

Object Relational Mapping < ORM > Bagian Pertama



IS 701 PENGEMBANGAN APLIKASI ENTERPRISE

**(C) 2010 NIKO IBRAHIM
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN MARANATHA**

Konsep ORM



- Object-relational mapping (ORM) berfungsi untuk memetakan entitas menjadi tabel dan atribut menjadi kolom tabel
- Pemetaan dapat dikategoriikan menjadi 4, yaitu:
 - Basic mapping
 - Relationship mapping
 - Composition mapping
 - Inheritance mapping

Contoh Entitas: Book.java



@Entity

```
public class Book {
```

@Id

```
    private Long id;
```

```
    private String title;
```

```
    private Float price;
```

```
    private String description;
```

```
    private String isbn;
```

```
    private Integer nbOfPage;
```

```
    private Boolean illustrations;
```

```
    public Book() {
```

```
    }
```

```
    // Getters, setters
```

```
}
```

Hasil Pemetaan

Book Class → Book Table (JavaDB)



```
CREATE TABLE BOOK (  
    ID BIGINT NOT NULL,  
    TITLE VARCHAR(255),  
    PRICE DOUBLE(52, 0),  
    DESCRIPTION VARCHAR(255),  
    ISBN VARCHAR(255),  
    NBOFPAGE INTEGER,  
    ILLUSTRATIONS SMALLINT,  
    PRIMARY KEY (ID)  
);
```

Anotasi Dasar



- Anotasi dasar berguna untuk meng-customise tabel, primary key, dan kolom.
- Berikutnya kita akan melihat teknik anotasi dasar yang meliputi:
 - Tables
 - Primary Keys
 - Attributes
 - Access Type
 - Collection of Basic Types
 - Map of Basic Types

@Table



- Anotasi `@javax.persistence.Table` memungkinkan kita untuk mengubah nilai default suatu tabel.
- Misalnya: kita dapat menentukan NAMA TABEL yang akan disimpan di database. Apabila anotasi `@Table` ini dihilangkan, maka nama tabel akan diambil dari nama entitas.

```
@Entity
@Table(name = "t_book")
public class Book {
    @Id
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    public Book() {
    }
    // Getters, setters
}
```

@SecondaryTable & @SecondaryTables



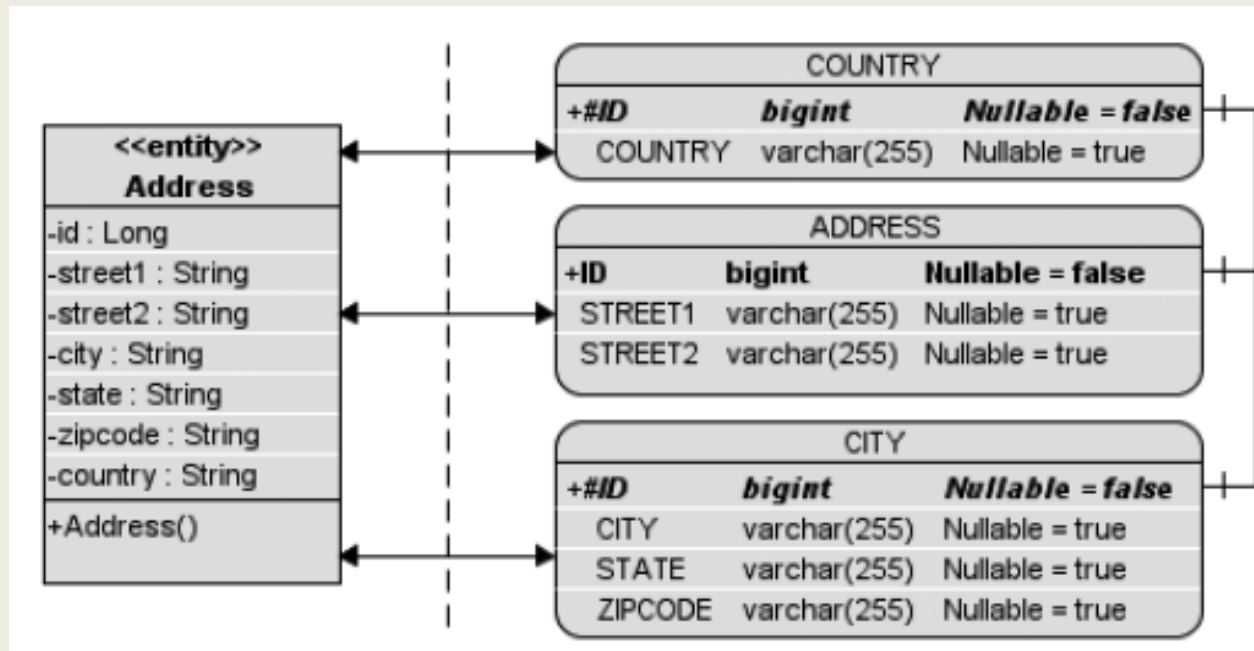
- Kadang-kadang, apabila kita perlu melakukan penyebaran data ke dalam banyak tabel.
- **@SecondaryTable** berguna untuk menghubungkan tabel tambahan dengan suatu entitas.
- **@SecondaryTables** (ada 's') berguna apabila tabel tambahannya ada lebih dari satu.

