cphbusiness

COPENHAGEN BUSINESS ACADEMY

Object Relational Mapping

Lars Mortensen

Literature: https://en.wikibooks.org/wiki/Java_Persistence

This second reference is for a specific database (ObjectDB) but since this database implements JPA, you can use the tutorial as a quick (or alternative) way to get started.

http://www.objectdb.com/java/jpa

Welcome to Flow-2
Fullstack for year 2019

cphbusiness

W-1 Object Relational Mapping

Java persistence API (JPA)

Restful Web Services

W-2 ----------

JSON

W-3 JavaScript required for REACT-Developers

AJAX

W-4 **CA-2**

Object Relational Mapping

cphbusiness

- Technique for converting data between tables in relational databases to objects in an object oriented language and vice versa
- Creates in effect a "virtual" object database

Why?

- Data management tasks in OO-programming are typically implemented by manipulating objects that are almost always **non-scalar** values
- Relational database management systems can only store and manipulate **scalar** values such as integers and strings organized within tables.

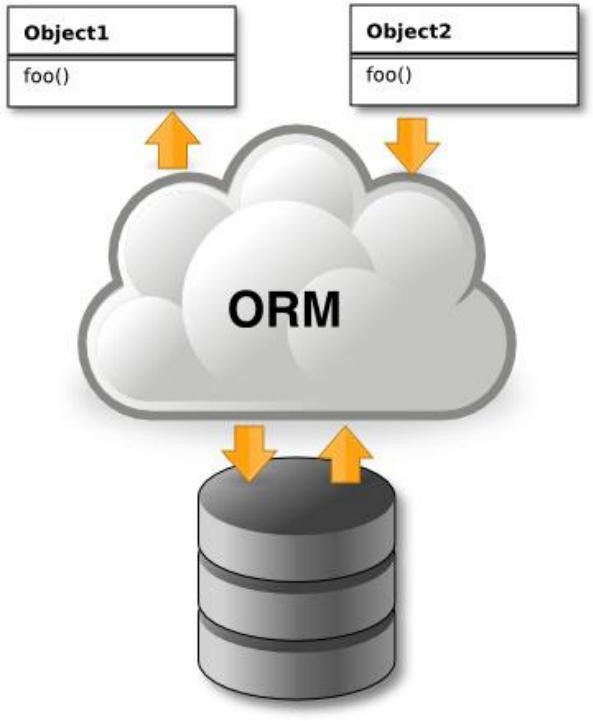
Without Object Relational Mapping

cphbusiness

Programmers must:

- Either convert the object values into groups of simpler values for storage in the database (and convert them back upon retrieval),
- Use only simple scalar values within the program.

Object-relational mapping is used to implement the first approach



To ORM or not to ORM ☺

cphbusiness

Comparison with traditional data access techniques

+ ORM typically reduces the amount of code that needs to be written
+ Avoids low level JDBC and SQL code
+ Provides database and schema independence
+ It allows us to use the OO-paradigm
+ Often protects against SQL Injection, but still: always validate inputs

- The high level of abstraction can obscure what is actually happening in the implementation code.
- Be aware of JPA's Convention-Over-Configuration Strategy
- Heavy reliance on ORM software has been cited as a major factor in producing poorly designed databases.

Our Goals when selecting a ORM Framework

cphbusiness

Take advantage of all the things Relational Databases do well

But doing it, without leaving all the all the things OO-languages do well

Have the illusion of only ”talking” OO, even when we manipulate data.

