Relationships and Inheritance

Antonio Goncalves
JAVA CHAMPION

@agoncal   www.antoniogoncalves.org

Previous Module

Main concepts and APIs

CRUD operations

Default mapping

Customize the mapping

Overview

Relationships

Inheritance

OOP

RDBMS

Mapping with JPA

Relationships in OOP

Associations between classes

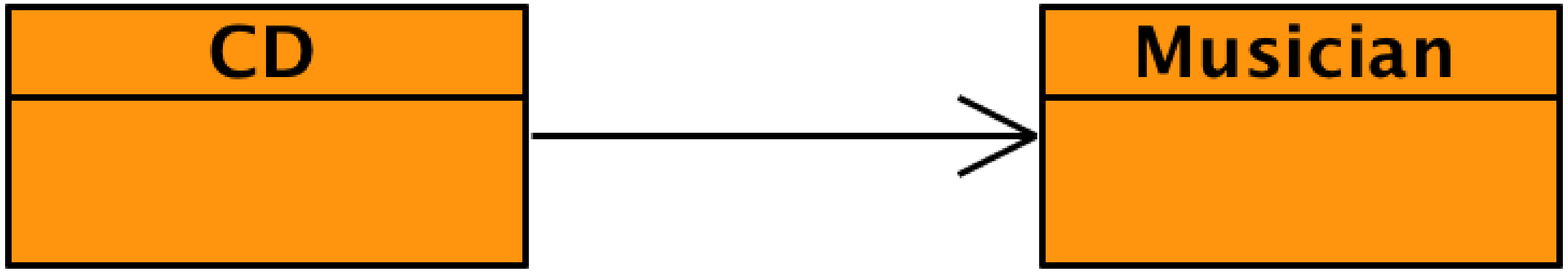
Link objects of one kind to objects of another

