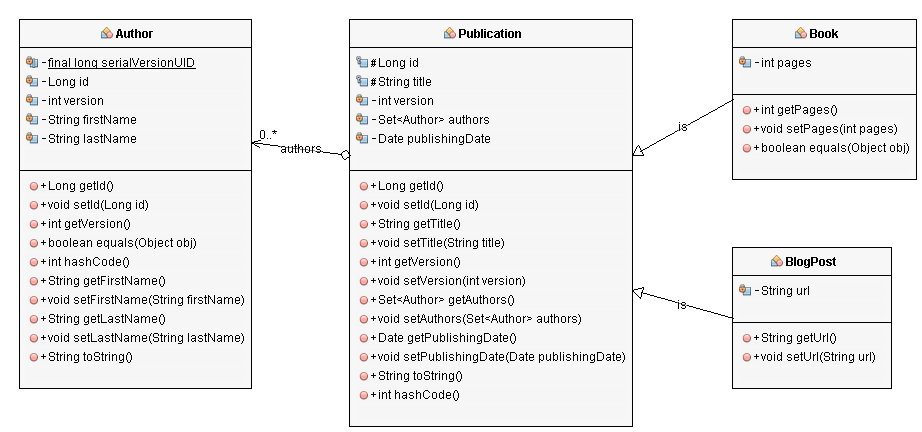
**Inheritance mapping**

Inheritance (kế thừa) được hầu hết trong các domain model. Nó thường xuyên xảy ra vấn đề khi cố gắng map những model đó tới các quan hệ database. SQL không hỗ trợ loại quan hệ này và Hibernate hoặc bất cứ JPA nào sẽ thực hiện map nó.

Bạn có thể chọn 4 loại inheritance mapping của domain model của bạn. Mỗi loại map sẽ có ưu điểm và nhược điểm của nó

**Domain model**

Có 1 domain model đơn giản như hình. Nó gồm 1 bảng *Author* người viết những loại *Publication* khác nhau. 1 *Publication* có thể là 1 *Boo*k hoặc 1 *BlogPos*t. Cả 2 đều có chung những thuộc tính như: id, title, version, và publishing date, trong đó *Book* có thêm số trang sách (pages) và *BlogPost* có thêm url của nó.

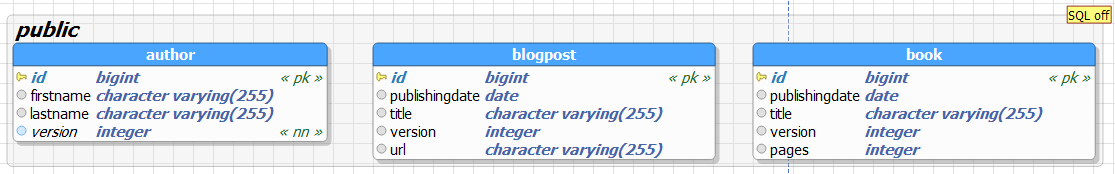


**4 Inheritance Strategies**

JPA và Hibernate hỗ trợ 4 loại inheritance mapping sẽ map những đối tượng domain tới những cấu trúc bảng khác nhau.

**Mapped Supperclass**

Mappes superclass là loại map gần như đơn giản nhất. Nó map mỗi class con tới bảng của chính nó với các column của lớp cha và con.



Kiểu map này cho phép bạn chia sẻ thuộc tính giữa nhiều entities. Nhưng nó cũng có 1 nhược điểm lớn , đó là lớp cha không phải 1 entity, không có bảng nào cho nó. Bạn không thể dùng câu lệnh đa hình (*polymorphic query*) để select tất cả Publication entity, và cũng không thể định nghĩa quan hệ 1 giữa thực thể Author và tất cả Publication. Cách giải quyết là định nghĩa quan hệ giữa 1 Author và mỗi loại Publication (Book và BlogPost).