Contoh Penggunaan SecondaryTable

```
@Entity
@SecondaryTables({
    @SecondaryTable(name = "city"),
    @SecondaryTable(name = "country")
})
public class Address {
    @Id
    private Long id;
    private String street1;
    private String street2;
    @Column(table = "city")
    private String city;
    @Column(table = "city")
    private String state;
    @Column(table = "city")
    private String zipcode;
    @Column(table = "country")
    private String country;
    // Constructors, getters, setters
}
```

Hasil Pemetaan

1 Entitas → 3 Tabel



PrimaryKey



- JPA mewajibkan entitas untuk memiliki suatu *identifier* yang akan dipetakan menjadi primary key.
- Primary key dapat berupa simpel atau komposit.
- Primary key simpel harus berkorespondensi dengan satu atribut dari entitas.
- **@javax.persistence.Id** berguna untuk menandai suatu atribut sebagai *unique identifier* dan dapat berupa salah satu tipe berikut:
 - *Primitive Java types: byte, int, short, long, char*
 - *Wrapper classes of primitive Java types: Byte, Integer, Short, Long, Character*
 - *Arrays of primitive or wrapper types: int[], Integer[], etc.*
 - *Strings, numbers, and dates: java.lang.String, java.math.BigInteger, java.util.Date, java.sql.Date*

Entitas Buku

Disertai Generated Identifier



```
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    // Constructors, getters, setters
}
```

Primary Key Komposit



- Pada dasarnya sewaktu kita memetakan entitas, sebaiknya menggunakan SINGLE COLUMN sebagai primary key.
- Namun, kadang-kadang ada beberapa kasus di mana kita membutuhkan primary key komposit.
- Misalnya: gabungan kolom id dengan tanggal atau jam, dll.
- Untuk itu, kita harus mendefinisikan suatu CLASS PRIMARY KEY yang merepresentasikan key komposit.
- Ada dua anotasi yang dapat kita gunakan
 - `@EmbeddedId`
 - `@IdClass`.
- Kedua anotasi tsb akan menghasilkan skema database yang sama, tetapi cara melakukan query terhadap entitas nantinya akan sedikit berbeda.