Do less work

Object-Relational Impedance Mismatch Issues

cphbusiness

- How are columns, rows, tables mapped to objects?
- How are relationships handled?
- How is OO inheritance mapped to relational tables?
- How is composition and aggregation handled?
- How are conflicting type systems between databases handled?
- How are objects persisted?
- How are different design goals handled?
- Relational model designed for data storage/retrieval
- Object Oriented model is about modelling behaviour

Object-Relational Impedance Mismatch Issues

cphbusiness

- Example – mismatch in data types
  - OO Languages such as Java C# have String and int data types
  - RDMBS such as MySQL has a varchar and smallint
- Although values are stored and manipulated differently the database driver (JDBC in Java) handles conversions automatically

Object-Relational Impedance Mismatch Issues

cphbusiness

- Example – collections versus tables
  - Java/C# use **collections** to manage lists of objects
  - Databases uses **tables** to manage lists of entities
- Example – blobs versus objects
  - Databases uses **blobs** to manage large objects as simple binary data
  - Java/C# use **objects** with **behaviors**

Object Relational Mapping Tools

cphbusiness

NHIBERNATE

HIBERNATE

mongoose

elegant mongodb object modeling for node.js

doctrine

BOOKSHELF.JS

Sequelize

http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software#.NET

Introduction to Java Persistence API JPA

cphbusiness

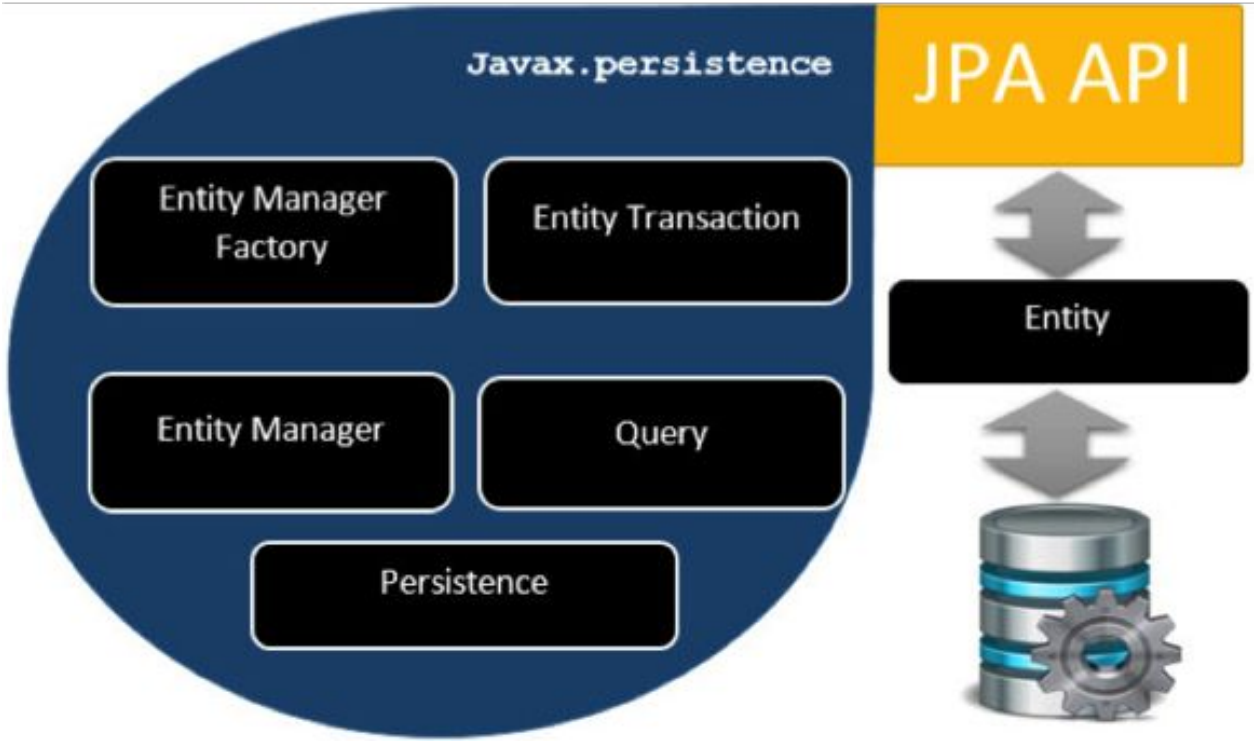
Java Persistence consists of four areas:

- The Java Persistence API
  - API which provides Java developers with an object/relational mapping facility for managing relational data in Java applications
- The query language (JPQL)
  - Query language allows us to write portable queries that work regardless of the underlying data store
- The Java Persistence Criteria API (OO-Queries)
  - Queries written using Java APIs, which are type safe, and portable.
- Object/relational mapping metadata

http://docs.oracle.com/javaee/7/tutorial/doc/persistence-intro.htm#BNBPZ

JPA Architecture

cphbusiness



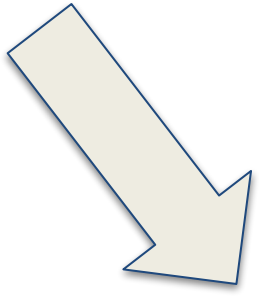
More Info: https://www.tutorialspoint.com/jpa/jpa_architecture.htm

http://docs.oracle.com/javaee/7/tutorial/doc/persistence-intro.htm#BNBPZ

Which JPA to use?

cphbusiness

https://en.wikibooks.org/wiki/Java_Persistence/Persistence_Products



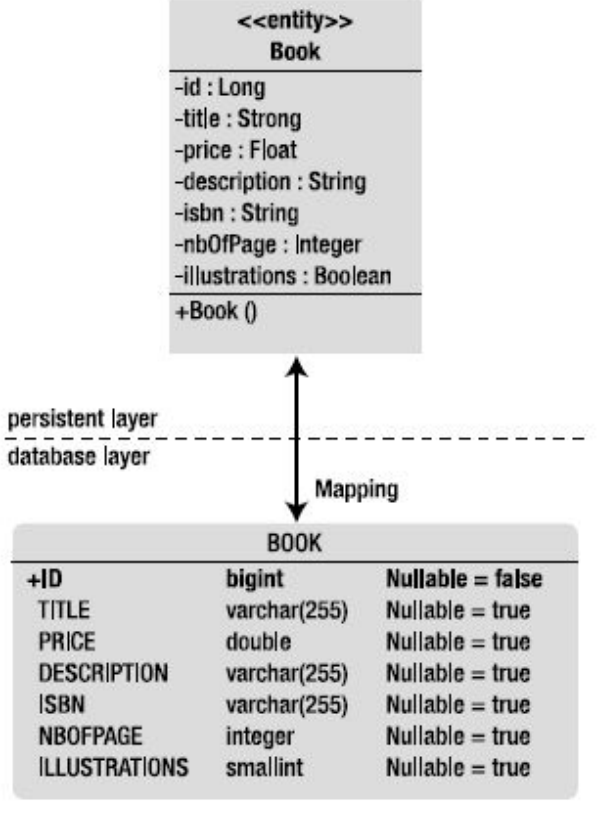
The one we will be using

https://en.wikibooks.org/wiki/Java_Persistence/EclipseLink

JPA - Entities

cphbusiness

An entity is a lightweight persistence domain object. Typically, an **Entity** represents a **table** in a relational database, and each *entity-instance* corresponds to a *row* in that table.



