06 상속관계

Inheritance Strategies with JPA and Hibernate

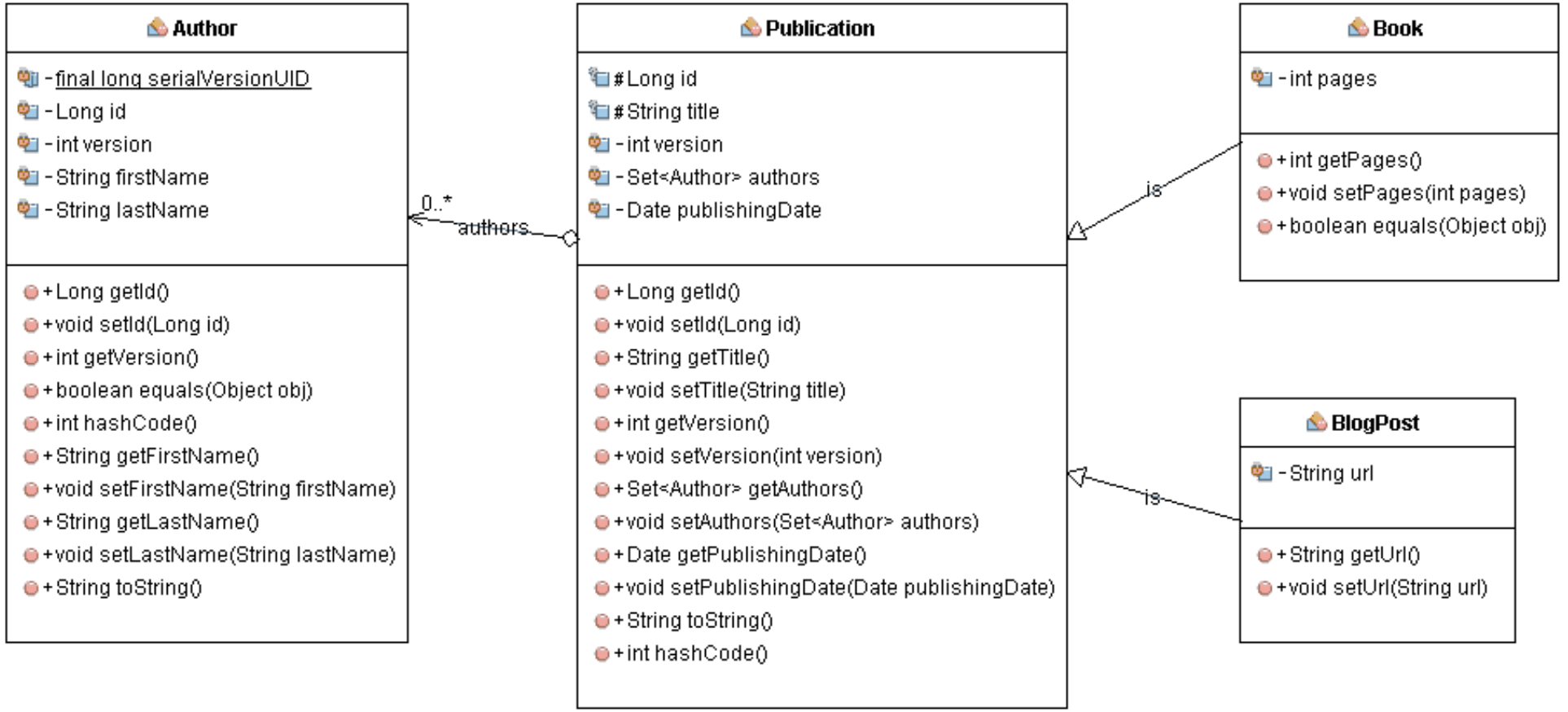
https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

소개

- **상속**
  - 상속은 자바에서 중요한 개념
  - 그러나 RDB에서는 상속이라는 개념이 없음
  - SQL은 상속관계를 위한 구문을 지원하지 않음
  - 그렇다면 RDB를 표현하는 자바객체는 무조건 상속 개념을 사용할 수 없는가?
- **JPA가 제공하는 상속의 네 가지 전략**
  - Mapped Superclass
  - Table per Class
  - Single Table
  - Joined

Domain Model

- 저자(Author)는 여러 건의 글을 게재할 수 있다
- 글은 책(Book)과 블로그(BlogPost)로 구분된다
- Book과 BlogPost는 공통사항(Publication)과 고유사항이 있다.