Primary Key Komposit Cara 1: @EmbeddedId



- Embedded Object tidak memiliki identitas/primary key sendiri. Atributnya akan menjadi kolom pada tabel dari entitas lain.
- Contoh: CLASS NewsID adalah embeddable class terdiri dari 2 atribut 'title' dan 'language'. Class ini tidak memiliki anotasi @Id

@Embeddable

```
public class NewsId {  
    private String title;  
    private String language;  
    // Constructors, getters, setters, equals, and hashCode  
}
```

Contoh Entitas News memiliki NewsId



- Entitas News berikut ini meng-embed NewsId sebagai primary key dengan menggunakan anotasi `@EmbeddedId`
- Setiap anotasi `@EmbeddedId` harus mengacu pada embeddable class yang telah ditandai dengan anotasi `Embeddable`

```
@Entity
public class News {
    @EmbeddedId
    private NewsId id;
    private String content;
    // Constructors, getters, setters
}
```

Contoh Kode Java

untuk mencari suatu Entitas melalui Primary Key Komposit



- Untuk mencari suatu Entitas melalui Primary Key Komposit, kita harus meng-instantiate class-nya terlebih dahulu dan mem-passing objek-nya kepada entity manager.
- Contoh:

```
NewsId pk = new NewsId("Richard Wright has died", "EN")  
News news = em.find(News.class, pk);
```

Primary Key Komposit 2: @IdClass



- Cara lain untuk mendeklarasikan primary key komposit adalah dengan menggunakan anotasi `@IdClass`.
- Melalui cara ini, setiap atribut dari class primary key harus dideklarasikan pada class entitas dan diberi anotasi `@Id`.
- Class entitas News harus mendefinisikan primary key menggunakan anotasi `@IdClass` dan menganotasi setiap key dengan `@Id`
- Contoh:

```
@Entity
@IdClass(NewsId.class)
public class News {
    @Id private String title;
    @Id private String language;
    private String content;
    // Constructors, getters, setters, equals, and hashCode
}
```

Tabel yang dihasilkan



- Kedua cara, @EmbeddedId dan @IdClass, akan dipetakan menjadi struktur tabel yang sama.

```
create table NEWS (  
  CONTENT VARCHAR(255),  
  TITLE VARCHAR(255) not null,  
  LANGUAGE VARCHAR(255) not null,  
  primary key (TITLE, LANGUAGE)  
);
```

Pemetaan Kolom: @Basic



- Anotasi **@javax.persistence.Basic** adalah tipe pemetaan yang paling simpel terhadap suatu kolom database.
- Anotasi ini memiliki 2 parameter: optional dan fetch.
 - **Optional** digunakan untuk menyatakan apakah atribut boleh NULL.
 - **Fetch** dibagi lagi menjadi 2 yaitu: **LAZY** dan **EAGER**
 - ✦ Digunakan untuk memberitahu JPA pada saat runtime, apakah data harus diambil belakangan (**LAZYLY**) pada saat aplikasi mengakses property (getter) atau diambil segera (**EAGERLY**) pada saat entitas diload pertama kali.
- Contoh kasus: entitas Track memiliki 'title', 'description', dan file 'wav'
- File 'wav' ini memiliki tipe BLOB yang dapat berukuran sampai beberapa Megabyte.
- Pada saat kita mengakses entitas Track ini, tentu kita tidak ingin segera me-load file WAV karena berukuran besar dan belum tentu dipakai.
- Untuk itu kita gunakan **@Basic(fetch = FetchType.LAZY)** sehingga data akan diload belakangan pada saat kita benar-benar membutuhkannya dengan menggunakan method getter.

The Track Entity with Lazy Loading on the wav Attribute



```
@Entity
public class Track {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String title;
    private Float duration;
    @Basic(fetch = FetchType.LAZY)
    @Lob
    private byte[] wav;
    @Basic(optional = true)
    private String description;
    // Constructors, getters, setters
}
```

- Note that the wav attribute of type byte[] is also annotated with @Lob to store the value as a large object (LOB).