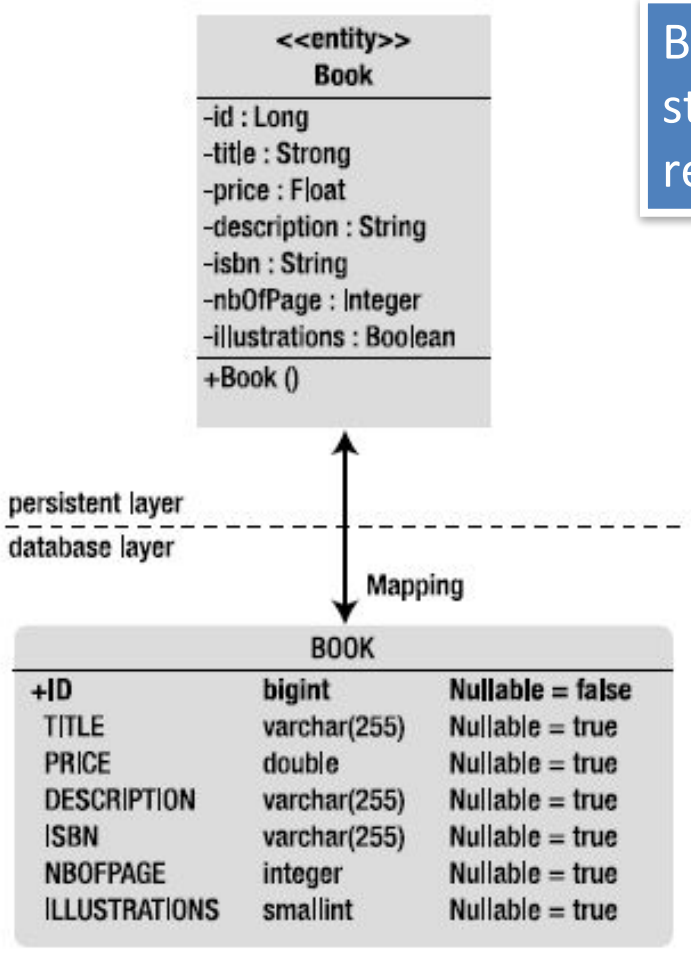
Requirements for Entity Classes

cphbusiness

- The class must be annotated with the **`@Entity`** annotation.
- The class must have (at least) a **public or protected, no-argument constructor**.
- The class must not be declared `final`. No methods or persistent instance variables must be declared `final`.
- If an entity instance is passed by value as a detached object, the class must implement the `Serializable` interface.
- Entities may extend both entity and non-entity classes, and non-entity classes may extend entity classes.
- Persistent instance variables must be declared private, protected, or package-private and can be accessed directly only by the entity class's methods. Clients must access the entity's state through accessor or business methods.

Mapping Entities

cphbusiness



Because of the **Convention-Over-Configuration** strategy used by the EE-framework, this is all that is required to turn the class in to a fully fledged Entity Class

```
@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
}
```

BUT ALSO, because of the **Convention-Over-Configuration** strategy, you often get POORLY designed databases. Use the documentation to see how you can control **size**, **nullable, Table Names, Column Names** and much more for the matching Tables

Mapping Entities and what to skip when you read the wiki-book

cphbusiness

https://en.wikibooks.org/wiki/Java_Persistence/Mapping

Getting Started ☺

cphbusiness

The Entity Manager

cphbusiness

JPA provides a runtime API defined by the javax.persistence. package.
The main runtime class is the **EntityManager** class.

The **`EntityManager`**:

- manages the state and life cycle of entities as well as querying entities within a persistence context.
- provides an API for creating queries, accessing transactions, and finding, persisting, merging and deleting objects
- can lock entities for protecting against concurrent access by using optimistic or pessimistic locking.

The Entity Manager

cphbusiness

```
Book2 book = new Book2();
book.setDescription("..");
 //...
EntityManagerFactory emf;
emf = Persistence.createEntityManagerFactory("pu-x");
EntityManager em = emf.createEntityManager();
Try{
  em.getTransaction().begin();
  em.persist(book);
  em.getTransaction().commit();
}
finally{
  em.close();
}

em.remove(book);
```

Just a POJO (Plain Old Java Object)

Entities can be used as regular objects by different layers of an application

and become managed by th
entity manager when
we need to load or insert d
into the database

Now the book is **Managed**

Again, just a POJO (**detached**)

Obtaining an Entity Manager

cphbusiness

Obtaining an Entity Manager depends on which of the following strategies are used

Application-Managed Entity Managers (what we will be using)

Uses the **Persistence** class to create an **EntityManagerFactory**.

```
emf = Persistence.createEntityManagerFactory("pu-x");
EntityManager em = emf.createEntityManager();
```

With an application Managed EntityManager, we as programmers are in charge of creating, closing and handle transactions.

