BINAR

Spring Data JPA (Part 1)

Gold - Chapter 4 - Topic 6

BINAR

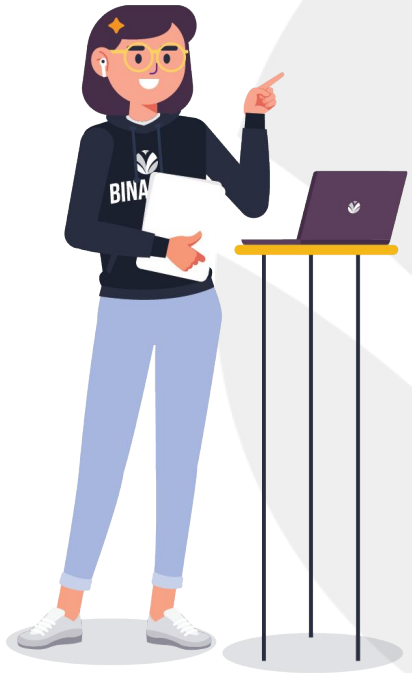
Selamat datang di Chapter 4 Topic 6 online course Back End Java dari Binar Academy!

Topik baru, hari baru! □

Udah masuk di topik menjelang akhir, nih. Setelah kita belajar tentang konsep dasar dari NoSQL, lanjut ke sini kita belajar tentang **Spring Data JPA**.

Mulai dari konsep Data JPA & JDBC, hubungan aplikasi ke database dan cara menghubungkannya, sampai konsep dari serializable pokoknya semuanya, deh~

Yuk, langsung aja kita kepoin~



Dari sesi ini, kita bakal bahas hal-hal berikut:

- Konsep Spring Data JPA & JDBC
- Mengenal framework lainnya
- Hubungan aplikasi ke database
- Cara menghubungkan one to one, one to many, many to one, many to many
- Konfigurasi Spring Data JPA
- Cara melakukan Primary Key dan Compound Key
- Konsep Serializable

Tarik napasss, hembuskan~

Materi pertama yang bakal kita pelajari, yaitu JPA atau Java Persistence API.

Kira-kira materinya kayak gimana, ya? Langsung aja kita cari tahu, yuk!

JPA sebagai aplikasi penengah~

Sebelumnya, kita udah belajar tentang bahasa pemrograman Java yang merupakan bahasa pemrograman berorientasi object dan juga database yang berupa relational database

Tapi kamu tahu nggak, sih? Supaya Java dan relational database bisa berkomunikasi, **perlu ada penengah** buat menerjemahkannya.

Penengah ini adalah **JPA yaitu Java Persistence API**, yaitu aplikasi penengah buat melakukan standarisasi komunikasi antara Java dengan relational database.

Kenalan lebih jauh sama JPA, yuk!

JPA pertama kali diperkenalkan pada tahun 2006.

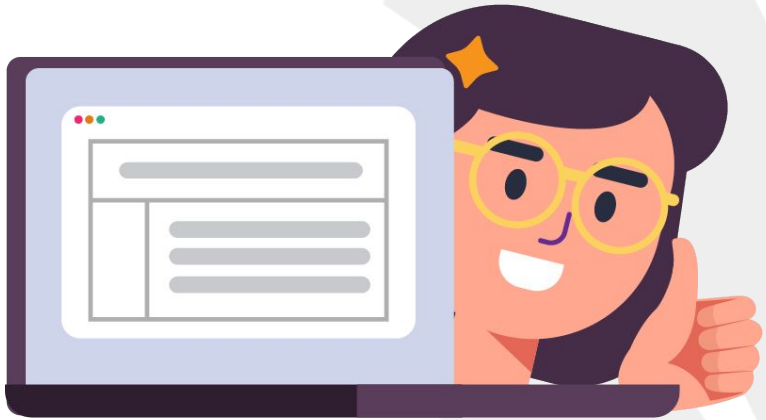
JPA ternyata bagian dari JEE (Java Enterprise Edition) yang merupakan perluasan dari Java SE (Standard Edition). Tujuannya sih supaya **Java bisa dipakai lebih aplikatif**, kayak menghubungkan ke database dan bikin suatu web.