@Column



- Anotasi **@javax.persistence.Column** berguna untuk mendefinisikan property dari suatu kolom.
- Dengan menggunakan anotasi ini, kita dapat menentukan NAMA KOLOM, Ukuran, NULL/NOT NULL, UNIQUE, Updatable/Insertable

Customizing Mapping for the Book Entity



- Contoh: kita mengubah nama kolom 'title' dan 'nbOfPage', panjang dari 'description' serta mengharuskan nilai NOT NULL pada bbrp kolom

```
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(name = "book_title", nullable = false, updatable = false)
    private String title;
    private Float price;
    @Column(length = 2000)
    private String description;
    private String isbn;
    @Column(name = "nb_of_page", nullable = false)
    private Integer nbOfPage;
    private Boolean illustrations;
    // Constructors, getters, setters
}
```

Tabel BOOK



```
create table BOOK (  
  ID BIGINT not null,  
  BOOK_TITLE VARCHAR(255) not null,  
  PRICE DOUBLE(52, 0),  
  DESCRIPTION VARCHAR(2000),  
  ISBN VARCHAR(255),  
  NB_OF_PAGE INTEGER not null,  
  ILLUSTRATIONS SMALLINT,  
  primary key (ID)  
);
```

- The **updatable** and **insertable** settings default to true, which means that any attribute can be inserted or updated in the database. You can set them to false when you want the persistence provider to ensure that it will not insert or update the data to the table in response to changes in the entity.

@Temporal



- Di dalam kode Java, kita dapat menggunakan `java.util.Date` and `java.util.Calendar` untuk menyimpan dalam bentuk tanggal, jam, atau millisecond.
- Untuk menspesifikasikan hal ini di dalam ORM, kita dapat menggunakan anotasi `@javax.persistence.Temporal`
- Anotasi ini memiliki 3 nilai pilihan: `DATE`, `TIME`, atau `TIMESTAMP`.

Contoh

Entitas Customer dengan 2 atribut @Temporal



```
@Entity
public class Customer {
    @Id
    @GeneratedValue
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
    private String phoneNumber;
    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;
    @Temporal(TemporalType.TIMESTAMP)
    private Date creationDate;
    // Constructors, getters, setters
}
```

```
create table CUSTOMER (
    ID BIGINT not null,
    FIRSTNAME VARCHAR(255),
    LASTNAME VARCHAR(255),
    EMAIL VARCHAR(255),
    PHONENUMBER VARCHAR(255),
    DATEOFBIRTH DATE,
    CREATIONDATE TIMESTAMP,
    primary key (ID)
);
```

@Transient



- Pada JPA, segera setelah suatu class dianotasi dengan @Entity, maka semua atribut-nya akan secara otomatis dipetakan ke dalam tabel.
- Jika kita tidan ingin memetakan suatu atribut, kita dapat menggunakan anotasi **@javax.persistence.Transient**
- Contoh: entitas Customer memiliki atribut 'age'
- Karena 'age' dapat dihitung otomatis dari tanggal lahir, maka atribut 'age' tidak perlu dipetakan

The Customer Entity with a Transient 'Age'

```
@Entity
public class Customer {
    @Id
    @GeneratedValue
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
    private String phoneNumber;
    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;
    @Transient
    private Integer age;
    @Temporal(TemporalType.TIMESTAMP)
    private Date creationDate;
    // Constructors, getters, setters
}
```

As a result, the age attribute doesn't need any AGE column to be mapped to.

Access Type



- Selama ini kita melakukan anotasi pada 'atribut' entitas.
- Cara lain adalah dengan memberikan anotasi pada property (getter method). Cara ini sama saja, hanya ada perbedaan pada saat inheritance digunakan.

```
@Entity
public class Customer {
    @Id @GeneratedValue
    private Long id;
    @Column(name = "first_name", nullable = false, length = 50)
    private String firstName;
    @Column(name = "last_name", nullable = false, length = 50)
    private String lastName;
    private String email;
    @Column(name = "phone_number", length = 15)
    private String phoneNumber;
    // Constructors, getters, setters
}
```