Container-Managed Entity Managers

In a container-managed evironment (i.e. Glassfish for us) the usual way to obtain a EntityManager is by injection:

```
@PersistenceContext
EntityManager em;
```

The component running in a container (servlet, EJB, web service, etc.) doesn't need to create or close the entity manager, as its life cycle is managed by the container.

Using EntityManager and EntityManagerFactory in JSE

cphbusiness

In JSE the **EntityManager** must be closed when your application is done with it. The life-cycle of the EntityManager is typically per client, or **per request**.

EntityManagerFactory can be shared among multiple threads or users, but the EntityManager should not be shared

Use the template given below for future facade classes

```
public class CustomerFacade {
  EntityManagerFactory emf;

  public CustomerFacade(EntityManagerFactory emf) {
    this.emf = emf;
  }

  EntityManager getEntityManager(){
     return emf.createEntityManager();
  }

  // Use this template for a method that uses the
  // EntityManager
  public Customer getCustomer(int id){
    EntityManager em = getEntityManager();
    try{
      // Use the entity manager
    }
    finally{
      em.close();
    }
  }

}
```

Persistence Units and Persistence.xml

cphbusiness

A persistence unit defines the details that are required when you acquire an entity manager.

To package your EclipseLink JPA application, you must configure the persistence unit during the creation of the **persistence.xml** file.

Define each persistence unit in a persistence-unit element in the persistence.xml file.

Important info, about the [Persistence Unit]

Limit problems related to Persistence Units by following these simple rules/hints

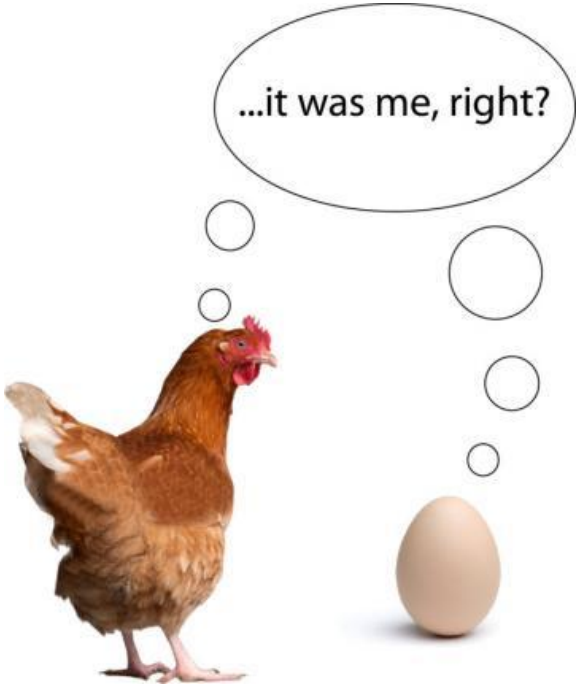
- A **Persistence.xml** file is created automatically by NetBeans first time you create an Entity Class.
- NEVER commit your persistence.xml file. Add persistence.xml to your **.gitignore** file
- You can always create a new Persistence.xml file, pointing to your own database, using the NetBeans Wizard
- Remember to change the persistence.xml file, before you **deploy** your project to DigitalOcean