Nếu chỉ muốn chia sẻ trạng thái và map thông tin giữa các entity thì mapped superclass là phù hợp và dễ dàng thực hiện. Bạn chỉ phải tạo cấu trúc kế thừa, đặt annotation cho các thuộc tính như bình thường và đặt *@MappedSuperClass* vào lớp cha và không cần *@Entity*. Nếu không có, hibernate sẽ bỏ qua thông tin mapping của lớp cha.

@MappedSuperclass  
**public abstract class** Publication {  
 @Id @GeneratedValue(strategy = GenerationType.***AUTO***)  
 @Column(name = **"id"**, nullable = **false**)  
 **private int id**;  
  
 @Column  
 **private** String **title**;  
  
 @Version  
 @Column(name = **"version"**)  
 **private int version**;  
  
 @Column  
 @Temporal(TemporalType.***DATE***)  
 **private** Date **publishingDate**;  
}

Với các class con Book và BlogPost kế thừa Publication:

@Entity(name = **"Book"**)  
**public class** Book **extends** Publication {  
 @Column  
 **private int pages**;

...  
}

@Entity(name = **"BlogPost"**)  
**public class** BlogPost **extends** Publication {  
 @Column  
 **private** String **url**;  
 ...  
}

Như đã giải thích thì không thể sử dụng polymorphic query cho loại map này hoặc định nghĩa các quan hệ nhưng vẫn có thể tạo query cho các entity giống như bình thường.

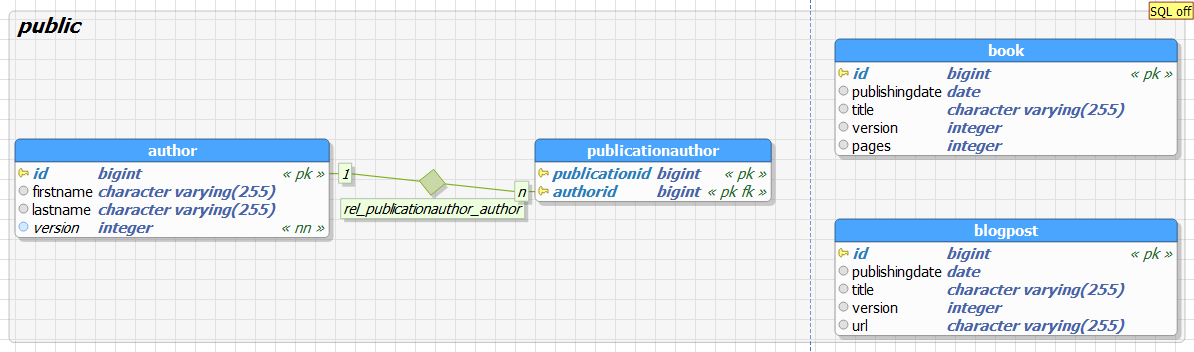
List listBook = session.createQuery(**"FROM Book"**).list();

Book entity và tất cả các thuộc tính của nó được map với bảng book. Điều này làm câu lệnh truy vấn trở nên đơn giản và hiệu quả, chỉ cần select tất cả các cột trong bảng book.

Hibernate: select book0\_.id as id1\_, book0\_.publishingDate as publishi2\_1\_, book0\_.title as title1\_, book0\_.version as version1\_, book0\_.pages as pages1\_

**Table per Class**

Loại map table per class tương đối giống với mapped superclass. Điểm khác nhau chính là lớp cha giờ sẽ là 1 entity. Mỗi class cụ thể vẫn được map bởi chính table của nó. Kiểu map này cho phép sử dụng polymorphic queries và định nghĩa quan hệ tới lớp cha. Nhưng cấu trúc bảng sẽ thêm rất nhiều câu lệnh polymorphic phức tạp và nên tránh nó.



