11ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 11

Persistencia con Jakarta

Dr. Omar Mendoza González

omarmendoza564@aragon.unam.mx

DGTIC

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Dirección General de Cómputo y de Tecnologías de información y Comunicación

Dirección de Docencia en TIC

50 AÑOS Educación Continua 1971 - 2021

Objetivo

- El participante integrará a sus aplicaciones de tipo empresarial los mecanismos apropiados de persistencia para almacenar la información en una base de datos.

Jakarta EE

Jakarta Persistence

- La API de persistencia de Java (JPA), en 2019 cambió su nombre a **Jakarta Persistence**, es una especificación de interfaz de programación de aplicaciones de Java que describe la gestión de datos relacionales en aplicaciones que utilizan la plataforma Java, edición estándar y la plataforma Java, edición empresarial/Jakarta EE.

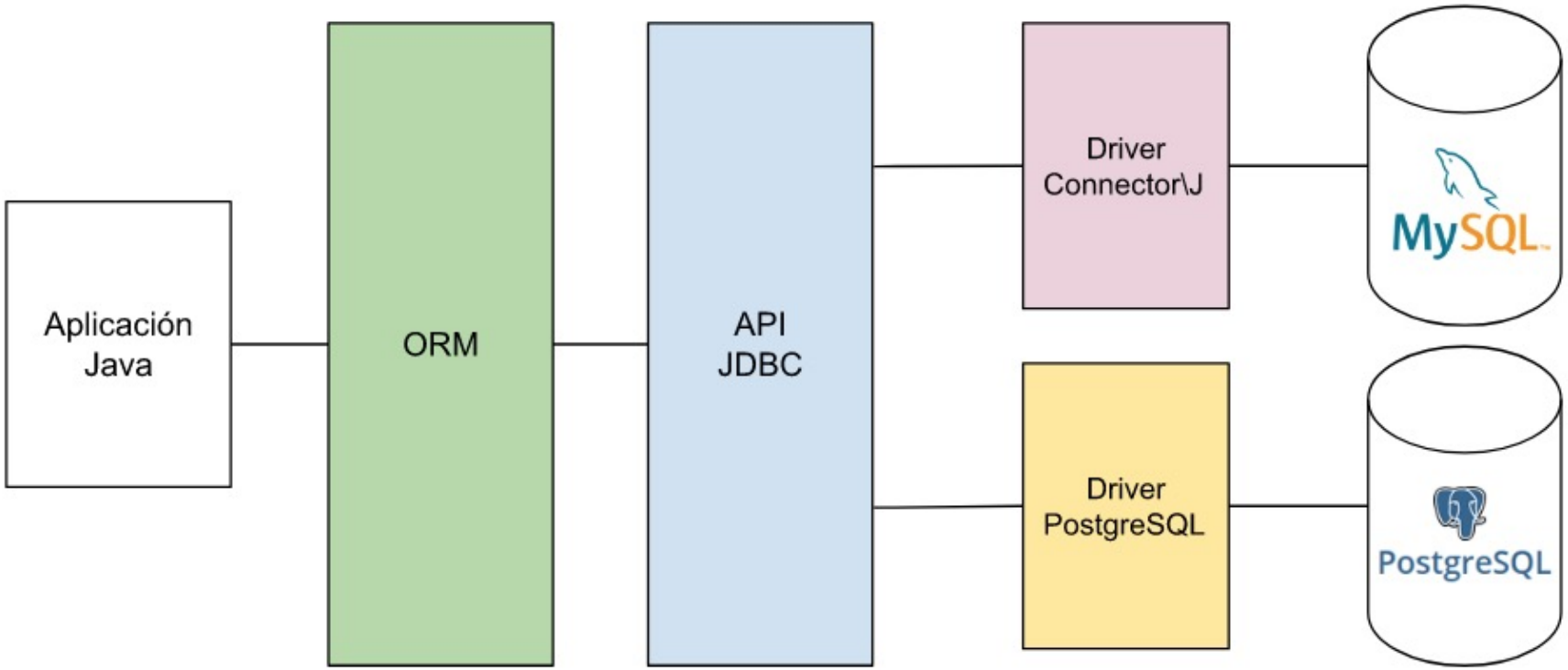
Jakarta Persistence

- La API de persistencia de Jakarta consta de cuatro áreas:
  - La API de persistencia de Jakarta
  - La API de criterios de persistencia de Jakarta
  - El lenguaje de consulta de persistencia de Jakarta
  - Metadatos de mapeo relacional de objetos

Jakarta Persistence

- **JPA no proporciona clase alguna para poder trabajar** con la información.
- Lo que hace es proveer una serie de interfaces que se pueden utilizar para implementar la capa de persistencia de una aplicación, apoyándose en alguna implementación concreta de JPA.
- **En la práctica significa que lo que se utiliza es una biblioteca de persistencia que implemente JPA**, no JPA directamente.