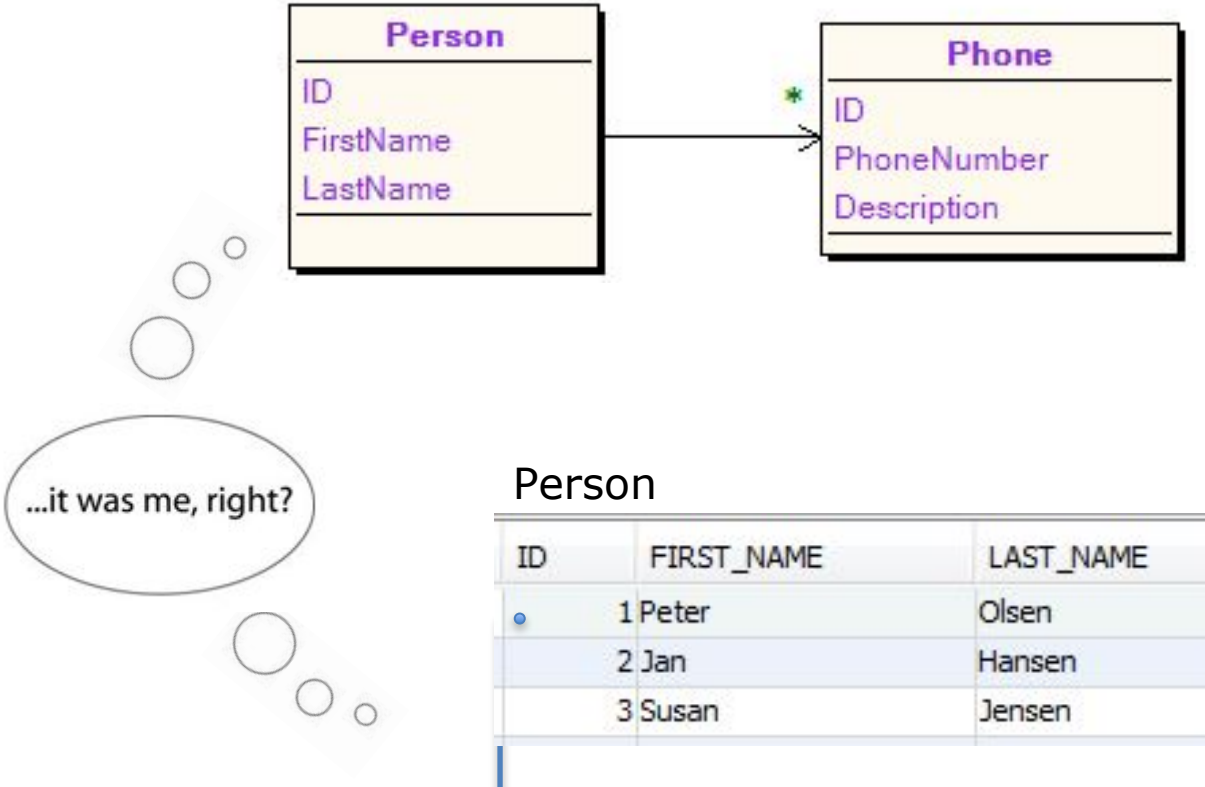
The Chicken or the Egg

cphbusiness

With NetBeans and JPA we can either generate tables from existing Entity-classes or generate the Entity-classes from existing tables.

So which way should we go?





Person

ID	FIRST_NAME	LAST_NAME
1	Peter	Olsen
2	Jan	Hansen
3	Susan	Jensen

Phone

ID	PHONE_NUMBER	DESCRIPTION	P_ID
1	33333333	private	1
2	12345678	work	1

Entity Classes from existing tables

cphbusiness

Tables from Entity Classes

cphbusiness

JPA – Object Relational Mappings

The following will far from introduce all OR-mapping annotations but should give you a general overview of what is possible.

Use the following for details:

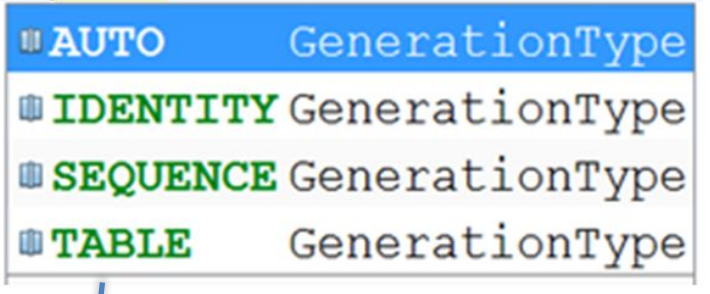
https://en.wikibooks.org/wiki/Java_Persistence.