Oh iya, JPA merupakan salah satu implementasi dari sebuah ORM, lho~

“Terus manfaat dari JPA itu apa, sih?”

Berikut adalah manfaat dari JPA:

- Bisa memetakan table yang ada di database pada class yang ada di program java.
- Semua table di database bakal direpresentasikan sebagai suatu object di Java. Class dengan annotation @Entity merupakan representasi dari sebuah table. Sedangkan field dari class tersebut akan menjadi kolom pada table.



- Kolom dari suatu table bakal direpresentasikan sama field di object.
- Relasi dari table bisa didefinisikan berkat JPA.
- Proses pendekatan buat menghubungkan Object dengan relational database ini disebut dengan object Relational mapping.

- Buat meng-cover kebutuhan dasar aplikasi kita buat JPA, maka kita bakal menggunakan Spring Data JPA aja, ya.
- Kalau buat **menambahkan Spring Data JPA, kita bisa menambahkan dependency berikut di pom.xml** nya.

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>       </dependency>
```

Kalau JPA merupakan kebutuhan tingkat tinggi buat Java.

Kalau JDBC merupakan kebutuhan dasarnya nih, sob.

“Emangnya harus pakai JDBC, ya?”

Iya, dong.

Pas pakai JPA, kita bisa menghubungkan Java sebagai bahasa pemrograman berorientasi object, tapi ternyata kita masih butuh bantuan nih untuk meng-koneksikan Java dengan database.

Selain itu, karena Program Java harus mengenal RDBMS yang digunakan, maka Program Java juga perlu melakukan authentication supaya dapetin permission buat akses database tersebut.

JDBC yang berperan buat kebutuhan dasar Java~

Jadi, sebelum kebutuhan tingkat tinggi seperti JPA dipenuhi, ternyata ada kebutuhan yang lebih mendasar supaya keduanya bisa berkomunikasi nih, yaitu JDBC (Java Database Connectivity).

“Terus cara menghubungkannya, gimana?”

Kalau meng-install JDBC, kita bisa menghubungkan berbagai jenis database, kayak SQL server, PostgreSQL, OracleSQL, MySQL.

Terus, kalau mau menambahkan database tersebut, kita perlu suatu driver JDBC.

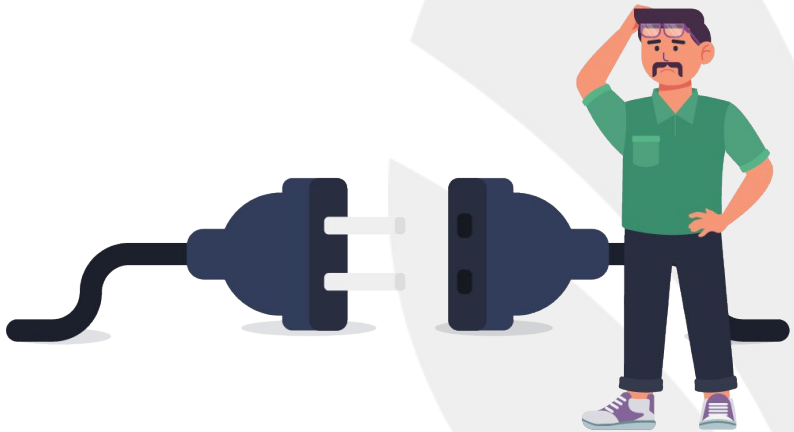
Oh, iya. Kita bisa menambahkan dependency di samping pada pom.xml project, gengs.