Perform an action on its behalf

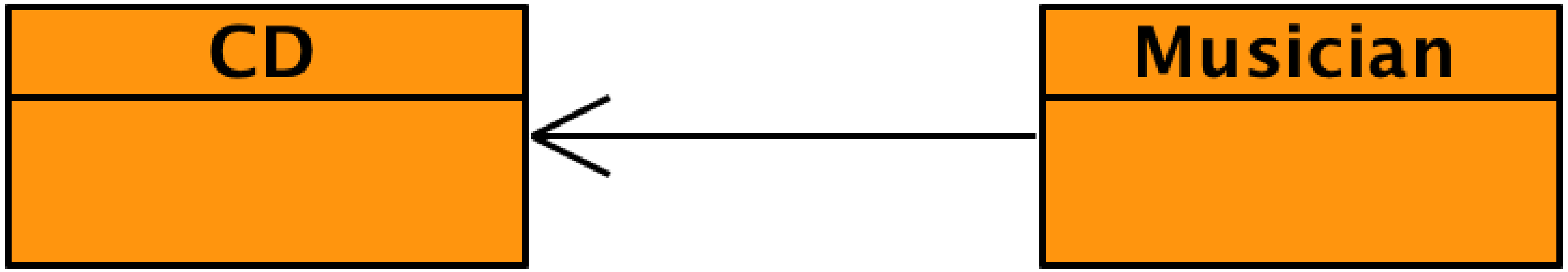
Direction

Cardinality

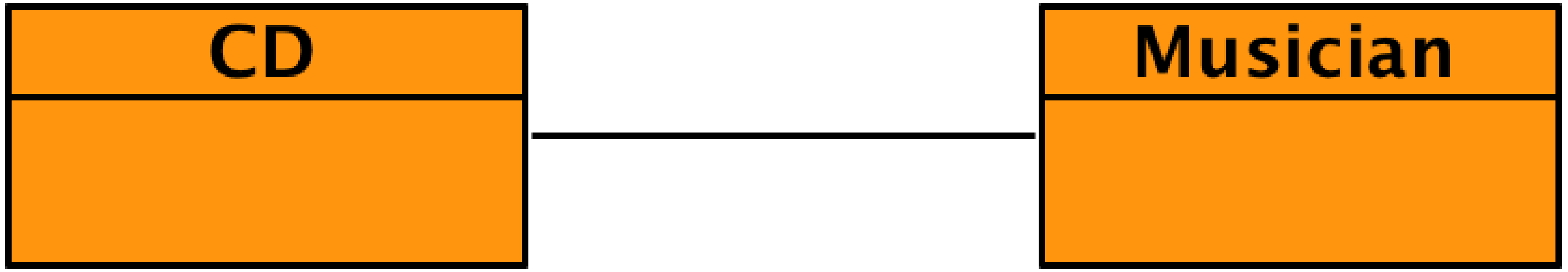
Direction



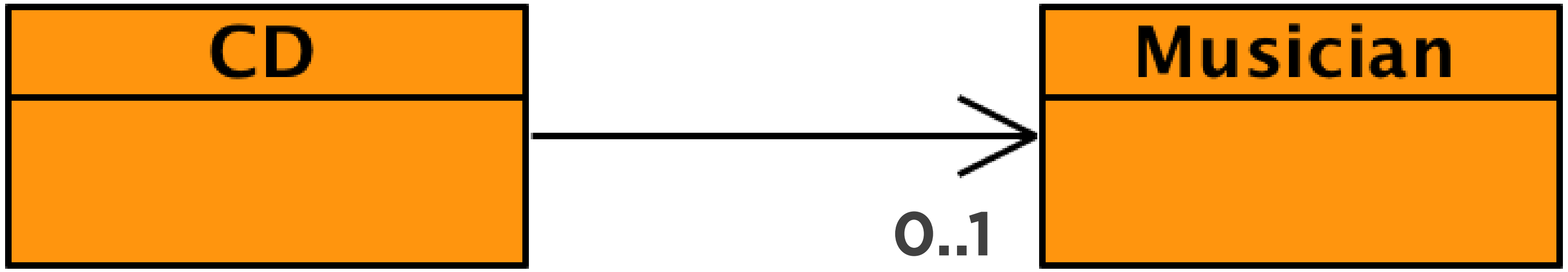
Direction



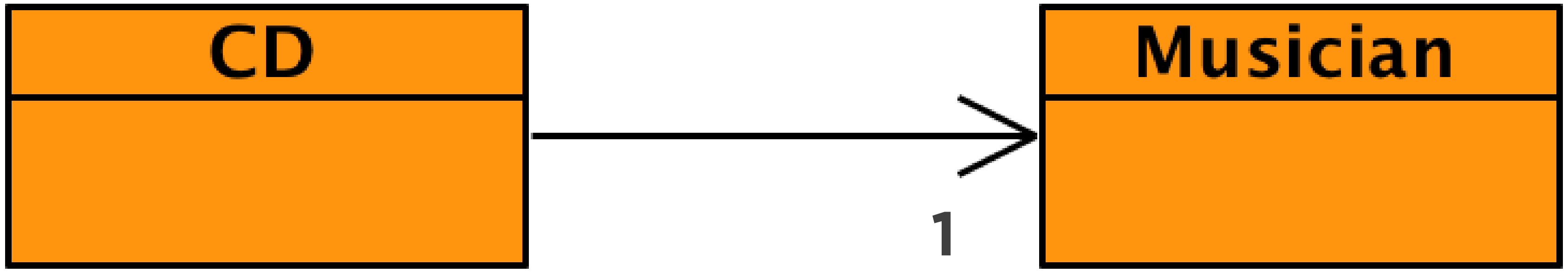
Direction



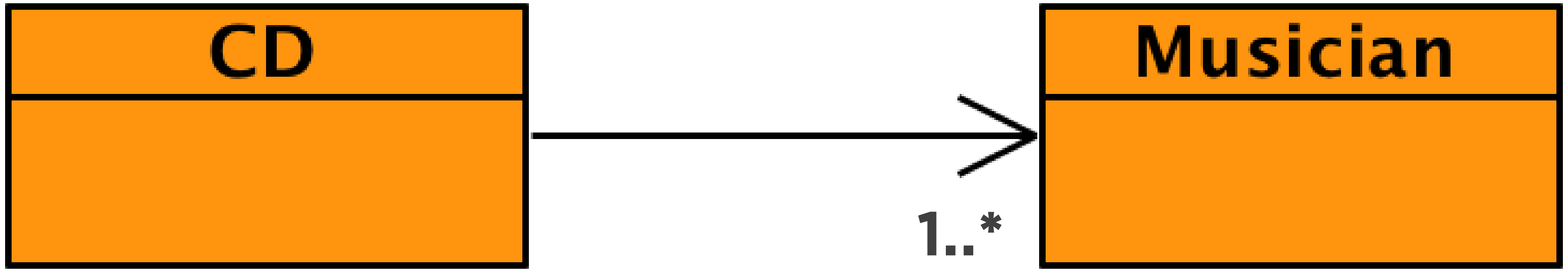
Cardinality



Cardinality



Cardinality



Cardinality in Java

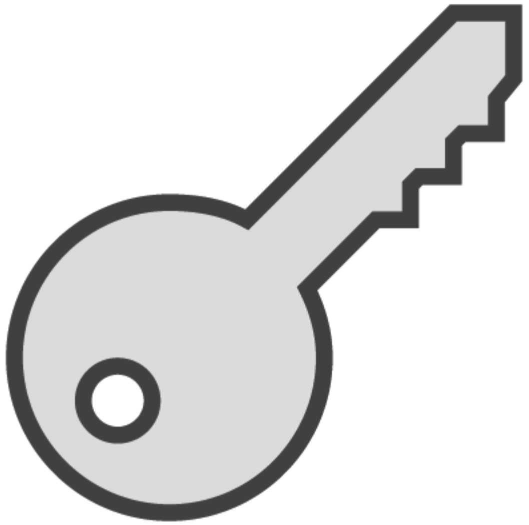
`java.util.Collection`

`java.util.Set`

`java.util.List`

`java.util.Map`

Relationships in RDBMS



Collection of relations

Table

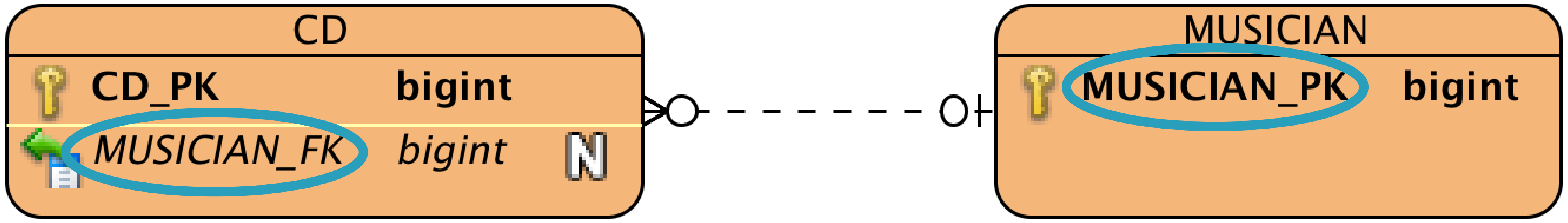
Primary key

Join column

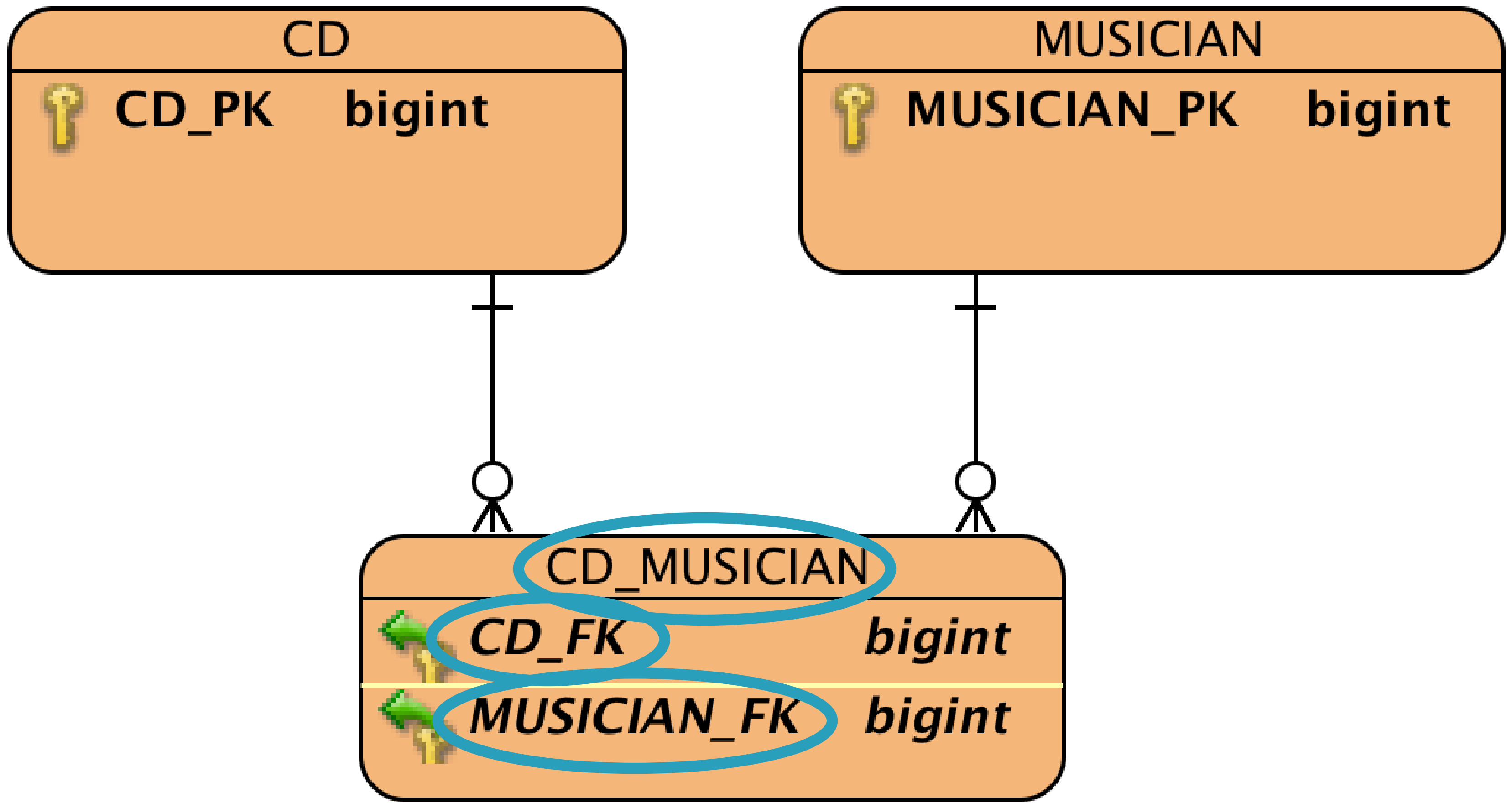
Join table

Foreign key

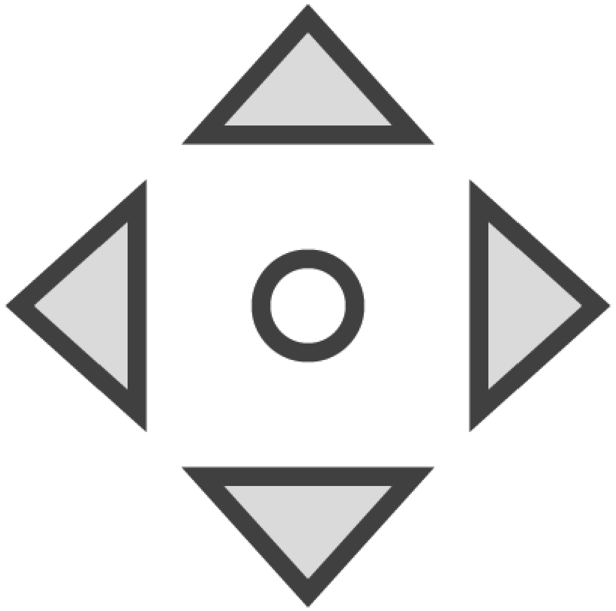
Join Column


CD
CD_PK bigint
MUSICIAN_FK bigint N
MUSICIAN
MUSICIAN_PK bigint

Join Table


CD
CD_PK bigint
MUSICIAN
MUSICIAN_PK bigint
CD_MUSICIAN
CD_FK bigint
MUSICIAN_FK bigint