논리적인 모델

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Mapped Superclass

- **JPA와 Hibernate가 제공하는 상속의 네 가지 전략**
  - Mapped Superclass
  - Table per Class
  - Single Table
  - Joined

- **Mapped Superclass**
  - 가장 간단한 전략
  - 각각의 구현클래스(book과 blogpost)를 개별 테이블에 매핑하는 전략

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Mapped Superclass

- **Mapped Superclass(@Entity가 없다)**

```java
@MappedSuperclass
public abstract class Publication {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", updatable = false, nullable = false)
    protected Long id;

    @Column
    protected String title;

    @Version
    @Column(name = "version")
    private int version;

    …
}
```

```java
@Entity(name = "Book")
public class Book extends
Publication {

    @Column
    private int pages;

    …
}
```

```java
@Entity(name = "BlogPost")
public class BlogPost extends
Publication {

    @Column
    private String url;

    …
}
```

DDL 확인

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

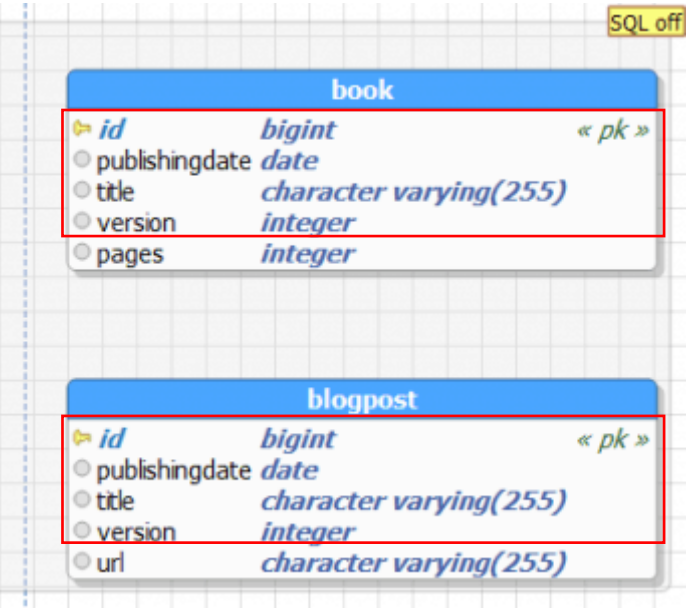
Mapped Superclass

- **Mapped Superclass**
  - 다수의 엔티티가 공통 속성을 공유할 수 있음
  - PK의 경우, 부모 클래스에 지정할 수 있지만 자식 클래스에 지정할 수도 있음
  - mapped superclass가 지정된 클래스는 엔티티가 아니므로 테이블에 매핑되는 것은 아님
  - 이는 다형성을 이용한 쿼리 사용이 어렵다는 의미
    - 예를 들어 blogPost와 Book 종류에 상관 없이 모든 Publication에 대한 쿼리를 실행할 수 없음
  - Author는 blogPost와 Book이라는 두 가지를 구분하여 각각의 연관관계를 유지해야 함
  - 따라서 주로 createdBy, createdOn과 같이 테이블에서 공통적으로 사용되는 반복되는 필드를 각 테이블에 끼워 넣을 때 사용
  - 상속개념을 사용하기 위해서는 뒤에 나오는 세 가지의 inheritance 전략을 고려해야 함