Định nghĩa lớp cha với kiểu map table per class tương tự như các kiểu định nghĩa entity khác. Thêm *@Entity* và thêm các annotation cho các thuộc tính. Điểm khác nhau duy nhất là thêm *@Inheritance* vào class để định nghĩa kiểu inheritance. Trong trường hợp này là *InheritanceType.TABLE\_PER\_CLASS*. Định nghĩa entity Book và Blogpost giống hệt với kiểu mapped superclass.

@Entity  
@Inheritance(strategy = InheritanceType.***TABLE\_PER\_CLASS***)  
**public abstract class** Publication {  
 @Id @GeneratedValue  
 **private int id**;  
  
 @Column  
 **private** String **title**;  
  
 @Version  
 @Column  
 **private int version**;  
  
 @ManyToMany(cascade = CascadeType.***ALL***)  
 @JoinTable(name = **"PublicationAuthor"**, joinColumns = {@JoinColumn(name =

**"publicationId"**,

referencedColumnName = **"id"**)},  
 inverseJoinColumns = {@JoinColumn(name = **"authorId"**,

referencedColumnName = **"id"**)})  
 **private** Set **authors**;  
  
 @Column  
 @Temporal(TemporalType.***DATE***)  
 **private** Date **publishingDate**;

}

Lớp cha giờ cũng là 1 entity và có thể định nghĩa quan hệ giữa Author và Publication. Điều này cho phép gọi phương thức *getPublications()* để có thể lấy tất cả Publication viết bởi Author.

Author author = (Author) session.get(Author.**class**, 1);  
**for**(Publication publication : author.getPublications()){  
 **if** (publication **instanceof** Book){  
 System.***out***.println(((Book) publication).getPages());  
 }  
}

Java code nhìn đơn giản và dễ dàng sử dụng nhưng nếu nhìn vào câu lệnh SQL, có thể nhận ra những bảng model khiến cho câu lệnh select khá phức tạp.

Hibernate: select author0\_.id as id0\_0\_, author0\_.firstname as firstname0\_0\_, author0\_.lastname as lastname0\_0\_, author0\_.version as version0\_0\_ from Author author0\_ where author0\_.id=?

Hibernate: SELECT publicatio0\_.authorId AS authorId0\_1\_,

publicatio0\_.publicationId AS publicat1\_1\_,

publicatio1\_.id AS id1\_0\_,

publicatio1\_.publishingDate AS publishi2\_1\_0\_,

publicatio1\_.title AS title1\_0\_,

publicatio1\_.version AS version1\_0\_,

publicatio1\_.url AS url2\_0\_,

publicatio1\_.pages AS pages3\_0\_,

publicatio1\_.clazz\_ AS clazz\_0\_

FROM PublicationAuthor publicatio0\_ **INNER JOIN** **(** SELECT id, publishingDate, title,

VERSION, url, NULL AS pages, 1 AS clazz\_

**FROM BlogPost**

**UNION**

**SELECT** id, publishingDate, title,

VERSION, NULL AS url, pages, 2 AS clazz\_

**FROM Book** ) publicatio1\_ ON publicatio0\_.publicationId=publicatio1\_.id

WHERE publicatio0\_.authorId=?

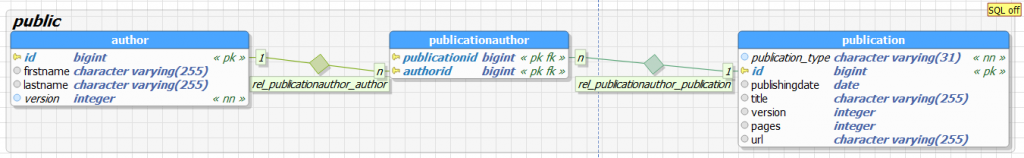
Hibenate phải nối bảng author với kết quả của câu lệnh select bảng con dùng *UNION* để lấy tất cả bản ghi phù hợp từ bảng book và blogpost. Dựa vào số lượng của bản ghi của cả 2 bảng, câu lệnh này có thể trở thành vấn đề. Nó sẽ trở nên khó khăn nếu có nhiều class con hơn trong kế thừa. Vì vậy nên tránh những loại câu lệnh này hoặc chọn 1 loại inheritance mapping khác.

