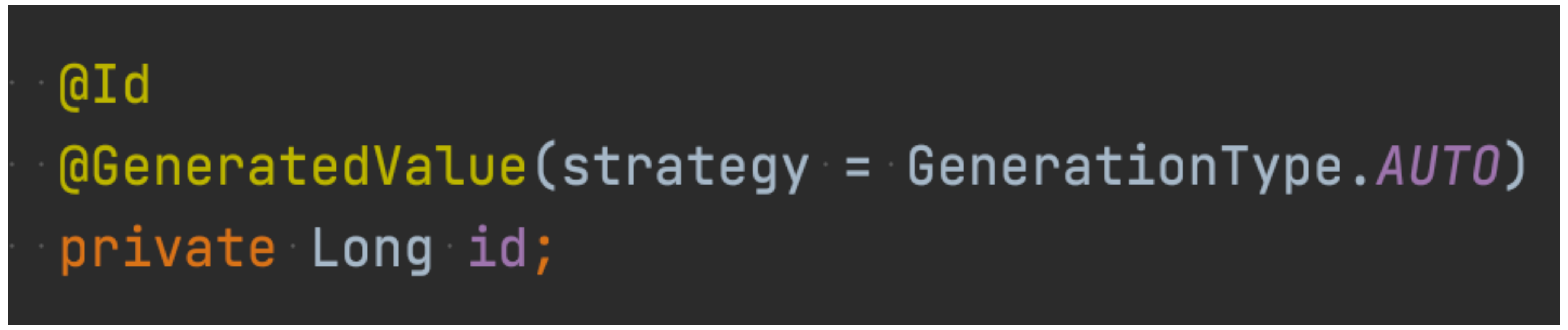
Hibernate & Spring Data JPA

Beginner to Guru

Hibernate Primary Keys

Numeric Primary Keys



```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
```

- GenerationType.AUTO - Let Hibernate Pick - best practice is to specify
- GenerationType.SEQUENCE - Use database sequence (Not a feature of MySQL)
- GenerationType.IDENTITY - Use auto-incremented database columns
- GenerationType.TABLE - Use database table to simulate sequence
    - This is what we have been using, least scalable of the options

UUID Primary Keys

- UUID (you-id) - Universally Unique Identifier, a unique 128 bit value
- Common to use as primary key, can help index performance
  - Downside is it uses more disk space
- IETF RFC 4122 - an international standard for UUID generation
  - Hibernate by default implements a custom generator
  - Hibernate can be configured to generate a IETF RFC 4122 compliant UUID

Natural Primary Keys

- Natural Primary Keys - A unique value with business meaning outside of the database
- A UPC or ISBN could be considered a natural key, since both are expected to be unique
- Common in old legacy databases
- NOT considered best practice

Composite Primary Keys

- Composite Primary Keys - Two values with business meaning combined to make a unique value
- Not considered best practice
- Also common in legacy databases

Which to Use???

- Small Table - ie, few million rows - favor number (Integer or Long)
- Large Table - ie, 10’s of millions or billions - favor UUID (if disk space allows)
- Generally avoid using natural or composite keys
  - Fine in edge cases, like a small code lookup table

Coming Up in the Course

- Hibernate Primary Key Examples
  - Auto Increment - Already showing table, sequence not supported by MySQL
  - UUID
  - UIID RFC 4122
  - Natural Key
  - Composite
- Goal of section is to provide you a cookbook for future reference

SPRING FRAMEWORK

GURU