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Inheritance

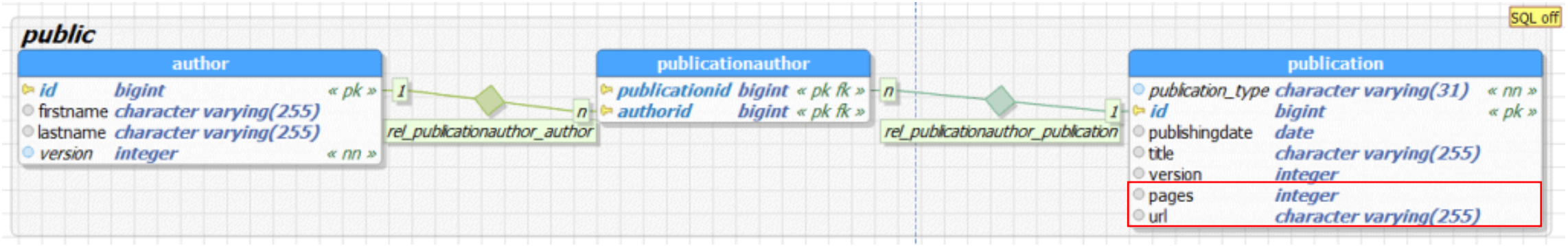
- **Table per Class**
  - mapped superclass 전략과 유사(mapped superclass와 같이 blogPost와 Book 테이블 각각을 생성)
  - 가장 큰 차이점은 superclass 또한 엔티티가 됨
  - 권장되는 방법은 아님
  - 공통사항(Publication)이 여기에도 있고 저기에도 있는 중복적인 구조



https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Inheritance

- **Single Table**
  - BlogPost와 Book을 하나의 테이블에 저장하는 방식
  - 운영이 간편
  - 하나의 테이블만 조회하면 되므로 성능 측면에서 가장 유리
  - Null허용 필드가 많아지는 단점이 존재
  - Null 허용과 데이터 무결성은 trade-off관계
  - DBA가 반가워하지 않는 상황



https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Inheritance

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = “Publication_Type”)
public abstract class Publication {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   @Column(name = “id”, updatable = false, nullable = false)
   protected Long id;

   @Column
   protected String title;

   @Version
   @Column(name = “version”)
   private int version;

}
```

```
@Entity(name = “Book”)
@DiscriminatorValue(“Book”)
public class Book extends Publication {

   @Column
   private int pages;

   …
}
```

```
@Entity(name = “BlogPost”)
@DiscriminatorValue(“Blog”)
public class BlogPost extends Publication {

   @Column
   private String url;