```
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
```

Karena pakai PostgreSQL, berarti kita pakainya driver PostgreSQL, ya.

Sebenarnya, Spring Data udah meng-cover JDBC, tapi kita harus tetap memasukkan driver dari database untuk menentukan RDBMS apa yang nantinya bakal dihubungkan.

Biar nggak salah sambung, gitu~



Pada topic sebelumnya, kita udah belajar tentang beberapa Framework, kan?

Buat menambah pengetahuan kamu, selanjutnya kita bakal cari tahu Framework lainnya yang masih berhubungan sama Java.

Apa aja yaaa?

“Emangnya apa aja sih framework-nya?”

Kayak yang udah dibahas, kita pakai Spring Data JPA untuk menghubungkan program Java dengan database.

Walaupun begitu, ternyata ada framework lain yang bisa kita pakai, yaitu Hibernate dan MyBatis. Framework tersebut cukup popular digunakan oleh banyak perusahaan.

Hibernate

myBatis

Kamu udah tahu belum kalau ternyata Framework Hibernate itu ada di Spring Data?

Iya, bener.

Jadi, sebenarnya beberapa implementasi dari Hibernate juga ada di Spring Data JPA, tapiii Framework Hibernate sendiri merupakan framework yang independent.

Ternyata bukan cuma kamu sama dia aja yang punya hubungan spesial. Tapi aplikasi sama database juga sama!

Biar nggak penasaran, selanjutnya kita bakal bahas **hubungan aplikasi ke database.**

Berikut cara menghubungkan program Java yang berorientasi object dengan relational database~

Misalnya gini, kita punya table yang bernama user yang punya kolom id, nama dan email.

Kemudian di program Java yang dibuat, kita bikin class kayak gambar di samping.

```java
@Entity // This tells Hibernate to make a table out of this class
@Getter
@Setter
public class User {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Integer id;

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

  private String email;
}
```

Pada contoh tersebut, ada table dan kolom-kolom yang nama pada database-nya nggak didefinisikan.

Yang didefinisikan cuma di kolom nama karena ada spring.jpa.hibernate.naming-strategy.

```
@Entity // This tells Hibernate to make a table out of this class
@Getter
@Setter
public class User {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Integer id;

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

  private String email;
}
```

Tapi, kalau kita pakai value pada:

1. spring.jpa.hibernate.ddl-auto create,
2. create-drop.
3. update,

Maka bakal diambil dari nama class atau field di entity. Jadi buat nama table bakal mengikuti nama class secara otomatis, yaitu user. Tetapi pada field name bakal jadi kolom nama.

```
@Entity // This tells Hibernate to make a table out of this class
@Getter
@Setter
public class User {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Integer id;

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

  private String email;
}
```

Selain itu penggunaan annotation @Id merupakan penanda sebuah field primary key.

Sedangkan pada @GeneratedValue, artinya nilai dari kolom tersebut bakal dibikin secara otomatis tanpa harus mengatur pada saat melakukan persisting.

```
@Entity // This tells Hibernate to make a table out of this class
@Getter
@Setter
public class User {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Integer id;

  @Column(name = “nama”)
  private String name;