Jakarta Persistence



Entidades

- Una entidad es un objeto de dominio ligero y persistente.

- Es una representación Java de la tabla de la base de datos que tiene características como
  - **Persistencia**
  - **Identidad**
  - **Transaccionalidad**
  - **Granularidad**

Clase de entidad

- Una clase de entidad debe anotarse con la anotación ***jakarta.persistence.Entity*** o debe estar indicada en el descriptor XML como una entidad.
- Debe tener un *constructor sin argumentos* que debe ser público o protegido.
- Debe ser una clase de nivel superior, una enumeración o interfaz no debe designarse como una entidad.
- No debe ser *final*.

Clase de entidad

- Ningún método o variable de instancia persistente de una clase de entidad puede ser final.

- Tanto las clases abstractas como las concretas pueden ser entidades.

- Las entidades pueden extender las clases que no son de entidad, así como las clases de entidad, y las clases que no son de entidad pueden extender las clases de entidad.

Persistencia

- El estado de una entidad se puede representar en un **almacén de datos** y se puede acceder a él en un **momento posterior**.
- **No se conserva automáticamente** y para que tenga una representación duradera, la aplicación debe invocar activamente un *método de persistencia*.
- Se deja el control sobre la persistencia en manos de la aplicación.
- La aplicación tiene la flexibilidad de manipular datos y realizar la lógica de negocios en la entidad.
- Las entidades pueden ser manipuladas **sin que necesariamente sean persistentes**.

Identidad

- Una entidad tiene una **identidad de objeto**, pero cuando existe en la base de datos, también tiene una **identidad persistente**.
- La identidad del objeto es simplemente la diferenciación entre los objetos que ocupan la memoria.
- La identidad persistente, es la clave que **identifica de forma única** una instancia de entidad y la distingue de todas las demás instancias del mismo tipo de entidad (PK).
- Una entidad tiene una identidad persistente cuando existe una representación de ella en el almacén de datos, **un renglon en una tabla de base de datos**.

Transaccionalidad

- Las entidades podrían llamarse *cuasi-transaccionales*.

- Aunque se pueden crear, actualizar y eliminar en cualquier contexto, estas operaciones normalmente **se realizan dentro del contexto de una transacción** porque se requiere una transacción para que los cambios se confirmen en la base de datos.

- Los cambios realizados en la BD tienen *éxito o fallan atómicamente*, por lo que la vista persistente de una entidad debería ser transaccional.

- En la memoria, las entidades *pueden cambiarse sin que los cambios persistan*.

Granularidad

- Las entidades están destinadas a ser **objetos detallados** que tienen un conjunto de estados agregados normalmente almacenados en un solo lugar, como una fila en una tabla, y normalmente tienen relaciones con otras entidades.
- Son **objetos de dominio empresarial** que tienen un significado específico para la aplicación que accede a ellos.
- Las entidades de persistencia estan destinadas a estar en el extremo más pequeño del espectro de granularidad.
- Idealmente deben diseñarse y definirse como objetos bastante livianos de un tamaño comparable al del objeto Java promedio.