**Single Table**

Loại map single table sẽ map tất cả các entity của cấu trúc kế thừa với cùng 1 bảng database. Điều này làm cho câu lệnh polymorphic query trở nên hiệu quả và cung cấp sự thực thi tốt nhất.

Nhưng nó cũng có vài nhược điểm. Các thuộc tính của tất cả các entity được map tới cùng 1 bảng. Mỗi bản ghi chỉ sử dụng 1 tập con của các cột phù hợp và đặt cột còn lại thành null. Vì vậy không thể dùng *not null* trên bất cứ cột nào không được map với tất cả các entity.

Điều này có thể tạo ra vấn đề không toàn vẹn do 1 thuộc tính được mô tả và lặp lại trong nhiều bản ghi dữ liệu khác nhau, khi cập nhật sửa đổi sẽ không sửa hết nội dung các mục đó. Nếu dữ liệu càng nhiều thì sự sai sót khi cập nhật, bổ sung càng lớn dẫn đến mâu thuẫn, không nhất quán trong lưu trữ dữ liệu.



Khi lưu tất cả các entity vào trong cùng 1 bảng, Hibernate cần 1 phương pháp để xác định lớp entity mỗi bản ghi trình chiếu. Đây là thông tin được lưu trong 1 cột dùng để phân biệt mà không phải là 1 thuộc tính entity. Có thể định nghĩa tên của cột đó với *@DiscriminatorColumn* annotation trên lớp cha hoặc Hibernate sẽ sử dụng *DTYPE* như 1 tên mặc định.

@Entity  
@Inheritance(strategy = InheritanceType.***SINGLE\_TABLE***)  
@DiscriminatorColumn(name = **"publication\_type"**)  
**public abstract class** Publication {  
  
 @Id @GeneratedValue  
 **private int id**;  
  
 @Column  
 **private** String **title**;  
  
 @Column  
 **private int version**;  
  
 @ManyToMany(cascade = CascadeType.***ALL***)  
 @JoinTable(name = **"PublicationAuthor"**,joinColumns = @JoinColumn(name = **"publicationId"**),  
 inverseJoinColumns = @JoinColumn(name = **"authorId"**))  
 **private** Set<Author> **authors**;  
  
 @Column  
 @Temporal(TemporalType.***DATE***)  
 **private** Date **publishingDate**;

}

Định nghĩa lớp con giống như ví dụ trước, nhưng trong trường hợp này nên cung cấp thêm 1 annotation *@DiscriminatorValue*. Nó chỉ rõ giá trị để phân biệt cho lớp entity cụ thể. Nếu không sử dụng discrminator value, Hibernate sẽ sử dụng tên của entity để mặc định.

@Entity  
@DiscriminatorValue(**"BlogPost"**)  
**public class** BlogPost **extends** Publication{  
 @Column  
 **private** String **url**;

}

@Entity  
@DiscriminatorValue(**"Book"**)  
**public class** Book **extends** Publication{  
  
 @Column  
 **private int pages**

}

Như đã giải thích phần đầu, loại map single table cho phép truy cập tới data dễ dàng và hiệu quả. Tất cả các thuộc tính của mỗi entity được lưu trong 1 bảng, và query sẽ không yêu cầu bất cứ lệnh join nào. Thứ duy nhất Hibernate cần để thêm vào SQL query để lấy 1 lớp entity cụ thể là giá trị phân biệt **discriminator value**. Trong trường hợp này là giá trị của bảng *publication\_type* chứa giá trị *“Book”.*

Hibernate: select book0\_.id as id1\_, book0\_.publishingDate as publishi3\_1\_, book0\_.title as title1\_, book0\_.version as version1\_, book0\_.pages as pages1\_ from Publication book0\_ where *book0\_.publication\_type='Book'*

List listBooks = session.createQuery(**"from Book"**).list();

Phần inheritance mapping trước đã xảy ra vấn đề với polymorphic quries, vừa không được hỗ trợ vừa có yêu cầu phức tạp với UNION để nối. Nó được giải quyết nếu sử dụng single table strategy. Tất cả entity trong hệ thống kế thừa được map tới 1 bảng duy nhất và có thể truy vấn với câu lệnh đơn giản.