  private String email;
}
```

Sipp deh! Sekarang kita bakal bahas tentang One to One, One to Many, Many to One, Many to Many.

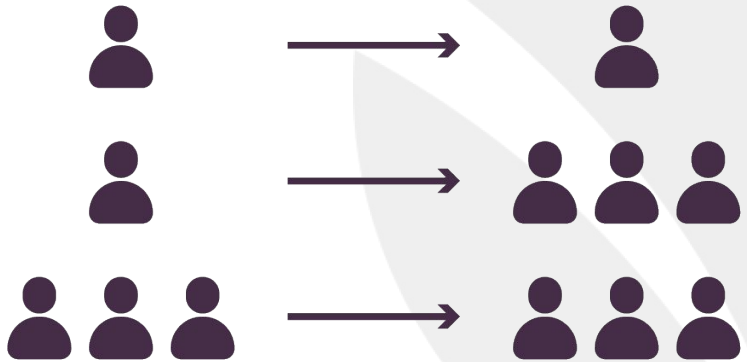
Kalau dari istilahnya harusnya udah kena spoilers, bahwa keempatnya itu bakal berhubungan sama dua sisi.

Dari sisi satu ke sisi lainnya.

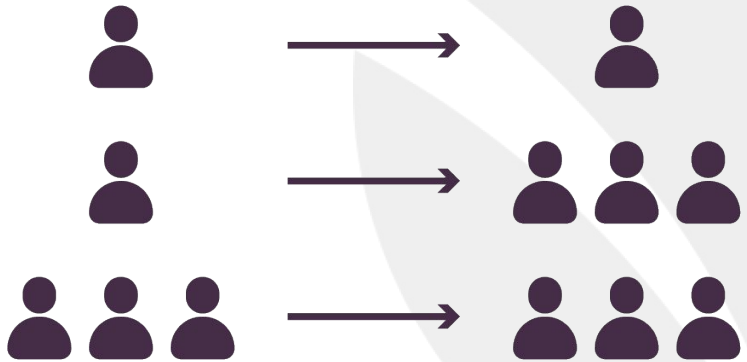
“Emangnya hubungan one to one, one to many, many to one, dan many to many itu apa?”

Kita bahas satu-satu, ya!

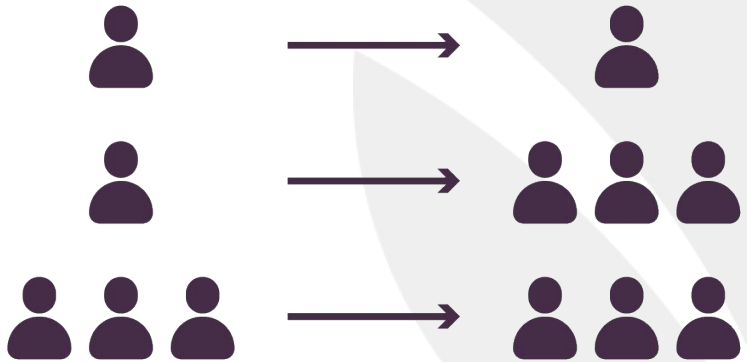
- **One to One**, punya relasi satu data di row yang berkaitan dan juga memiliki satu row di table yang punya relasi.



- **One to many dan many to one**, salah satu table punya satu row di table yang punya relasi dan dari table ini nantinya punya row yang banyak (foreign key tetap satu tetapi punya banyak row).



- **Many to many**, di kedua table yang punya relasi memungkinkan punya lebih dari satu data yang berkaitan.



Berikut adalah contoh penggunaan relasinya~

- **Annotation @OneToOne**

  Contoh penggunaannya bisa kamu cek secara detail di sini, ya.

- **Annotation @OneToMany dan @ManyToOne**

  Buat penjelasan lengkap tentang contoh penggunaannya, kamu juga bisa cek di sini ya, gengs.

- **Annotation @ManyToMany**

  Contoh penggunaannya bisa kamu pelajari di sini ya, bestie.

Dari pembahasan referensi tersebut kita jadi tahu beberapa terminologi baru lho, gengs~

Berikut adalah terminologinya:

- **Annotation @JoinColumn** bakal menandakan sebuah field jadi foreign key bagi yang lain.
- **Unidirectional** adalah kondisi kepemilikan data untuk mengakses dari satu sisi tanpa mengubah data yang punya relasi.
- **Bidirectional** adalah kondisi kepemilikan data yang kedua sisi table-nya, bisa mempengaruhi satu dan lainnya.

Lanjutt~

Karena udah ada banyak gambaran tentang Spring Data JPA, berarti sekarang waktunya kita cari tahu Common Spring Data JPA config.

Yuk yukkk~

Ternyata ada banyak cara untuk melakukan konfigurasi database pada program Java, lho~

Pas udah menambahkan driver dan spring data JPA di project Java, ada beberapa konfigurasi yang harus dilakukan.

Kita bisa bikin Bean untuk melakukan setup Database yang mengembalikan Object DataSource.

Di dalamnya, kita melakukan setup url, user dan password dari database yang dituju. Selain itu juga masih banyak properties yang lainnya.

Eh walaupun begitu, kita bakal menggunakan konfigurasi yang lebih gampang karena value bakal di store pada **application.properties.**

Jangan panik dulu, gengs~

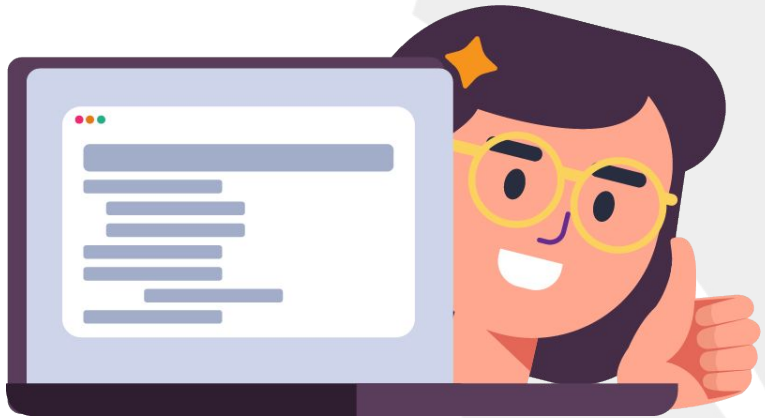
BINAR

Berikut beberapa konfigurasi yang perlu ditambahkan pada application.properties:

```
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=none
spring.jpa.hibernate.show-sql=true
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=admin
```

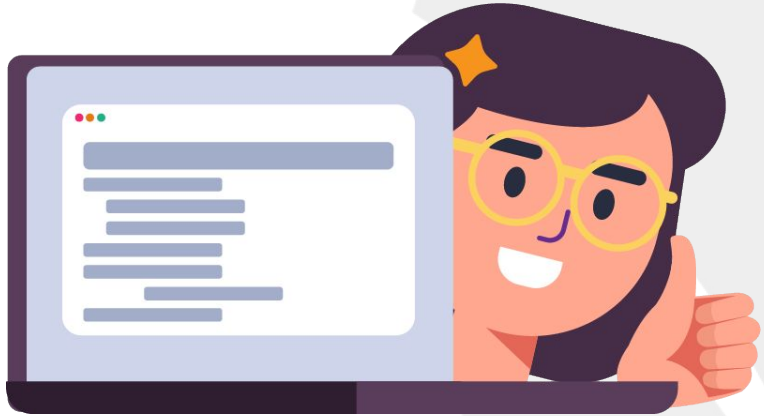
Pada url, username dan password, valuenya bakal disimpan pakai placeholder variable yang biasanya ada di dalam suatu secret file.

Yaitu tempat dimana environment tertentu disimpan oleh tim Dev Ops atau IT DBA.



Tapi, nggak masalah kok kalau kamu ingin menuliskan default value untuk pengetesan di environment developer team.

Oh iya, pada url yang disediakan, udah termasuk ke database scheme. Jadi, kalau kita mengkoneksikan aplikasi kita ke database, maka database scheme juga harus disertakan di url-nya



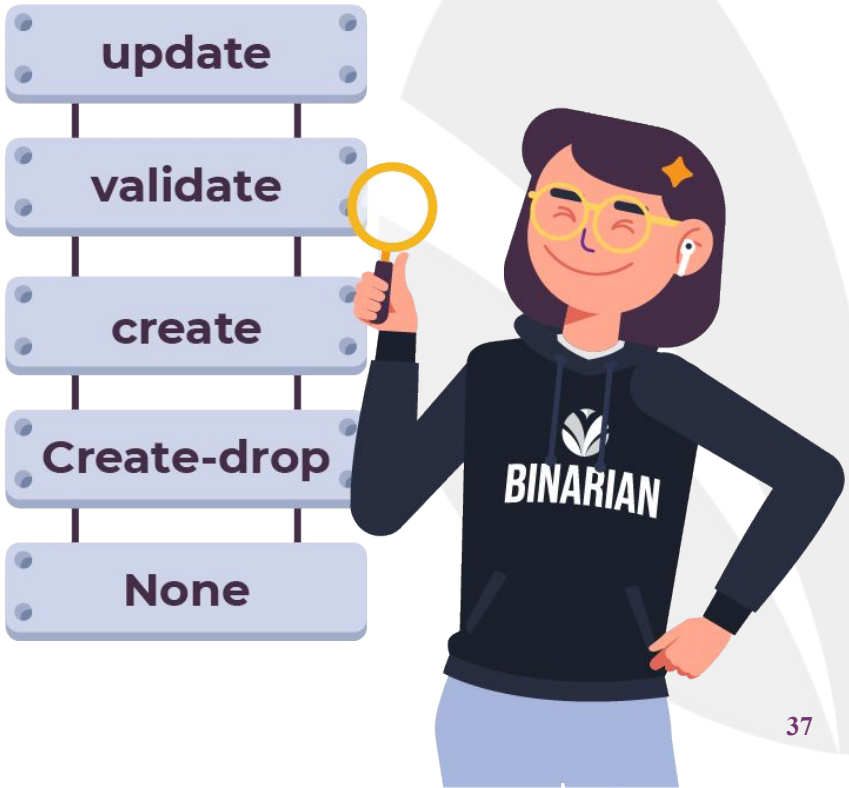
Selain itu ada properties yang cukup penting lagi lho, sob!

Properties ini adalah **spring.jpa.hibernate.ddl-auto**, yang punya beberapa value sebagai berikut:

1. **update**, yaitu mengupdate kolom yang ada di database apabila ada penambahan field pada entity.

   Sebagai catatan, pas menghubungkan program dan database pertama kali, table-table bakal langsung dibikin sesuai sama entity pada program yang dibikin.