Mapeo de Entidades

- Se refiere a definir cómo se relacionan las clases de la aplicación con los elementos del sistema de almacenamiento.
- Si se considera **el caso común de acceso a una base de datos relacional**, sería definir:
  - La relación existente **entre las clases** de la aplicación **y las tablas** de la base de datos
  - La relación **entre las propiedades** de las clases **y los campos** de las tablas
  - La relación **entre diferentes clases y las claves externas** de las tablas de la base de datos

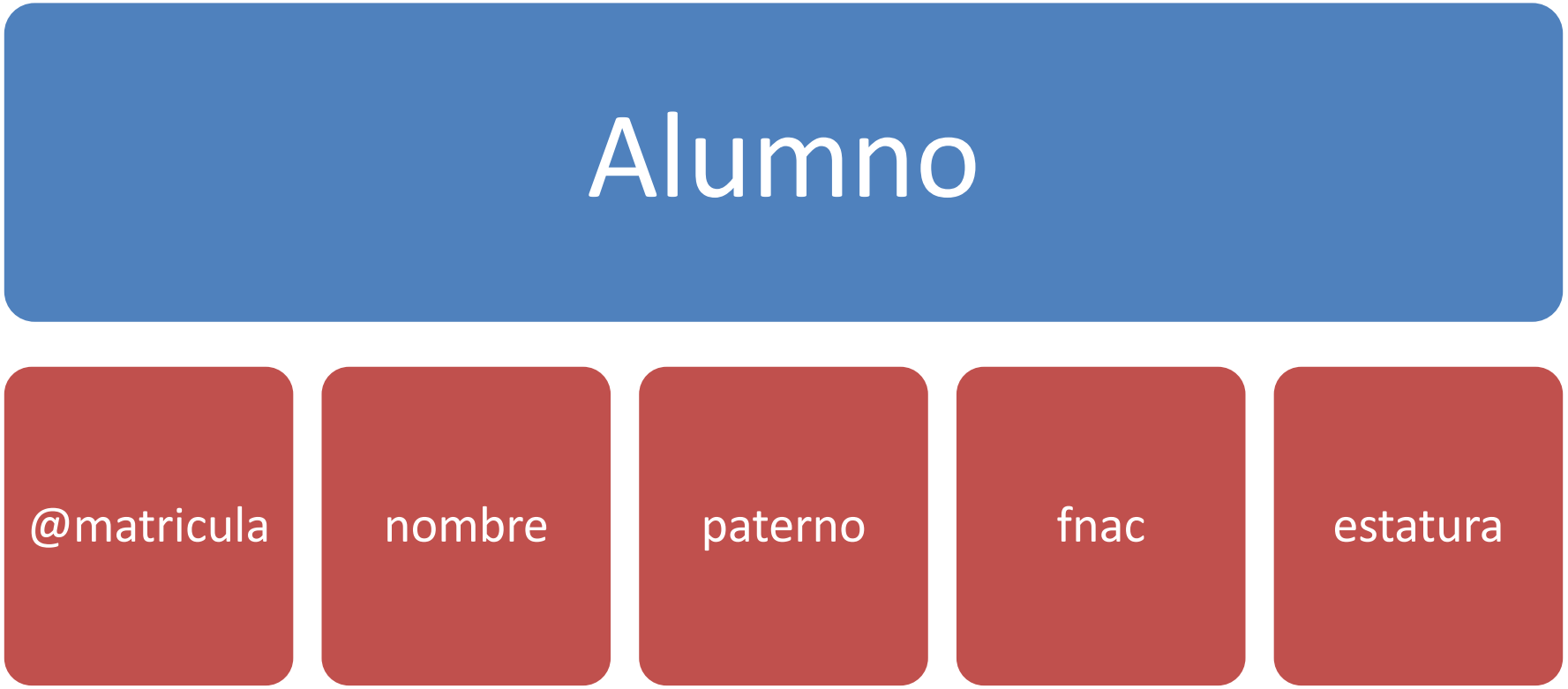
Mapeo de Entidades

```
@Entity
public class Alumno {
   @Id
   private String matricula;
    ...

   @ManyToOne private List<Calificaciones> calificaciones;

   ...
}
```