Anotasi pada Property (Getter)



```
@Column(name = "first_name", nullable = false, length = 50)  
public String getFirstName() {  
    return firstName;  
}
```

```
@Column(name = "last_name", nullable = false, length = 50)  
public String getLastName() {  
    return lastName;  
}
```

```
@Column(name = "phone_number", length = 15)  
public String getPhoneNumber() {  
    return phoneNumber;  
}
```

Collection of Basic Types



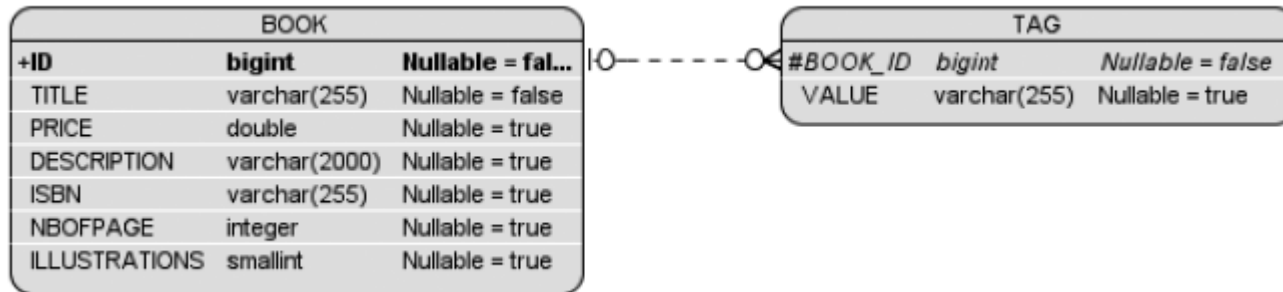
- Sejak JPA 2.0, kita dapat melakukan anotasi terhadap tipe data collection dengan menggunakan **@ElementCollection** dan **@CollectionTable**
 - @ElementCollection digunakan untuk memberitahu bahwa atribut bertipe `java.util.Collection` berisikan tipe data yg lain.
 - @CollectionTable memungkinkan kita untuk meng-customise detail dari tabel collection.

Contoh Entitas Book Memiliki atribut 'tag'

```
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    @ElementCollection(fetch = FetchType.LAZY)
    @CollectionTable(name = "Tag")
    @Column(name = "Value")
    private ArrayList<String> tags;
    // Constructors, getters, setters
}
```

- Tag merupakan collection berisi String

Tabel yang dihasilkan



- Note: apabila **@CollectionTable** tidak digunakan, maka nama tabel default-nya adalah “BOOK_TAG”

Map of Basic Types



- Pada dasarnya, pemetaan tipe data 'Map' dapat dilakukan sama halnya seperti pemetaan 'Collection'
- Pada Map kita perlu menentukan kolom **Key** dan **Value**
- Contoh kasus: pada album CD, sebuah track dapat memiliki POSISI (key) dan TITLE (value)

Contoh entitas CD disertai tipe data Map



```
@Entity
public class CD {
    @Id
    @GeneratedValue
    private Long id;
    private String title;
    private Float price;
    private String description;
    @Lob
    private byte[] cover;
    @ElementCollection
    @CollectionTable(name="track")
    @MapKeyColumn (name = "position")
    @Column(name = "title")
    private Map<Integer, String> tracks;
    // Constructors, getters, setters
}
```

Tabel yang dihasilkan



CD		
+ID	bigint	Nullable = false
TITLE	varchar(255)	Nullable = true
PRICE	double	Nullable = true
DESCRIPTION	varchar(255)	Nullable = true
COVER	blob(64000)	Nullable = true



TRACK		
#CD_ID	bigint	Nullable = false
POSITION	integer	Nullable = true
TITLE	varchar(255)	Nullable = true

Next Meeting



- Mapping menggunakan XML
- Lebih lanjut dengan Embeddable
- Relationship Mapping
- Inheritance Mapping