。

このような

ロールテーブルのエンティティはのとおりです。

```
    @Entity
    @Table(name = "rightrole")
    public class RightRole extends BaseEntity<RightRolePK> {

        private static final long serialVersionUID = 1L;

        @EmbeddedId
        protected RightRolePK rightRolePK;


        @JoinColumn(name = "roleID", referencedColumnName = "roleID", insertable = false,
updatable = false)
        @ManyToOne(fetch = FetchType.LAZY)
        private Role role;

        @JoinColumn(name = "rightID", referencedColumnName = "rightID", insertable = false,
updatable = false)
        @ManyToOne(fetch = FetchType.LAZY)
        private Right right;

        ......
     }


    @Embeddable
    public class RightRolePK implements Serializable {
    private static final long serialVersionUID = 1L;

      @Basic(optional = false)
      @NotNull
      @Column(nullable = false)
      private long roleID;

      @Basic(optional = false)
      @NotNull
      @Column(nullable = false)
     private long rightID;

   .....

}
```

めみなはオブジェクトのはありませんが、レコードをするにがします。

は、ユーザーが`rights role`つしい`role`をし、のユーザーが`role`オブジェクトを`store(persist)`があり、その、ユーザーが`flush`をして、しくされた`id`をロールとしてするがあるときです。ユーザーはそれを`rightrole`エンティティのオブジェクトにれることができます。

このユーザーをするには、エンティティをわずかになるできむことができます。

ロールテーブルのエンティティはのとおりです。

```
@Entity
@Table(name = "role")
public class Role {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @NotNull
    @Column(nullable = false)
    private Long roleID;


    @OneToMany(cascade = CascadeType.ALL, mappedBy = "role", fetch = FetchType.LAZY)
    private List<RightRole> rightRoleList;

    @ManyToMany(cascade = {CascadeType.PERSIST})
    @JoinTable(name = "rightrole",
            joinColumns = {
                @JoinColumn(name = "roleID", referencedColumnName = "ROLE_ID")},
            inverseJoinColumns = {
                @JoinColumn(name = "rightID", referencedColumnName = "RIGHT_ID")})
    private List<Right> rightList;
.......
}
```

@JoinTableは、エンティティがなくても`rightrole`テーブルにされますそのテーブルのroleとrightのidカラムのみがあるり。

ユーザーはにのことができます。

```
Role role = new  Role();
List<Right> rightList = new ArrayList<>();
Right right1 = new Right();
Right right2 = new Right();
rightList.add(right1);
rightList.add(right2);
role.setRightList(rightList);
```

inverseJoinColumnsに**@ManyToManycascade = {CascadeType.PERSIST}**をするがあります。そうしないと、がされるとデータがされます。