2. **validate**, yaitu mem-validasi kondisi antara entity harus sesuai sama table yang ada di database.



3. **create,**

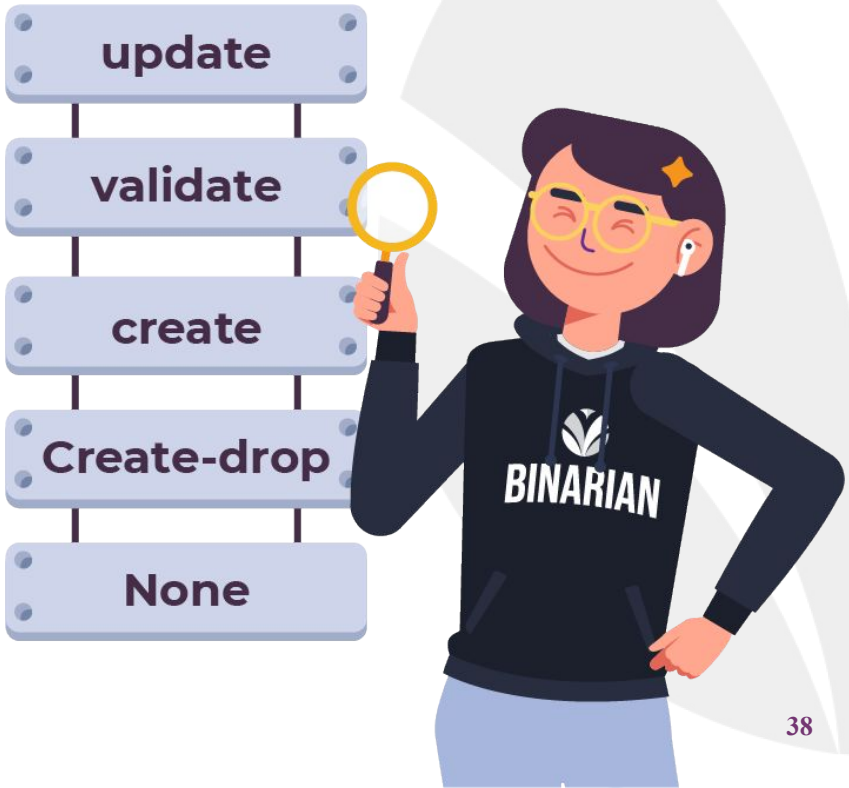
   yaitu ketika menghubungkan program dan database pertama kali, table akan langsung disesuaikan oleh entity pada program yang dibuat. Tapi, nggak bakal ada update kolom.

4. **Create-drop**,

   yaitu melakukan create table baru ketika dihubungkan lalu akan melakukan drop ketika aplikasi dimatikan.

5. **None**

   yaitu menghubungkan tanpa perlu adanya validasi.



Property spring.jpa.hibernate.naming-strategy ternyata bermanfaat banget, lho~

Properties:

spring.jpa.hibernate.naming-strategy

membantu penamaan table supaya sesuai dengan nama yang ada di entity. Terutama kalau nama field nggak didefinisikan.

Masih banyak properties lainnya untuk melakukan konfigurasi Spring Data JPA nih, gengs~

Berikut Referensi yang lebih lengkap:

- Configuration Spring Data JPA with Spring Boot
- Configuration JDBC
- Spring Data JPA - Reference Documentation