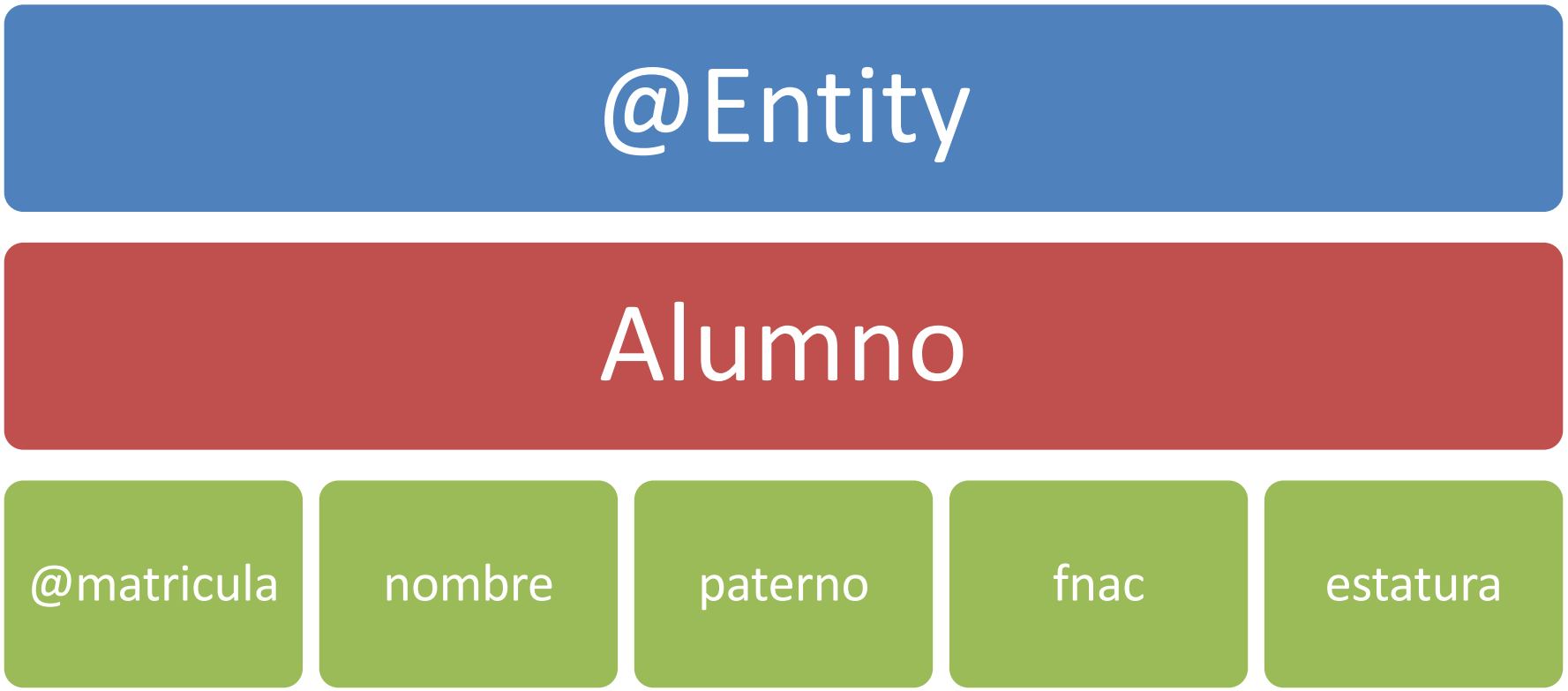
Mapeo de Entidades

- Una clase entidad no tiene que heredar ni implementar nada, es POJO estándar que cumple con los siguientes requisitos.
  - Está anotada con @Entity.
  - Puede tener varios constructores, pero siempre debe existir el constructor vacío y ser público o protegido.
  - Ni la clase ni los atributos persistibles pueden ser finales.
  - Las entidades pueden especializar clases que no lo sean y viceversa.
  - Debe tener un atributo utilizado como identificador, o heredar de una entidad que ya lo tenga
  - Se aceptan clases abstractas.