オンラインでマッピングをむ https://riptutorial.com/ja/jpa/topic/6532/マッピング

10: にしました

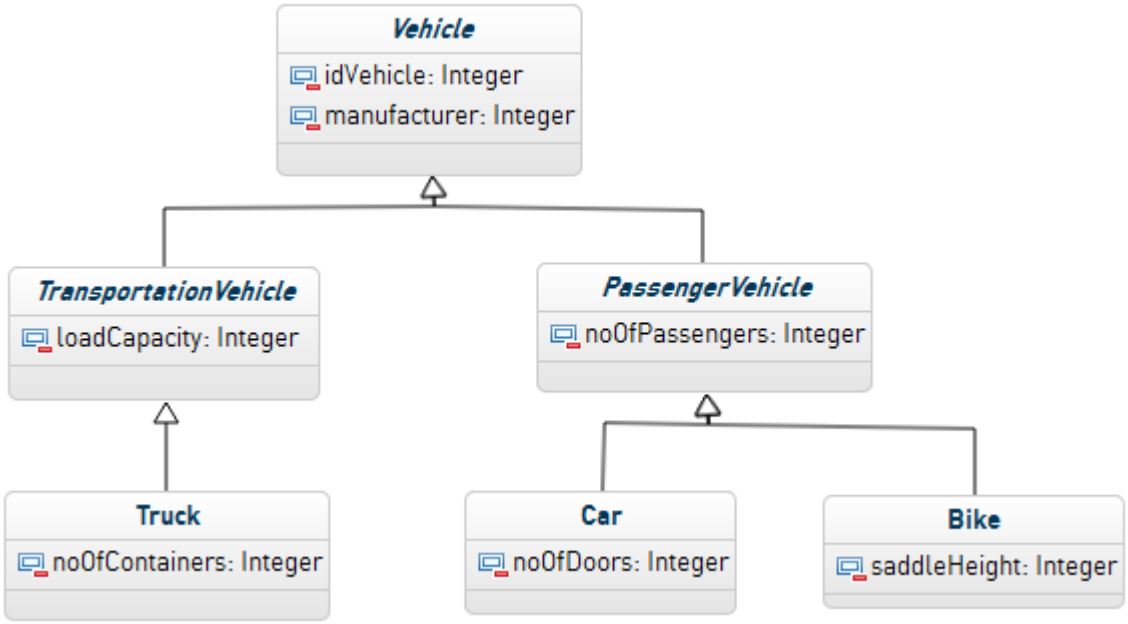
パラ メーター

@	されるストラテジのタイプをします。
@DiscriminatorColumn	エンティティにりてられたのIDにづいてなるエンティティをするためにされるデータベースのをします
@MappedSuperClass	マップされたスーパークラスはではなく、そのサブクラスのをするためにのみされます。にJavaクラスは@MapperSuperClassでマークされています

Examples

にしました

JPAのをするサンプルのクラス。




```
@Entity
@Table(name = "VEHICLE")
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name = "VEHICLE_TYPE")
public abstract class Vehicle {

    @TableGenerator(name = "VEHICLE_GEN", table = "ID_GEN", pkColumnName = "GEN_NAME",
valueColumnName = "GEN_VAL", allocationSize = 1)
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "VEHICLE_GEN")
    private int idVehicle;
    private String manufacturer;

    // getters and setters
}
```

TransportationVehicle.java

```
@MappedSuperclass
public abstract class TransportationVehicle extends Vehicle {

    private int loadCapacity;

    // getters and setters
}
```

Truck.java

```
@Entity
public class Truck extends TransportationVehicle {

    private int noOfContainers;

    // getters and setters

}
```

PassengerVehicle.java

```
@MappedSuperclass
public abstract class PassengerVehicle extends Vehicle {

    private int noOfpassengers;

    // getters and setters
}
```

Car.java

```
@Entity
public class Car extends PassengerVehicle {

    private int noOfDoors;

    // getters and setters
}
```