Masih berkaitan sama Java dan Spring Data, berikutnya kita bakal bahas **Primary Key dan Compound Key.**

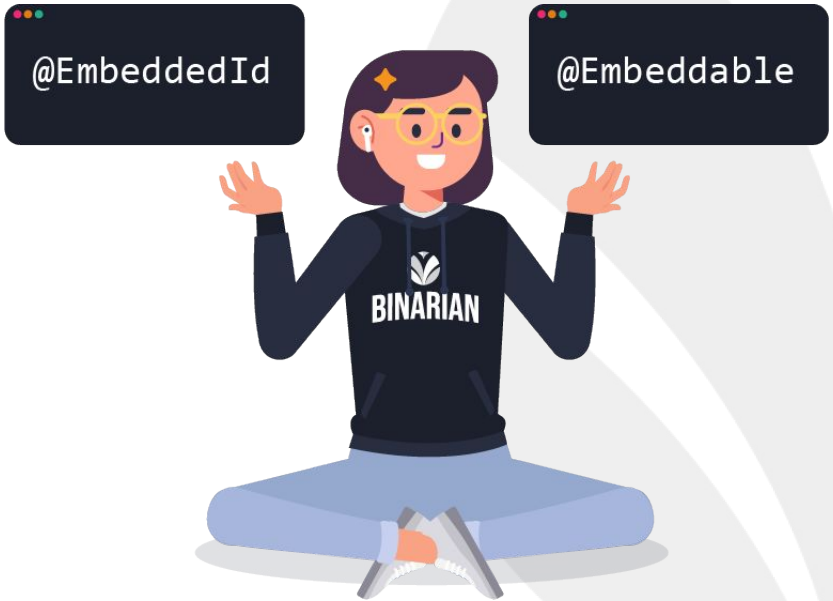
Eits, ini bukan key yang artinya kunci buat rumah atau kendaraan, ya~

Kita udah tahu cara memetakan sebuah table pakai satu primary key kan?

Caranya yaitu pakai annotation @Id pada field yang mau jadi primary key.

Tapi, gimana kalau primary key merupakan compound key, yaitu primary key disusun dari beberapa kolom?

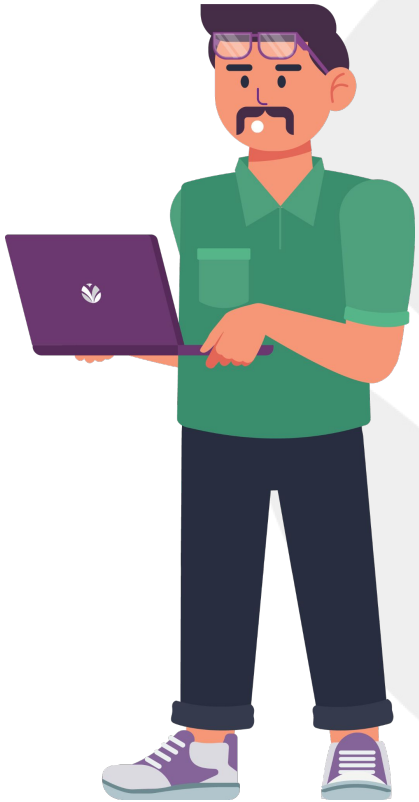
Caranya adalah pakai **@EmbeddedId** dan **@Embeddable**.



Seluruh kolom compound key dijadikan satu dan dibikin sebuah class pakai annotation @Embeddable.

Sedangkan pada table yang entity-nya nggak lagi pakai annotation @Id, akhirnya pakai @EmbeddedId dengan field dari class compound key.

Buat penjelasan codenya, bisa kamu cek di link ini, ya!



Sampai di materi terakhir nih, gengs. Pasti kamu udah penasaran banget, kan?

Buat menutup topic 5, mari kita sambut pembahasan tentang Serializable.

“Emang materi Serialization itu tentang apa, sih?”