Hibernate: select author0\_.id as id0\_0\_, author0\_.firstname as firstname0\_0\_, author0\_.lastname as lastname0\_0\_, author0\_.version as version0\_0\_ from Author author0\_ where author0\_.id=?

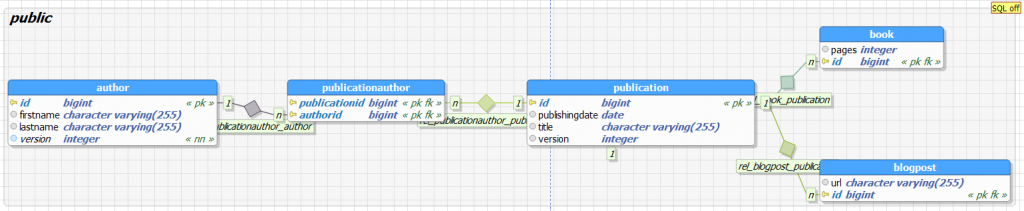
Hibernate: select publicatio0\_.authorId as authorId0\_1\_, publicatio0\_.publicationId as publicat1\_1\_, publicatio1\_.id as id1\_0\_, publicatio1\_.publishingDate as publishi3\_1\_0\_, publicatio1\_.title as title1\_0\_, publicatio1\_.version as version1\_0\_, publicatio1\_.url as url1\_0\_, publicatio1\_.pages as pages1\_0\_, publicatio1\_.publication\_type as publicat1\_1\_0\_ **from** **PublicationAuthor** publicatio0\_ **inner join** **Publication** publicatio1\_ on publicatio0\_.publicationId=publicatio1\_.id **where** publicatio0\_.authorId=?

510

Author author1 = (Author) session.get(Author.**class**, 1);  
**for**(Publication p : author1.getPublications()){  
 **if** (p **instanceof** Book){  
 System.***out***.println(((Book) p).getPages());  
 }  
}

**Joined**

Kiểu map nối bảng gần như map mỗi class của sự kế thừa tới bảng db của chính nó. Nghe có vẻ giống table per class nhưng lần này cả lớp abstract cha Publication cũng được map tới 1 bảng database. Bảng này gồm những cột cho tất cả những thuộc tính dùng chung của entity. Các bảng của class con thì sẽ bé hơn trong table per class. Nó chỉ giữ những cột chứa thuộc tính riêng của từng class và khóa chính cùng giá trị với bản ghi trong bảng của lớp cha.



Mỗi query của lớp con yêu cầu 1 sự kết nối của 2 bảng để select được những cột của tất cả các thuộc tính entity. Điều này sẽ tăng độ phức tạp của mỗi query, nhưng nó cũng cho phép để sử dụng *not null* trên các thuộc tính lớp con để đảm bảo tính toàn vẹn dữ liệu. Định nghĩa của lớp cha Publication giống như ví dụ trước, điểm khác nhau duy nhất là giá trị của loại inheritance strategy là *InheritanceType.JOINED*.

Định nghĩa của lớp con không yêu cầu thêm bất cứ annotation nào khác, chỉ cần kế thừa lớp cha, cung cấp @Entity annotation và định nghĩa mapping của mỗi thuộc tính cụ thể trong class.