Relationships in JPA



Map any relationship

Direction

Cardinality

Configuration by exception

Annotations to customize the mapping

- `@OneToOne`
- `@OneToMany`
- `@ManyToOne`
- `@ManyToMany`

Default Database Mapping

JPA Annotation	RDBMS
@OneToOne	Join Column
@ManyToOne	Join Column
@OneToMany	Join Table
@ManyToMany	Join Table

One-to-One Unidirectional

```java
@Entity
public class CD {
  // Attributes and Constructors

  @OneToOne(fetch = FetchType.LAZY)
  @JoinColumn(name = "musician_fk")
  private Musician musician;
}
```

```java
@Entity
public class Musician {
  // Attributes and Constructors
}
```

One-to-One Unidirectional with Join Table

```java
@Entity
public class CD {
  // Attributes and Constructors

  @OneToOne(fetch = FetchType.LAZY)
  @JoinTable
  private Musician musician;
}
```

```java
@Entity
public class Musician {
  // Attributes and Constructors
}
```

One-to-One Unidirectional with Join Table

```
@Entity
public class CD {
  // Attributes and Constructors

  @OneToOne(fetch = FetchType.LAZY)
  @JoinTable(name = "cd_musician",
     joinColumns = @JoinColumn(name = "cd_fk"),
     inverseJoinColumns = @JoinColumn(name = "musician_fk")
  )
  private Musician musician;
}


@Entity
public class Musician {
  // Attributes and Constructors
}
```

One-to-One Unidirectional

```
@Entity
public class CD {
  // Attributes and Constructors


  private Musician musician;
}



@Entity
public class Musician {
  // Attributes and Constructors



}
```

One-to-One Bidirectional

```
@Entity
public class CD {
  // Attributes and Constructors

  @OneToOne
  private Musician musician;
}
```

```
@Entity
public class Musician {
  // Attributes and Constructors

  @OneToOne
  private CD cd;
}
```

One-to-Many Unidirectional

```
@Entity
public class CD {
  // Attributes and Constructors

  @OneToMany
  private Set<Musician> musicians = new HashSet<>();
}



@Entity
public class Musician {
  // Attributes and Constructors



}
```

One-to-Many Bidirectional

```java
@Entity
public class CD {
  // Attributes and Constructors

  @OneToMany
  private Set<Musician> musicians = new HashSet<>();
}



@Entity
public class Musician {
  // Attributes and Constructors

  @ManyToOne
  private CD cd;
}
```

Many-to-Many Bidirectional

```
@Entity
public class CD {
  // Attributes and Constructors

  @ManyToMany
  private Set<Musician> musicians = new HashSet<>();
}



@Entity
public class Musician {
  // Attributes and Constructors

  @ManyToMany
  private Set<CD> cds = new HashSet<>();
}
```

Demo

One to many unidirectional relationship

CD and Musician

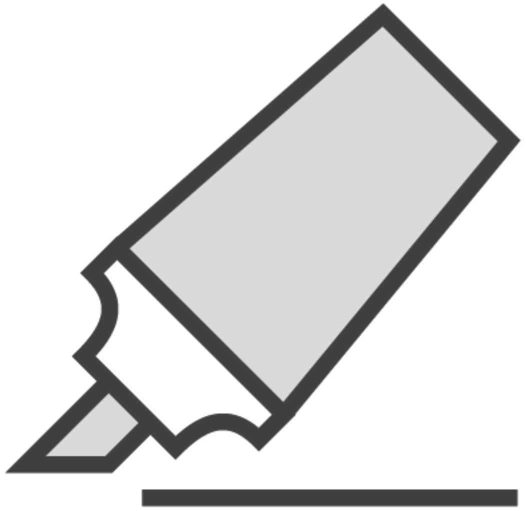
Default mapping

Join table

Customize the mapping

Join column

CRUD Operations on Relationships



Entity passed as argument

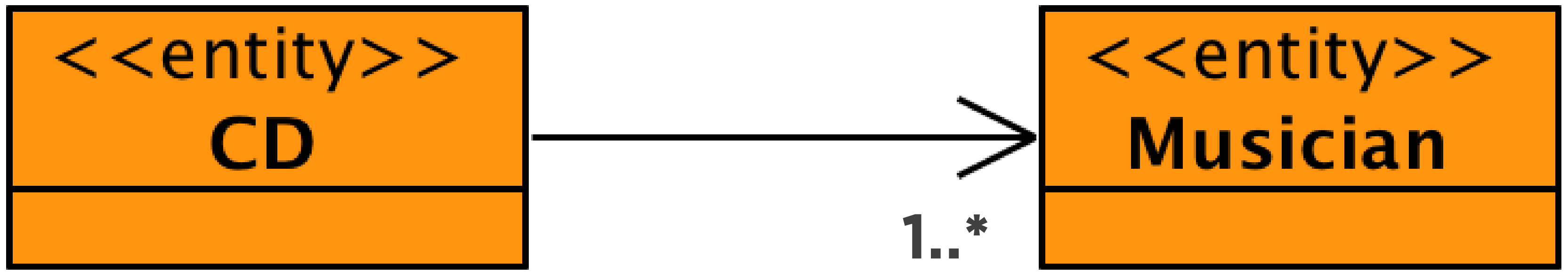
- `em.persist(cd)`
- `em.remove(cd)`
- `em.find(CD.class, id)`

Operations are not cascaded

Fetching

- Eager
- Lazy

One to Many Relationship



Persisting a CD with Musicians

```
public class Main {
  public static void main(String[] args) {

    Set<Musician> beatles = new HashSet<>();
    beatles.add(new Musician("John", "Lennon"));
    beatles.add(new Musician("Paul", "McCartney"));
    beatles.add(new Musician("Ringo", "Starr"));
    beatles.add(new Musician("Georges", "Harrison"));

    CD sergentPepper = new CD("Sergent Pepper");
    sergentPepper.setMusicians(beatles);

    em.persist(cd);
    for (Musician musician : cd.getMusicians()) {
      em.persist(musician);
    }
  }
}
```

Cascading the Persist Event

```java
@Entity
public class CD {
  // Attributes and Constructors

  @OneToMany(cascade = PERSIST)
  private Set<Musician> musicians = new HashSet<>();
}
```

```java
@Entity
public class Musician {
  // Attributes and Constructors
}
```

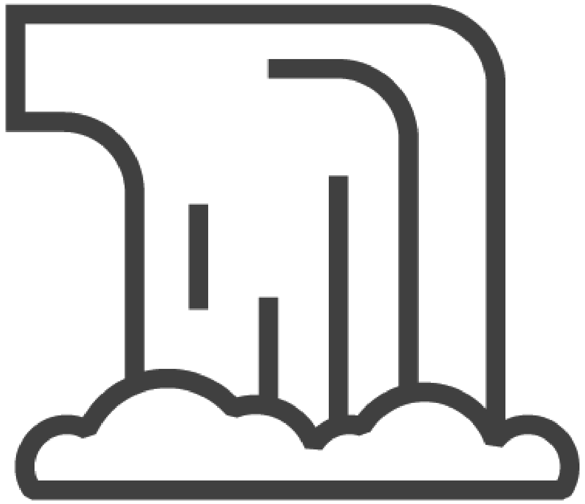
Cascading Several Events

```java
@Entity
public class CD {
  // Attributes and Constructors

  @OneToMany(cascade = {PERSIST, REMOVE, MERGE})
  private Set<Musician> musicians = new HashSet<>();
}



@Entity
public class Musician {
  // Attributes and Constructors
}
```

Cascading All Events

```java
@Entity
public class CD {
  // Attributes and Constructors

  @OneToMany(cascade = ALL)
  private Set<Musician> musicians = new HashSet<>();
}



@Entity
public class Musician {
  // Attributes and Constructors
}
```