   …
}
```

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

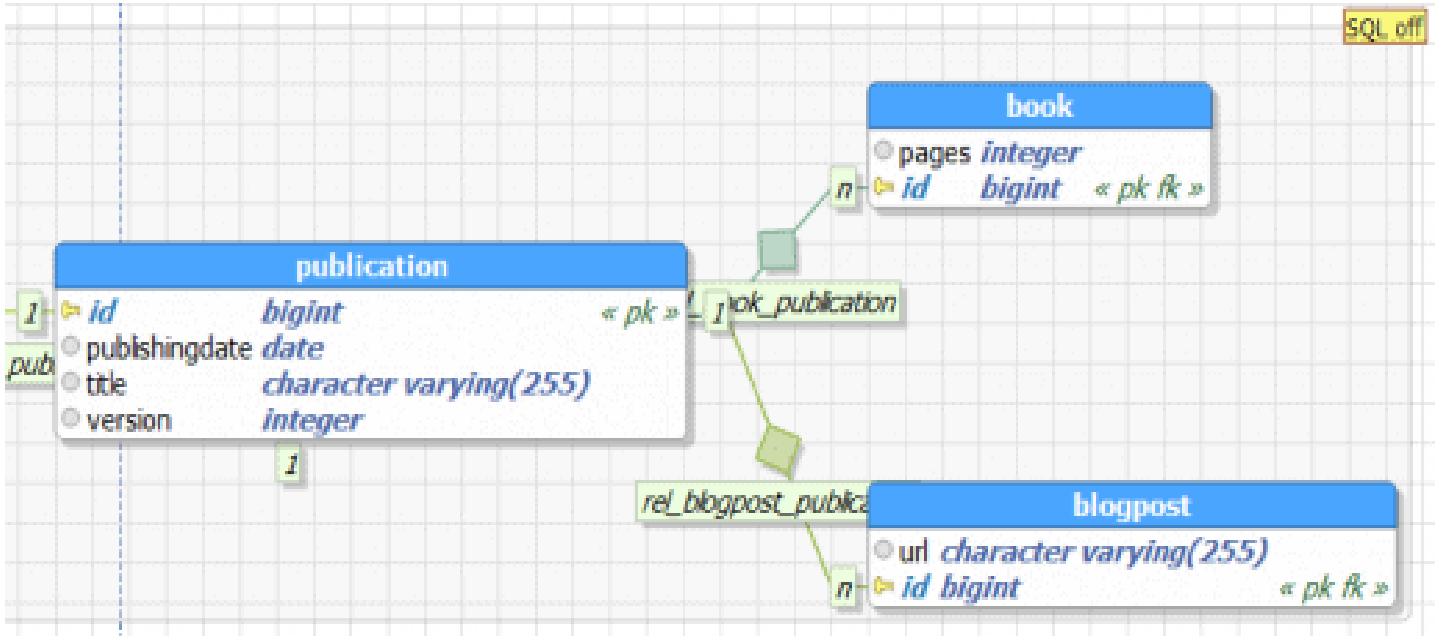
Inheritance

- **Single Table**
  - @DiscriminatorColumn(name = “Publication_Type”)
    - 현재의 레코드가 BlogPost에 관한 것인지, Book에 관한 것인지를 구분하기 위해 추가되는 필드
    - 하나의 테이블에 두 가지 주제가 포함되어 있으므로 구분을 위한 추가 필드가 필요
    - 참고) RDB에서는 하나의 테이블은 하나의 주제만 표현하는 것을 권장
  - @DiscriminatorValue
    - DiscriminatorColumn에 표시될 값
    - 예제에서는 각각 Book과 Blog로 표시
  - 값 비싼 연산으로 인식되는 JOIN연산 없이 " Publication과 Book" 혹은 " Publication과 BlogPost " 내용을 조회할 수 있음

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Inheritance

- **Joined**
  - BlogPost와 Book 테이블을 따로 관리
  - Table per Class와 유사한 구조
  - 가장 큰 차이점 → abstract superclass도 DB 테이블로 매핑(모든 공유 속성을 포함하는 테이블)
    - Publication도 테이블을 가짐



https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Inheritance

- **Joined**

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Publication {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = “id”, updatable = false, nullable = false)
    protected Long id;

    @Column
    protected String title;

    @Version
    @Column(name = “version”)
    private int version;

    @Column
    @Temporal(TemporalType.DATE)
    private Date publishingDate;

    …
}
```

- subclass조회에는 superclass와의 join이 반드시 필요
→ 쿼리 복잡도 증가

- subclass 에 NOT NULL 제약 조건을 사용할 수 있다는 장점
- DiscriminatorColumn을 사용하지 않아도 됨

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

정리

- **어떤 전략을 쓸 것인가**
  - 성능적으로 중요하고 쿼리 다형성이 필요하다면 → single table
    - table의 크기가 엄청나게 커진다면 항상 좋은 성능을 보인다는 장담을 하지는 못함
    - 또한 NOT NULL 제약조건을 쓸 수 없다는 것은 무결성 관점에서 좋지 않다는 것을 상기해야 함
  - 무결성, 일치성이 중요하다면 → joined
  - 쿼리나 연관관계에서 다형성이 필요없다면 → table per class
    - 위 예제에서 Publication에 대한 쿼리를 실행할 수 없으며 이를 위한 쿼리가 복잡해짐
  - 따라서 single table 혹은 joined이 가장 현실적인 방법

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/