The link below is for another specific database (ObjectDB) but is relevant for 99% of its content, and very easy to read
http://www.objectdb.com/java/jpa

Primary Keys

cphbusiness

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



Good choice for an MySQL database
AUTO_INCREMENT

Good choice for a Database which support
Sequences (Oracle or JavaDb)

Can be used by "all" DataBase Vendors.
Often the selected Strategy for AUTO

Auto Generation of ID's - Sequences
(for databases that supports this, like Oracle and Derby)

cphbusiness

We can control how a Sequence is generated or map it to an existing sequence.

```
Class Book{
...
  @ID
  @GeneratedValue(strategy = GenerationType.SEQUENCE,generator="s1")
  @SequenceGenerator(name="s1",sequenceName = "My_SEQ",
                     initialValue = 200000,allocationSize = 1)
```

These annotations will:

- Create a sequence as sketched below, if we are creating tables from Entities
- Map to the existing sequence if we are creating Entities from tables

Table Create Script

```
DROP SEQUENCE My_SEQ RESTRICT;
CREATE SEQUENCE My_SEQ START WITH 200000 INCREMENT BY 1;
```

Auto Generation of ID's - Tables

cphbusiness

All databases can use a separate Table as their strategy to provide a "next id" value

This is usually the default when you select: **GenerationType.AUTO**

```
Class Book{
...
  @Id
  @GeneratedValue(strategy = GenerationType.TABLE,generator="s1")
  @TableGenerator(name="s1",table = "My_SEQ",
  initialValue = 200000,allocationSize = 50)
```

These annotations will:

- **Create a table for auto id's if we are creating tables from Entities**
- **Map to the existing table if we are creating Entities from tables**

Auto Generation of ID's - IDENTITY

cphbusiness

MySQL does not provide **Sequences** to generate a unique value for new Rows.
MySQL provides a construct AUTO_INCREMENT as sketched below:

```
CREATE TABLE Persons
(
  ID int NOT NULL AUTO_INCREMENT,
  Name varchar(80),
  PRIMARY KEY (ID)
)
```

This is how you Signal JPA to use this strategy for automatic key generation:

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
 private Integer id;
```

There is no way, as for the other two strategies, to provide a start value and allocation size via JPA.
Se exercises for an example script you can use to insert data without conflicting with JPA.

Composite Primary Keys

cphbusiness

Composite primary keys can be defined in two ways:

Using an **Id Class**

```
@Entity @IdClass(ProjectId.class)
public class Project {
    @Id int departmentId;
    @Id long projectId;
     :
}

Class ProjectId {
    int departmentId;
    long projectId;
}
```

Using an **Embeddable Class**

```
@Entity
public class Project {
  @EmbeddedId ProjectId id;
    :
}

@Embeddable
Class ProjectId {
    int departmentId;
    long projectId;
}
```

The main purpose of both the `IdClass` and the `Embeddable` Class is to be used as the structure passed to the `EntityManager` `find()` and `getReference()` AP

Date Time and Transient Properties

cphbusiness

```
@Temporal(TemporalType.DATE)
private Date dateOfBirth;

@Temporal(TemporalType.TIMESTAMP)
private Date creationDate;

@Transient
private int age;
```

Enums

cphbusiness

```
public enum CustomerType {
   GOLD,
   SILVER,
   IRON,
   RUSTY
}
```

```
public class Customer {
   ...
   @Enumerated(EnumType.STRING)
   private CustomerType customerType;

}
```

Collections and Maps of Basic Types

cphbusiness


```
@ElementCollection(fetch = FetchType.LAZY)
private List<String> hobbies= new ArrayList();
```

```
@ElementCollection(fetch = FetchType.LAZY)
@MapKeyColumn(name = "PHONE")
@Column(name = "Description")  //Name of the Value column
private Map<String, String> phones = new HashMap();
```