Entidades



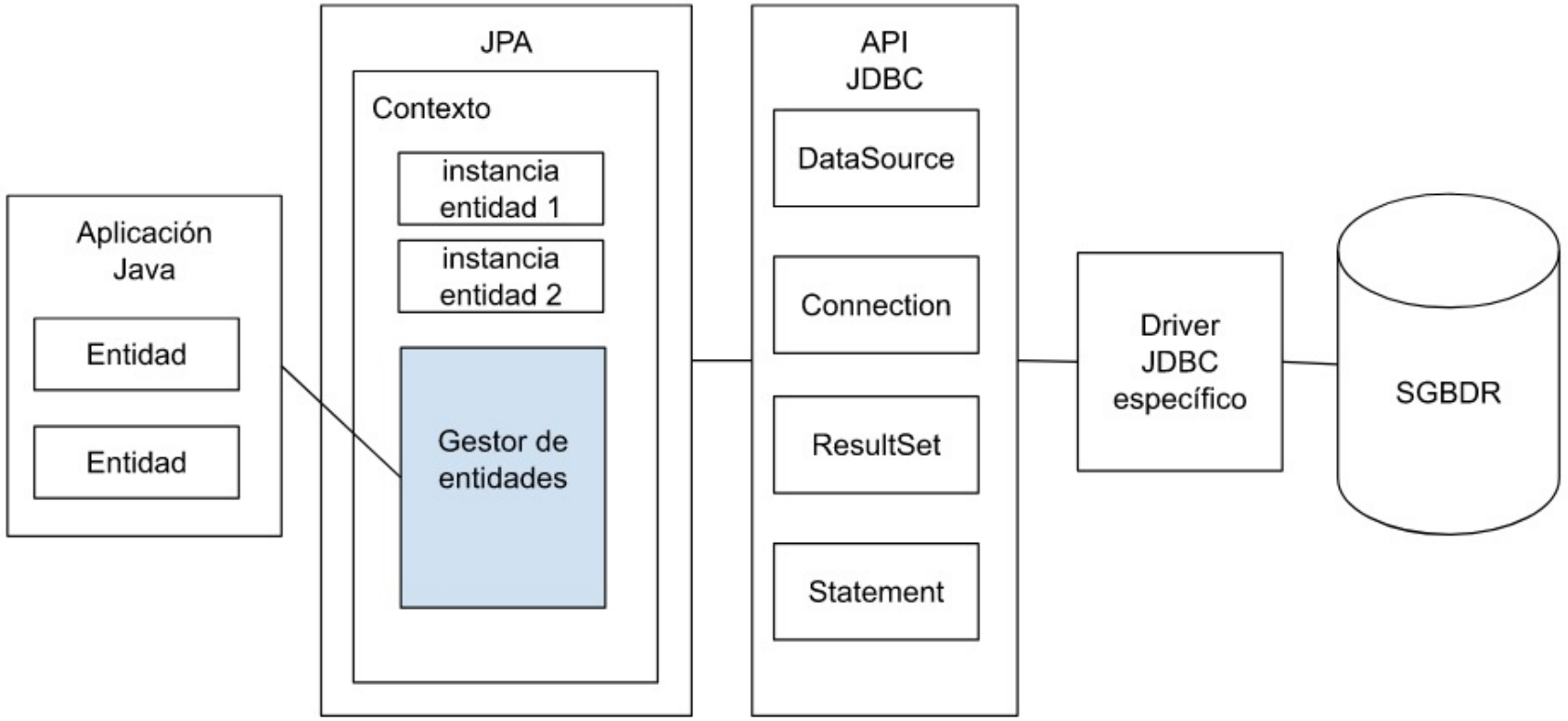
Entidades



Entity Manager

- Consiste en una interfaz llamada *jakarta.persistence.EntityManager*
- Proporciona la interacción con un contexto de persistencia asociado a la misma
- Con esta API se obtienen, añaden, modifican y eliminan objetos del contexto, y el proveedor de JPA, propaga esos cambios hacia la base de datos.
- Se tienen objetos y métodos, en lugar de SQL y llamadas a la API JDBC.
- Además, EntityManager también permite ejecutar consultas escritas con JPQL, SQL o la API Criteria.