Cascading Events



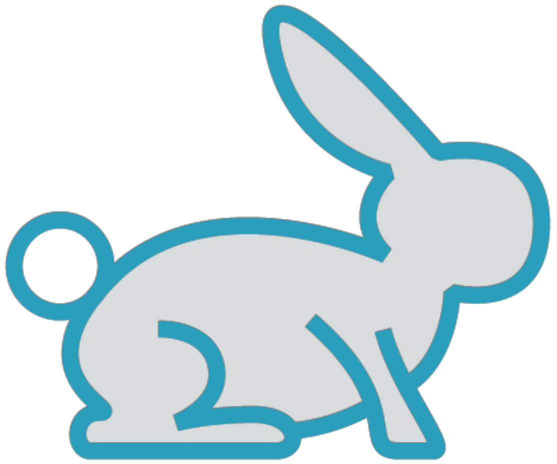
Cascaded on all relationships

- `@OneToOne`
- `@OneToMany`
- `@ManyToOne`
- `@ManyToMany`

Events that can be cascaded

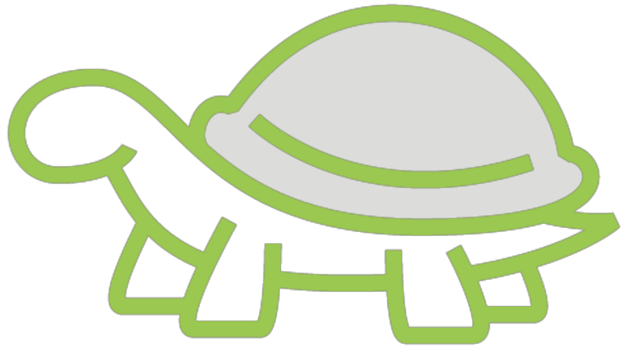
- PERSIST
- REMOVE
- MERGE
- ALL

Fetching Relationships



Eager

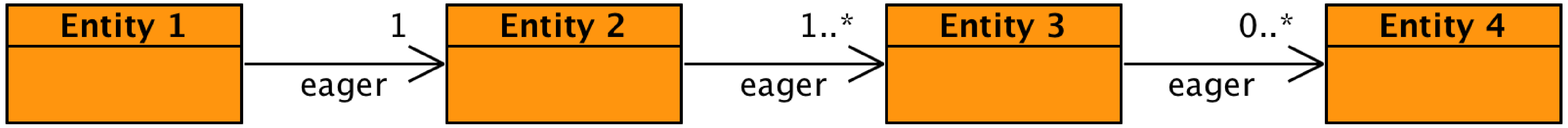
Immediately



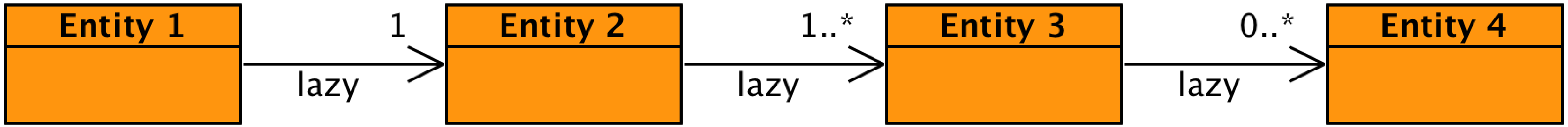
Lazy

Deferred

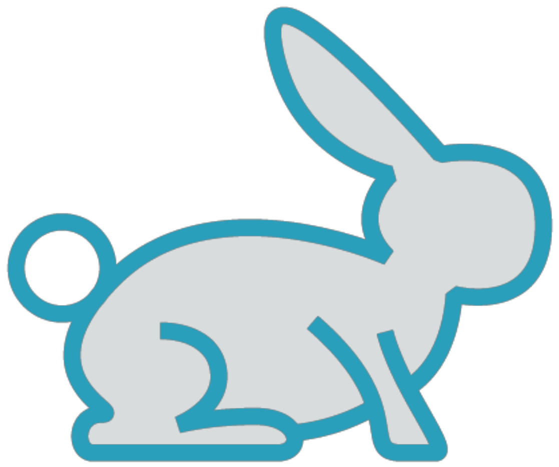
Eager Fetching

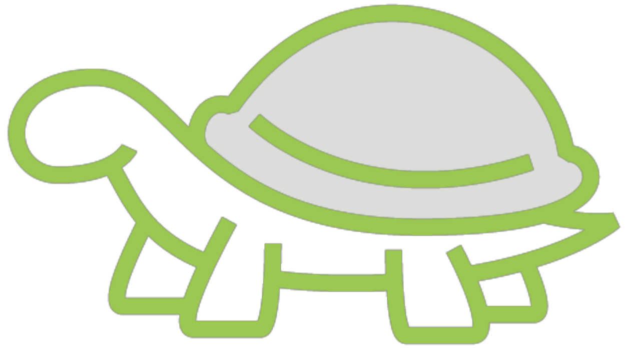


Lazy Fetching



Fetching Relationships





Eager

Brings all data in memory

Reduce database access

Lazy

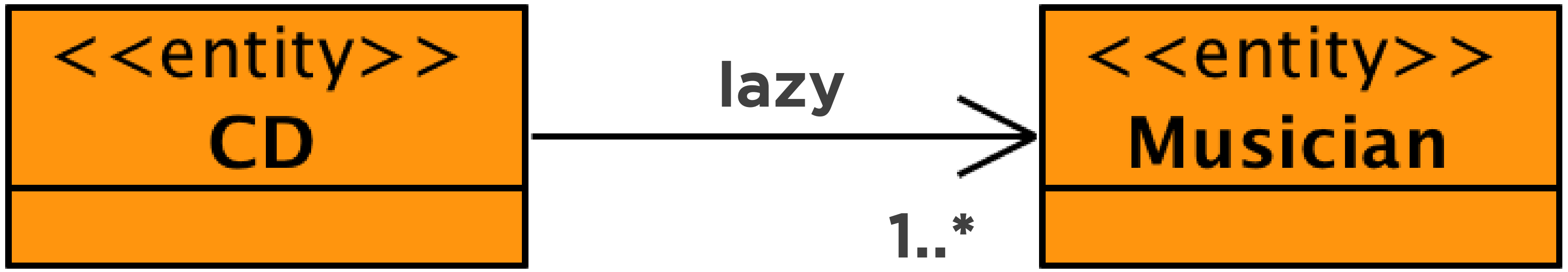
Brings some data in memory

Several database accesses

Default Fetching

Cardinality	RDBMS
@OneToOne	EAGER
@ManyToOne	EAGER
@OneToMany	LAZY
@ManyToMany	LAZY

Fetching One to Many Relationship



Fetch Attribute

```
@Entity
public class CD {
  // Attributes and Constructors

  @OneToMany(fetch = LAZY)
  private Set<Musician> musicians = new HashSet<>();
}



@Entity
public class Musician {
  // Attributes and Constructors
}
```

Fetch Attribute

```
@Entity
public class CD {
  // Attributes and Constructors

  @OneToMany(fetch = EAGER)
  private Set<Musician> musicians = new HashSet<>();
}



@Entity
public class Musician {
  // Attributes and Constructors
}
```