@Entity  
**public class** BlogPost **extends** Publication {  
 @Column  
 **private** String **url**;

}

@Entity  
**public class** Book **extends** Publication{  
 @Column  
 **private int pages**;

}

**public class** Publication {  
 @Id @GeneratedValue  
 **private int id**;  
  
 @Column  
 **private** String **title**;  
  
 @Column  
 **private int version**;  
  
 @ManyToMany(cascade = CascadeType.***ALL***)  
 @JoinTable(name = **"PublicationAuthor"**, joinColumns = {@JoinColumn(name = **"publicationId"**)},  
 inverseJoinColumns = @JoinColumn(name = **"authorId"**))  
 **private** Set<Author> **authors**;  
  
 @Column  
 @Temporal(TemporalType.***DATE***)  
 **private** Date **publishingDate**;

}

Như đã giải thích, các cột được map bởi mỗi lớp con được lưu trong 2 bảng db khác nhau. Bảng publication chứa tất cả các cột được map bởi class cha Publication và tất cả các cột trong bảng book được map thực thể Book. Hibernate cần phải nối 2 bảng bằng khóa chính để lấy được tất cả các thuộc tính của Book entity. Điều này làm các query chậm hơn 1 chút so với query đơn giản của loại map single table.

Hibernate: select author0\_.id as id0\_0\_, author0\_.firstname as firstname0\_0\_, author0\_.lastname as lastname0\_0\_, author0\_.version as version0\_0\_ from Author author0\_ where author0\_.id=?

Hibernate: select publicatio0\_.authorId as authorId0\_1\_, publicatio0\_.publicationId as publicat1\_1\_, publicatio1\_.id as id1\_0\_, publicatio1\_.publishingDate as publishi2\_1\_0\_, publicatio1\_.title as title1\_0\_, publicatio1\_.version as version1\_0\_, publicatio1\_1\_.pages as pages2\_0\_, publicatio1\_2\_.url as url3\_0\_, case when publicatio1\_1\_.id is not null then 1 when publicatio1\_2\_.id is not null then 2 when publicatio1\_.id is not null then 0 end as clazz\_0\_ **from** PublicationAuthor publicatio0\_ **inner join** Publication publicatio1\_ on publicatio0\_.publicationId=publicatio1\_.id **left outer join** Book publicatio1\_1\_ on publicatio1\_.id=publicatio1\_1\_.id **left outer join** BlogPost publicatio1\_2\_ on publicatio1\_.id=publicatio1\_2\_.id where publicatio0\_.authorId=?

510

Author author1 = (Author) session.get(Author.**class**, 1);  
**for** (Publication p : author1.getPublications()){  
 **if** (p **instanceof** Book){  
 System.***out***.println(((Book) p).getPages());  
 }  
}

**Lựa chọn loại strategy**

Lựa chọn đúng loại map inheritance không phải 1 nhiệm vụ dễ dàng. Thông thường, người dùng sẽ phải lựa chọn ưu điểm nào người dùng cần và nhược điểm nào có thể chấp nhận cho ứng dụng đó.

* Nếu người dùng yêu cầu việc thực thi tốt nhất, cần sử dụng polymorphic queries và các quan hệ thì nên chọn loại map single table. Nhưng phải lưu ý rằng không thể sử dụng ràng buộc not null trên các thuộc tính của class con và sẽ làm tăng nguy cơ mâu thuẫn dữ liệu (data inconsistency).
* Nếu mâu thuẫn dữ liệu quan trọng hơn thực thi và cần sử dụng polymorphic queies và các quan hệ, loại joined strategy chắc chắn là lựa chọn tốt nhất.
* Nếu không cần polymorphic queries hoặc các quan hệ, loại table per class strategy sẽ là lựa chọn phù hợp nhất. Nó cho phép sử dụng các ràng buộc để đảm bảo toàn vẹn dữ liệu và cung cấp 1 lựa chọn của polymorphic queries. Nhưng phải lưu ý, polymorphic queries đó rất phức tạp cho cấu trúc bảng này và nên tránh nó.