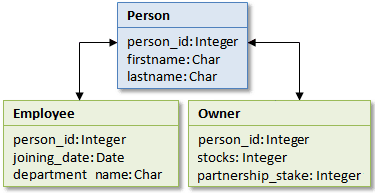
**Hibernate Inheritance ,** Java is an object oriented language. It is possible to implement Inheritance in Java. Inheritance is one of the most visible facets of Object-relational mismatch. Object oriented systems can model both

**“is a”** and **“has a”** relationship. Relational model supports only “has a” relationship between two entities. Hibernate can help you map such Objects with relational tables. But you need to choose certain mapping strategy based on your needs.

1. **Table Per Concrete Class**

Suppose we have a class Person with subclasses Customer and Owner. Following the class diagram and relationship of these classes.



In One Table per Concrete class scheme, each concrete class is mapped as normal persistent class. Thus we have 3 tables; PERSON, CUSTOMER and OWNER to persist the class data. In this scheme, the mapping of the subclass repeats the properties of the parent class.

Following are the advantages and disadvantages of One Table per Subclass scheme.

#### Advantages

* This is the easiest method of Inheritance mapping to implement.

#### Disadvantages

* Data thats belongs to a parent class is scattered across a number of subclass tables, which represents concrete classes.
* This hierarchy is not recommended for most cases.
* Changes to a parent class is reflected to large number of tables
* A query couched in terms of parent class is likely to cause a large number of select operations

This strategy has many drawbacks (esp. with polymorphic queries and associations) explained in the JPA spec, the Hibernate reference documentation, Hibernate in Action, and many other places. Hibernate work around most of them implementing this strategy using SQL UNION queries. It is commonly used for the top level of an inheritance hierarchy:

**Following is the example where we map Person, Customer and Owner entity classes using JPA Annotations**

**@Inheritance** – Defines the inheritance strategy to be used for an entity class hierarchy. It is specified on the entity class that is the root of the entity class hierarchy.

**@InheritanceType** – Defines inheritance strategy options. TABLE\_PER\_CLASS is a strategy to map table per concrete class.

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Inheritance;

**import** javax.persistence.InheritanceType;

**import** javax.persistence.Table;

@Entity

@Table(name = "PERSON")

@Inheritance(strategy = InheritanceType.***TABLE\_PER\_CLASS***)

**public** **class** Person {

@Id

@Column(name = "PERSON\_ID")

**private** Long personId;

@Column(name = "FIRSTNAME")

**private** String firstname;

@Column(name = "LASTNAME")

**private** String lastname;

**public** Person() {

}

//setter & getters

}

Both Employee and Owner classes are child of Person class. Thus while specifying the mappings, we used @AttributeOverrides to map them.

**@AttributeOverrides** – This annotation is used to override mappings of multiple properties or fields.

**@AttributeOverride** – The AttributeOverride annotation is used to override the mapping of a Basic (whether explicit or default) property or field or Id property or field.

The AttributeOverride annotation may be applied to an entity that extends a mapped superclass or to an embedded field or property to override a basic mapping defined by the mapped superclass or embeddable class. If the AttributeOverride annotation is not specified, the column is mapped the same as in the original mapping.

Thus in over case, firstname and lastname are defined in parent class Person and mapped in child class Employee and Owner using @AttributeOverrides annotation.

This strategy supports one-to-many associations provided that they are bidirectional. This strategy does not support the IDENTITY generator strategy: the id has to be shared across several tables. Consequently, when using this strategy, you should not use AUTO nor IDENTITY. Note that in below Main class we specified the primary key explicitly.

**import** javax.persistence.AttributeOverride;

**import** javax.persistence.AttributeOverrides;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Table;

@Entity

@Table(name = "CUSTOMER")

@AttributeOverrides({

@AttributeOverride(name = "firstname", column = @Column(name = "FIRSTNAME")),

@AttributeOverride(name = "lastname", column = @Column(name = "LASTNAME")) })

**public** **class** Customer **extends** Person {

@Column(name = "joining\_date")

**private** Date joiningDate;

@Column(name = "department\_name")

**private** String departmentName;

//setter & getters

}

**import** javax.persistence.AttributeOverride;

**import** javax.persistence.AttributeOverrides;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Table;

@Entity

@Table(name = "OWNER")

@AttributeOverrides({

@AttributeOverride(name = "firstname", column = @Column(name = "FIRSTNAME")),

@AttributeOverride(name = "lastname", column = @Column(name = "LASTNAME")) })

**public** **class** Owner **extends** Person {

@Column(name = "stocks")

**private** Long stocks;

@Column(name = "partnership\_stake")

**private** Long partnershipStake;

//setter & getters

**public** Owner() {

}

}

**public** **class** TablePerConcreteClassMain {

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

SessionFactory sf = HibernateConfig.*getSessionFactory*();

Session session = sf.openSession();

session.beginTransaction();

Person person = **new** Person("Steve", "Balmer");

person.setPersonId(1L);

session.save(person);

Customer customer = **new** Customer("James", "Gosling", "Marketing",

**new** Date());

customer.setPersonId(2L);

session.save(customer);

Owner owner = **new** Owner("Bill", "Gates", 300L, 20L);

owner.setPersonId(3L);

session.save(owner);

session.getTransaction().commit();

session.close();

}

}

The Main class is used to persist Person, Customer and Owner object instances. Note that these classes are persisted in different tables and parent attributes (firstname, lastname) are repeated across all tables.

Hibernate: insert into PERSON (FIRSTNAME, LASTNAME, PERSON\_ID) values (?, ?, ?)

Hibernate: insert into EMPLOYEE (FIRSTNAME, LASTNAME, department\_name, joining\_date, PERSON\_ID) values (?, ?, ?, ?, ?)

Hibernate: insert into OWNER (FIRSTNAME, LASTNAME, partnership\_stake, stocks, PERSON\_ID) values (?, ?, ?, ?, ?)