Demo

CRUD operations

One to many unidirectional relationship

CD and Musicians

Persist a CD

Cascading events

Inheritance in OOP

Inheriting attributes

Inheriting behavior

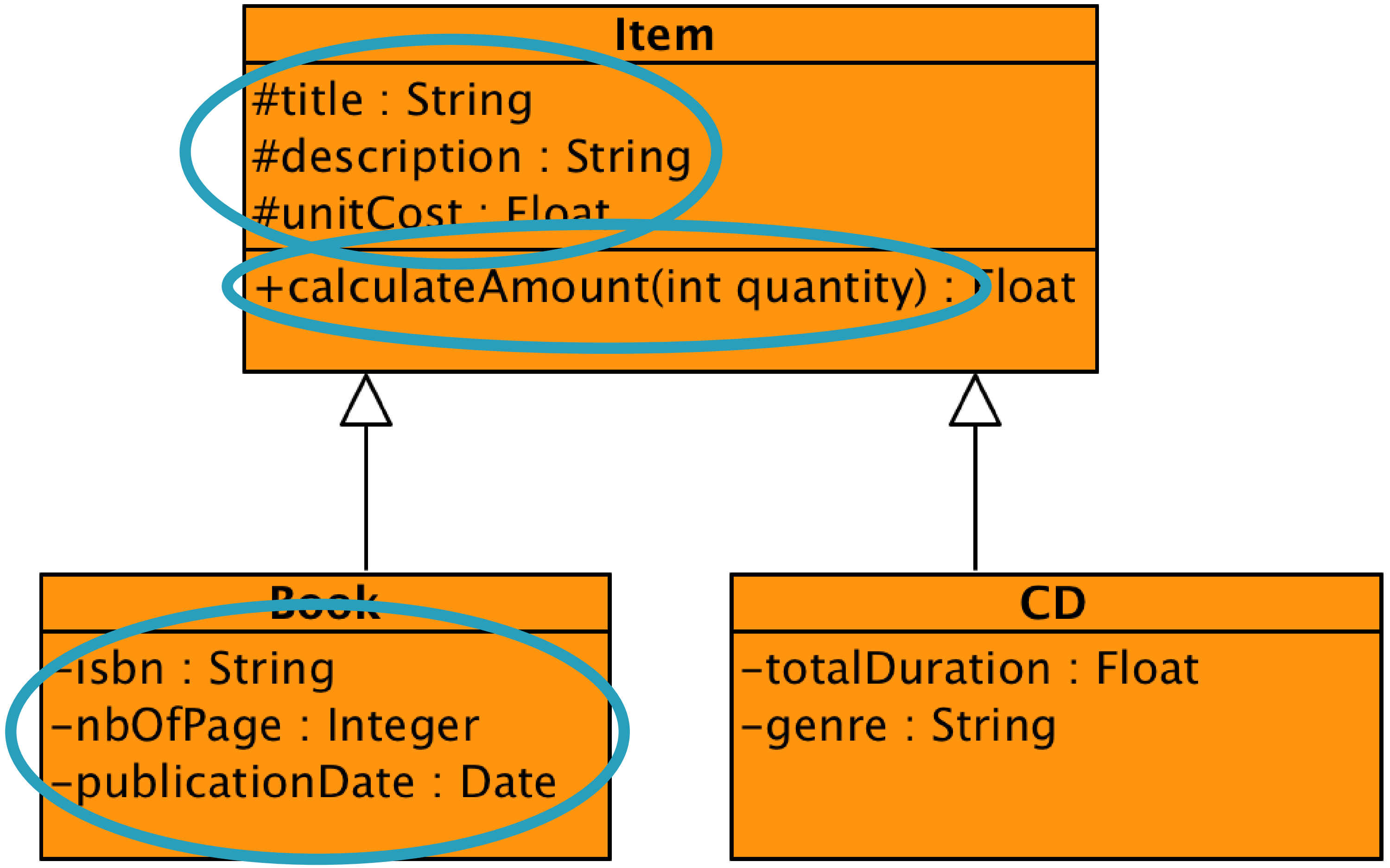
Java supports single-class inheritance

Implement several interfaces

No direction

No cardinality

Single-class Inheritance



Inheritance in RDBMS

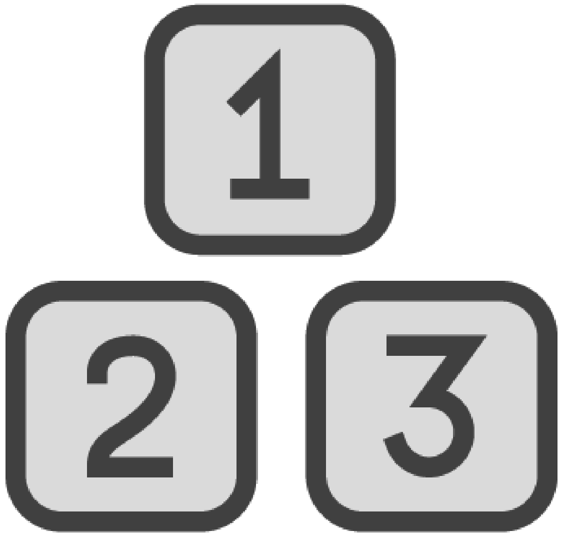
Completely unknown concept

Not natively implemented

Mapping inheritance

Throws in several twists

Inheritance in JPA



Hierarchical object model

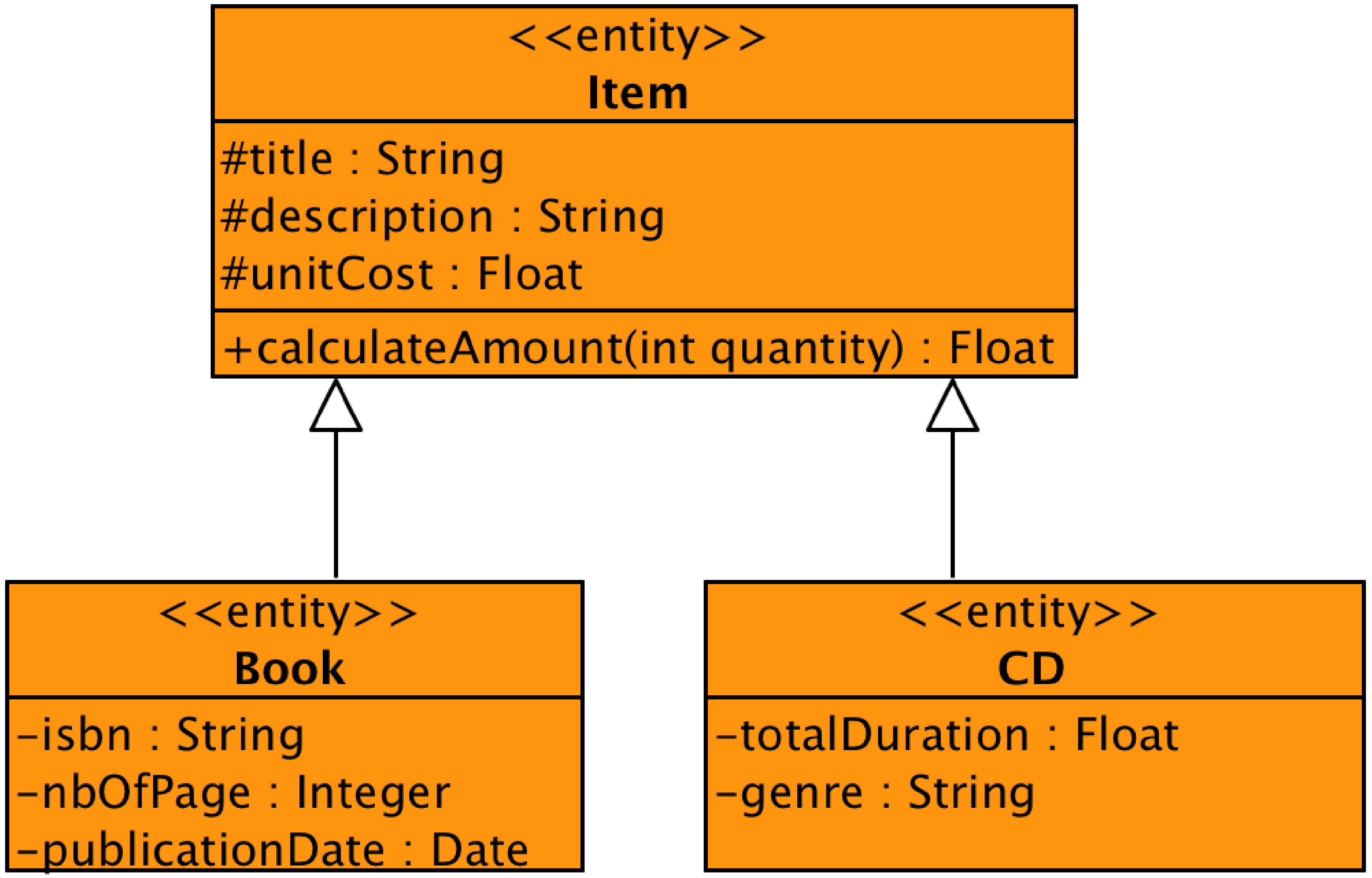
Flat relational structure