Yepp, **Serialization merupakan pengubahan object jadi suatu byte supaya bukan cuma JVM aja yang mengenalnya**.

Di Java, untuk melakukan serialization, kita cuma perlu meng-implement Serializable pada class yang kita tuju.

Sederhana banget kan, sob?

“Kasih contoh penggunaan Serializable, dong!”

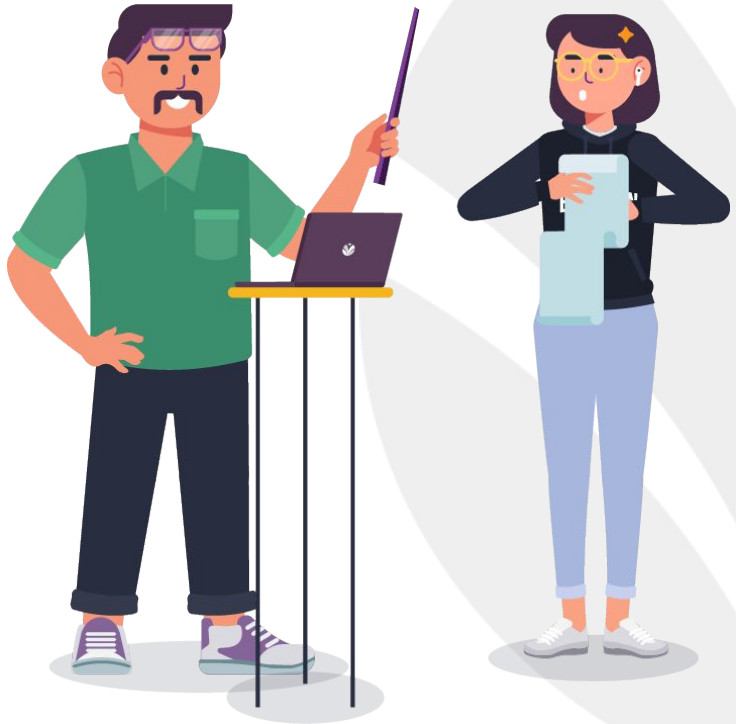
Contoh case penggunaan serializable adalah ketika kita memakai compound key.

Class yang merupakan compound key nggak bakal dideteksi karena yang terdeteksi cuma class entity.

Sehingga untuk mengirimkan informasi tersebut, class dari compound key ini harus meng-implement Serializable.

Berikut dokumentasi mengenai Serialization:

[Serializing Java Objects]



Mari kita recall sedikit mengenai si Spring JPA ini

JPA alias Java Persistence API adalah aplikasi penengah buat melakukan standarisasi komunikasi antara Java dengan relational database.

Kalo dipikir-pikir, apa yang terjadi ya kalo JPA ini nggak ada dan apa imbasnya?

Nah, selesai sudah pembahasan kita di Chapter 4 Topic 6 ini.

Selanjutnya, kita bakal bahas tentang Spring Data JPA part 2.

Penasaran kayak gimana? Cus langsung ke topik selanjutnya~