Bike.java

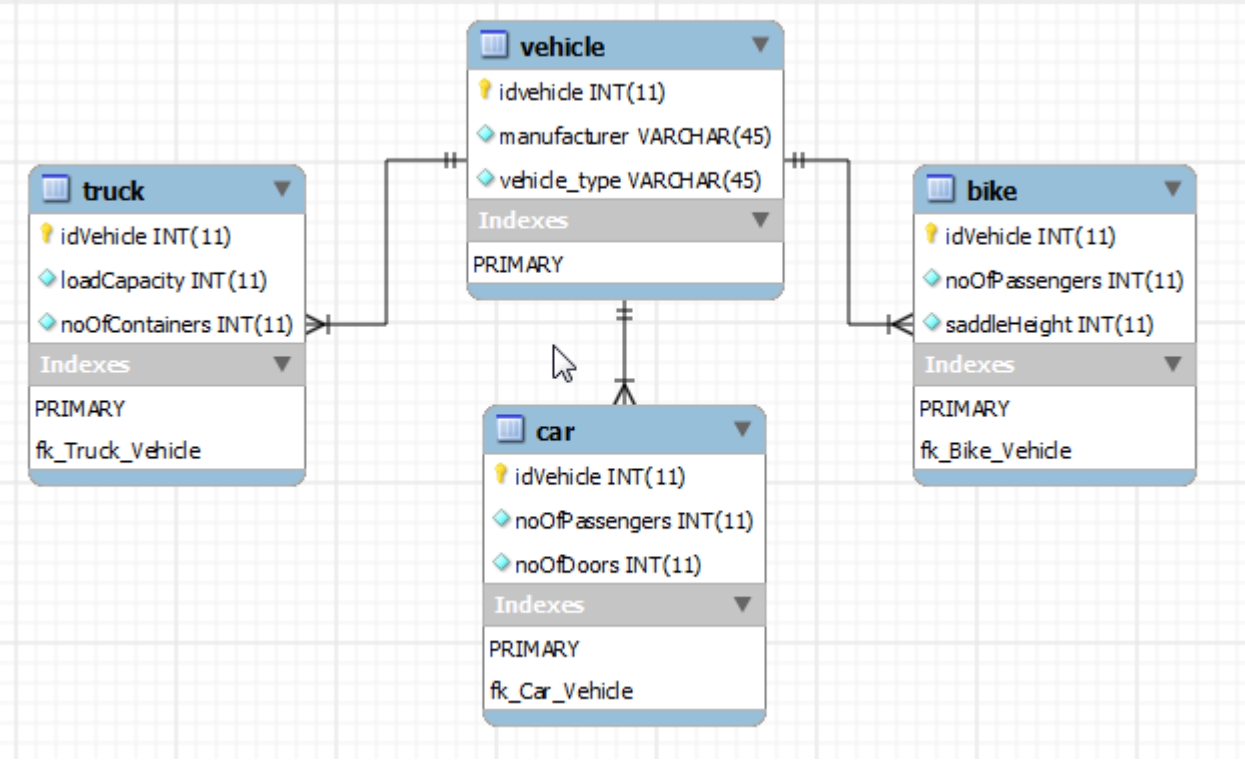
```
@Entity
public class Bike extends PassengerVehicle {

    private int saddleHeight;
```

```
    // getters and setters

}
```

テストコード

```
/* Create EntityManagerFactory */
EntityManagerFactory emf = Persistence
        .createEntityManagerFactory("AdvancedMapping");

/* Create EntityManager */
EntityManager em = emf.createEntityManager();
EntityTransaction transaction = em.getTransaction();

transaction.begin();

Bike cbr1000rr = new Bike();
cbr1000rr.setManufacturer("honda");
cbr1000rr.setNoOfpassengers(1);
cbr1000rr.setSaddleHeight(30);
em.persist(cbr1000rr);

Car aventador = new Car();
aventador.setManufacturer("lamborghini");
aventador.setNoOfDoors(2);
aventador.setNoOfpassengers(2);
em.persist(aventador);

Truck truck = new Truck();
truck.setLoadCapacity(1000);
truck.setManufacturer("volvo");
truck.setNoOfContainers(2);
em.persist(truck);

transaction.commit();
```

データベースダィアグラムはのようになります。



されたのは、テーブルののようにデータベースのスペースをしないことです。、とりしのたびにのがわれるため、がくくなるとパフォーマンスがになります。

きのなはここでむことができます

オンラインでにしましたをむ https://riptutorial.com/ja/jpa/topic/6473/にしました

クレジット

S. No		Contributors
1	jpaをいめる	Billy Frost, Community, DimaSan, , Manuel Spigolon, Michael Piefel, Neil Stockton, ppeterka
2	11マッピング	Prasad Kharkar
3	1の	Prasad Kharkar
4	エンティティの	DimaSan,
5	なクラスのの	Prasad Kharkar
6	テーブル	Prasad Kharkar
7	なマッピング	Jeffrey Brett Coleman, Michael Piefel, Neil Stockton, Pete
8	1マッピング	Prasad Kharkar
9	マッピング	Prasad Kharkar, Ronak Patel, Vetle
10	にしました	bw_üezi, Prasad Kharkar