Three different strategies

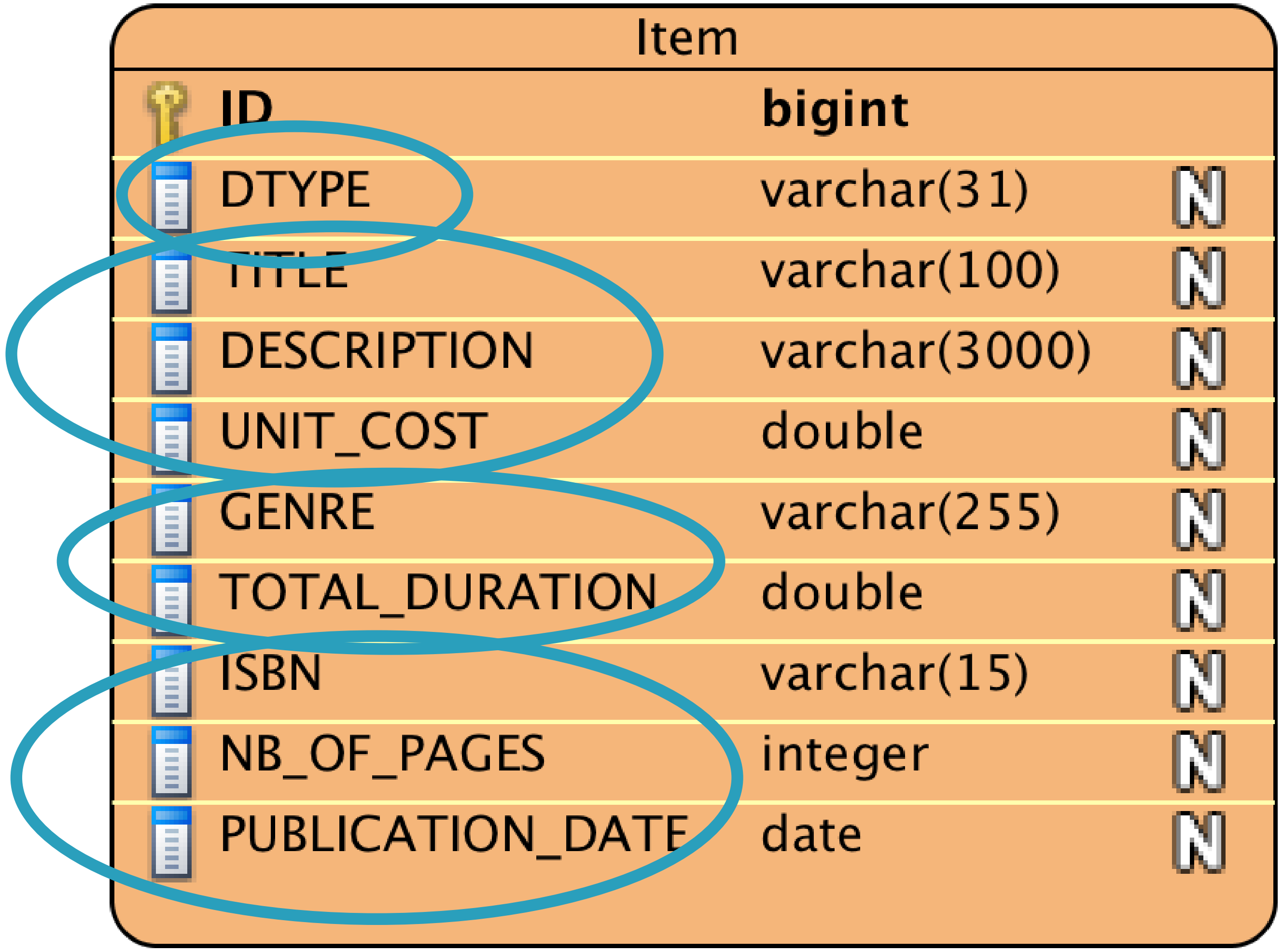
- single-table-per-class
- joined-subclass
- table-per-concrete-class

No strategy is better/worse

Entity Inheritance

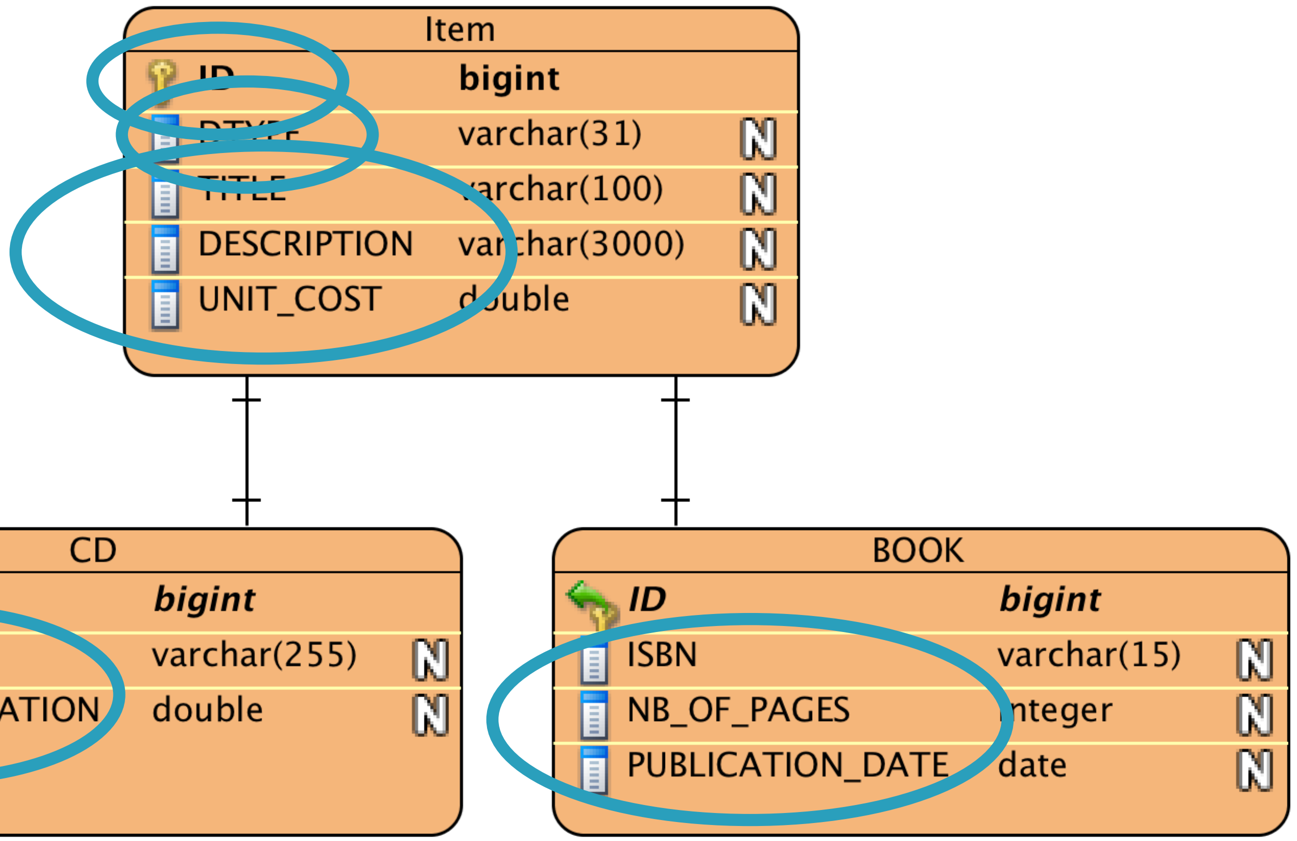


Single Table Strategy



Item		
ID	**bigint**	
DTYPE	varchar(31)	N
TITLE	varchar(100)	N
DESCRIPTION	varchar(3000)	N
UNIT_COST	double	N
GENRE	varchar(255)	N
TOTAL_DURATION	double	N
ISBN	varchar(15)	N
NB_OF_PAGES	integer	N
PUBLICATION_DATE	date	N

Joined-subclass Strategy



Item

ID	bigint	
DTYPE	varchar(31)	N
TITLE	varchar(100)	N
DESCRIPTION	varchar(3000)	N
UNIT_COST	double	N

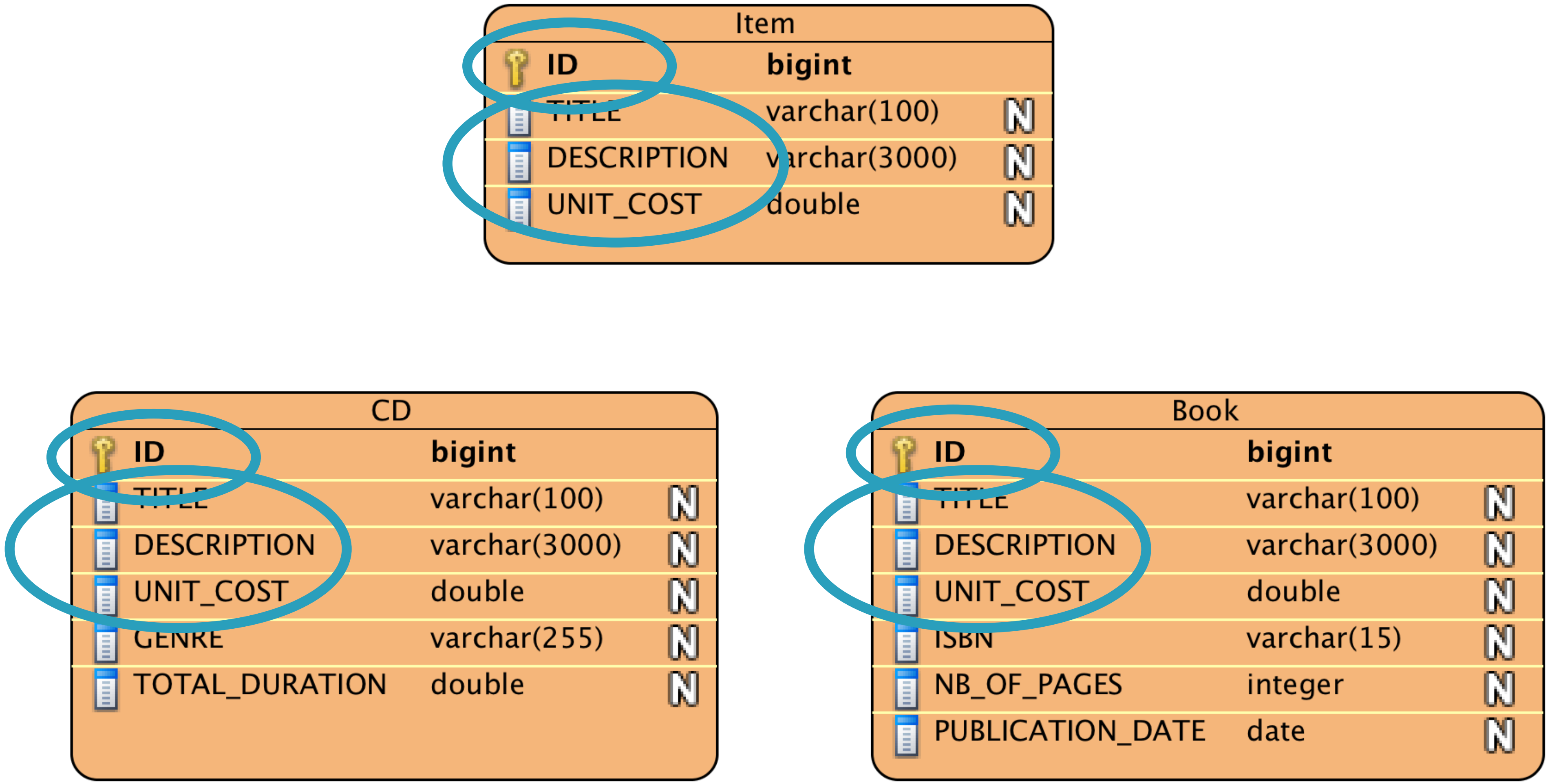
CD

ID	*bigint*	
GENRE	varchar(255)	N
TOTAL_DURATION	double	N

BOOK

ID	*bigint*	
ISBN	varchar(15)	N
NB_OF_PAGES	integer	N
PUBLICATION_DATE	date	N

Table-per-concrete-class Strategy



Item

ID	**bigint**	
TITLE	varchar(100)	N
DESCRIPTION	varchar(3000)	N
UNIT_COST	double	N

CD

ID	**bigint**	
TITLE	varchar(100)	N
DESCRIPTION	varchar(3000)	N
UNIT_COST	double	N
GENRE	varchar(255)	N
TOTAL_DURATION	double	N

Book

ID	**bigint**	
TITLE	varchar(100)	N
DESCRIPTION	varchar(3000)	N
UNIT_COST	double	N
ISBN	varchar(15)	N
NB_OF_PAGES	integer	N
PUBLICATION_DATE	date	N

Inheritance in JPA

```
@Entity
@Inheritance(strategy = SINGLE_TABLE)
public class Item { // Attributes and Constructors }


@Entity
public class Book extends Item { // Attributes and Constructors }


@Entity
public class CD extends Item { // Attributes and Constructors }
```

Inheritance in JPA

```
@Entity
@Inheritance(strategy = JOINED)
public class Item { // Attributes and Constructors }


@Entity
public class Book extends Item { // Attributes and Constructors }


@Entity
public class CD extends Item { // Attributes and Constructors }
```

Inheritance in JPA

```
@Entity
@Inheritance(strategy = TABLE_PER_CLASS)
public class Item { // Attributes and Constructors }


@Entity
public class Book extends Item { // Attributes and Constructors }


@Entity
public class CD extends Item { // Attributes and Constructors }
```

Discriminator

```
@Entity
@Inheritance(strategy = SINGLE_TABLE)
@DiscriminatorColumn(name = "DISC", discriminatorType=CHAR)
@DiscriminatorValue("I")
public class Item { // Attributes and Constructors }

@Entity
@DiscriminatorValue("B")
public class Book extends Item { // Attributes and Constructors }

@Entity
@DiscriminatorValue("C")
public class CD extends Item { // Attributes and Constructors }
```

Demo

Mapping inheritance

CD and Book extend from Item

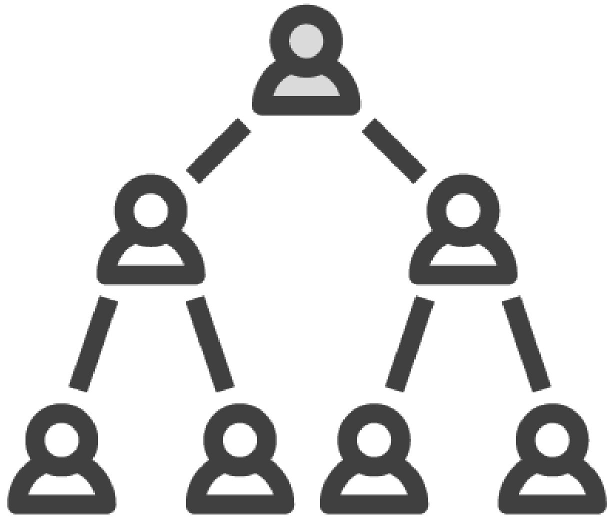
Default mapping

Customize mapping

Changing inheritance strategy

Database structure

Inheritance Hierarchy



Entities don't have to inherit from entities

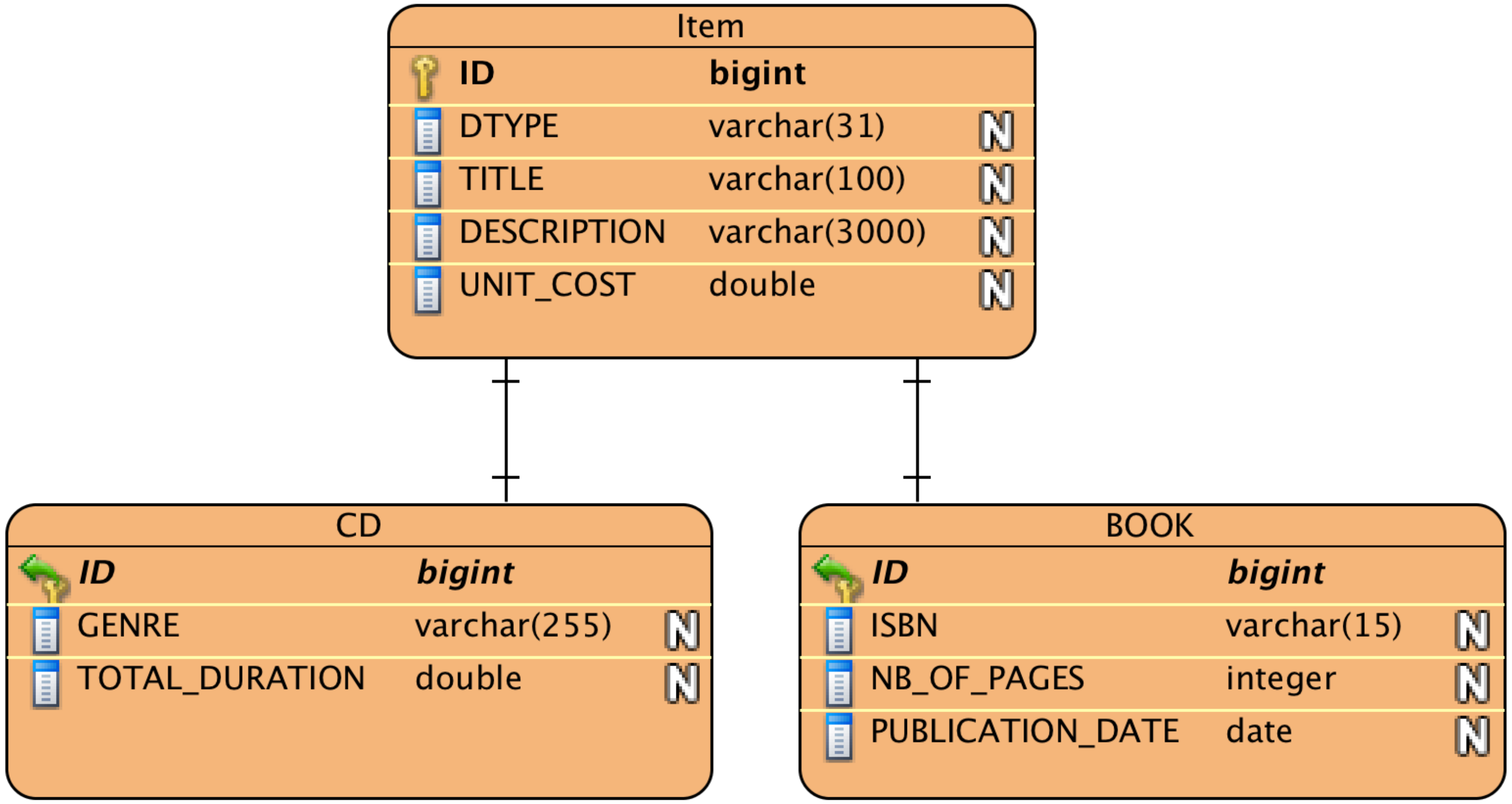
Entities can inherit from

- Transient classes
- Abstract entities
- Mapped superclasses

Inheriting from an Entity

```java
@Entity
@Inheritance(strategy = JOINED)
public class Item {
  @Id @GeneratedValue
  protected Long id;
  protected String title;
  @Column(length = 3000)
  protected String description;
  @Column(name = "unit_cost")
  protected Float unitCost;
}

@Entity
public class Book extends Item { }

@Entity
public class CD extends Item { }
```

Inheriting from an Entity



Item

ID	**bigint**	
DTYPE	varchar(31)	N
TITLE	varchar(100)	N
DESCRIPTION	varchar(3000)	N
UNIT_COST	double	N

CD

ID	***bigint***	
GENRE	varchar(255)	N
TOTAL_DURATION	double	N

BOOK

ID	***bigint***	
ISBN	varchar(15)	N
NB_OF_PAGES	integer	N
PUBLICATION_DATE	date	N

Inheriting from an Abstract Entity

```
@Entity
public abstract class Item {

  @Id @GeneratedValue
  protected Long id;
  protected String title;
  @Column(length = 3000)
  protected String description;
  @Column(name = "unit_cost")
  protected Float unitCost;
}

@Entity
public class Book extends Item { }

@Entity
public class CD extends Item { }
```

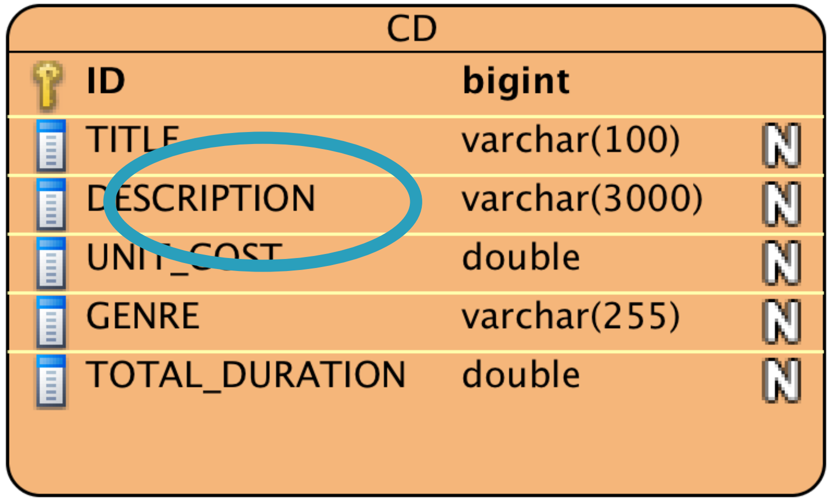
Inheriting from a Transient Class

```
public class Item {

  @Id @GeneratedValue
  protected Long id;
  protected String title;
  @Column(length = 3000)
  protected String description;
  @Column(name = "unit_cost")
  protected Float unitCost;
}

@Entity
public class Book extends Item { }

@Entity
public class CD extends Item { }
```

Inheriting from a Mapped Super Class

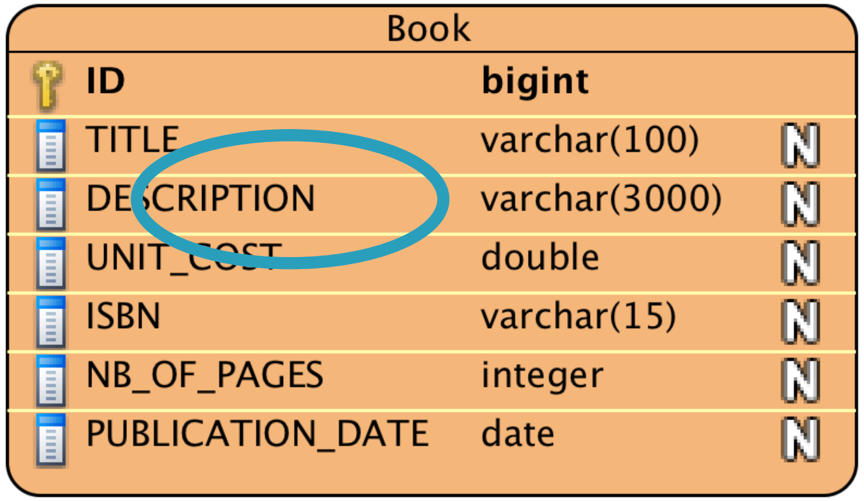
```
@MappedSuperclass
public class Item {

  @Id @GeneratedValue
  protected Long id;
  protected String title;
  @Column(length = 3000)
  protected String description;
  @Column(name = "unit_cost")
  protected Float unitCost;
}

@Entity
public class Book extends Item { }

@Entity
public class CD extends Item { }
```

Inheriting from a Mapped Super Class



CD

ID	**bigint**	
TITLE	varchar(100)	N
DESCRIPTION	varchar(3000)	N
UNIT_COST	double	N
GENRE	varchar(255)	N
TOTAL_DURATION	double	N



Book

ID	**bigint**	
TITLE	varchar(100)	N
DESCRIPTION	varchar(3000)	N
UNIT_COST	double	N
ISBN	varchar(15)	N
NB_OF_PAGES	integer	N
PUBLICATION_DATE	date	N

Demo

xxx

Summary

Relationships are easy to map

Cascade events

Fetch relationships

Inheritance is not native in RDBMS

Choose from three strategies

Inherit from different types of classes

Next Module

Query entities

Java Persistence Query Language

Dynamic queries

Named queries