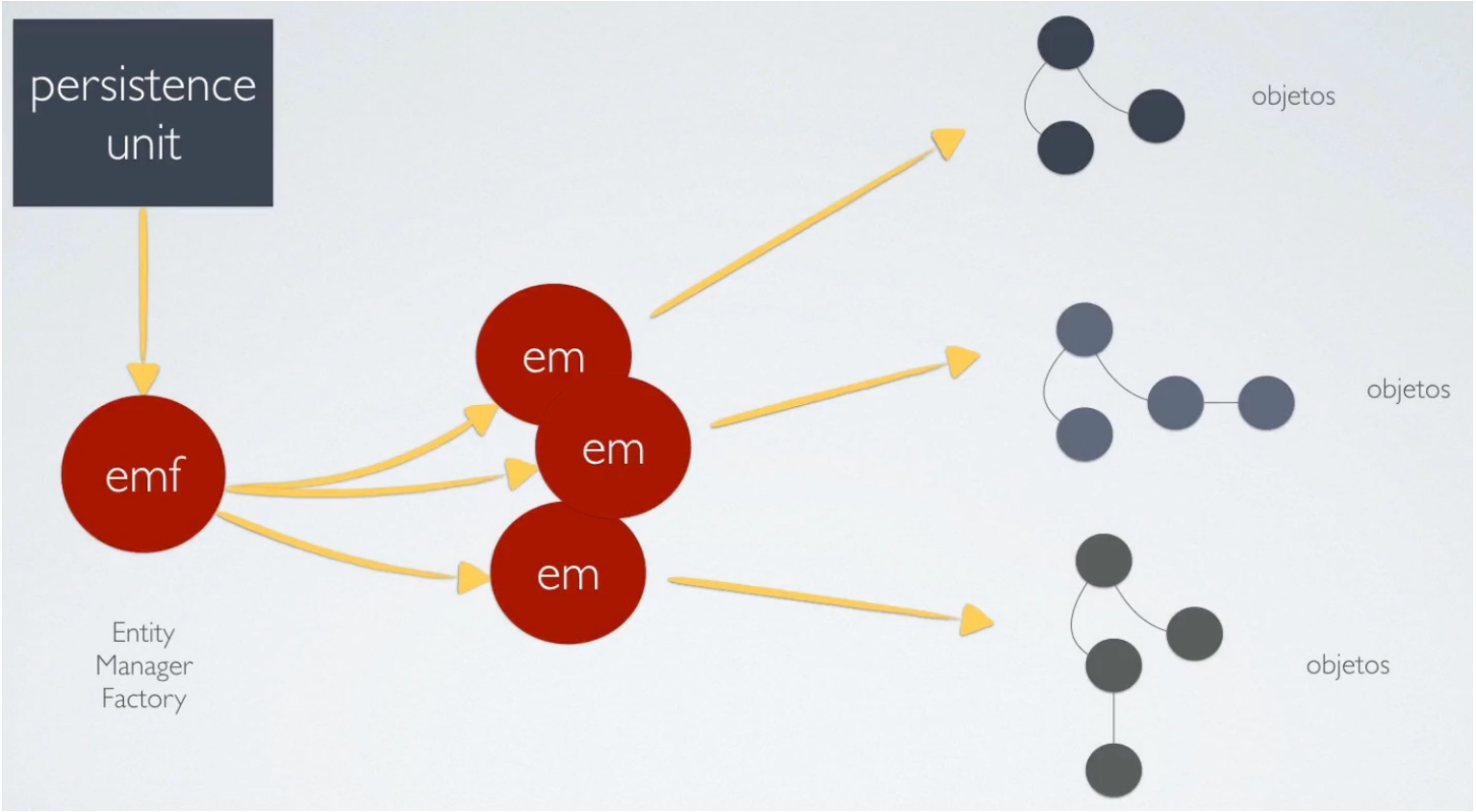
Entity Manager



Entity Manager

- Todos los Entity Manager provienen de fábricas de tipo *jakarta.persistencia.EntityManagerFactory*
- La configuración de un Entity Manager tiene una plantilla de la *EntityManagerFactory* que lo creó, pero se define por separado como una unidad de persistencia.
- Una ***unidad de persistencia*** dicta implícita o explícitamente la configuración y las clases de entidad utilizadas por todos los Entity Manager obtenidos de la instancia única de EntityManagerFactory vinculada a esa unidad de persistencia.
- Por lo tanto, existe una correspondencia uno a uno entre una unidad de persistencia y su instancia concreta de EntityManagerFactory.

Entity Manager



Contexto de persistencia

- También llamado sesión, es un contenedor de instancias de entidades que están sincronizadas con la base de datos y que poseen un *ciclo de vida* bien definido.

- Los cambios en estos objetos se vuelcan en las tablas con sentencias SQL generadas automáticamente, y solo hay un objeto (*entidad*) por cada registro de la base de datos que se haya importado o creado en el contexto.

- Los contextos se construyen partiendo de la configuración de una unidad de persistencia declarada en el *persistence.xml*

Contexto de persistencia

- Dentro de un contexto de persistencia, los objetos que representan a las entidades se reutilizan para evitar llamadas innecesarias a la base de datos y facilitar el seguimiento de los cambios.

- Una entidad siempre está representada en un contexto por el mismo objeto.

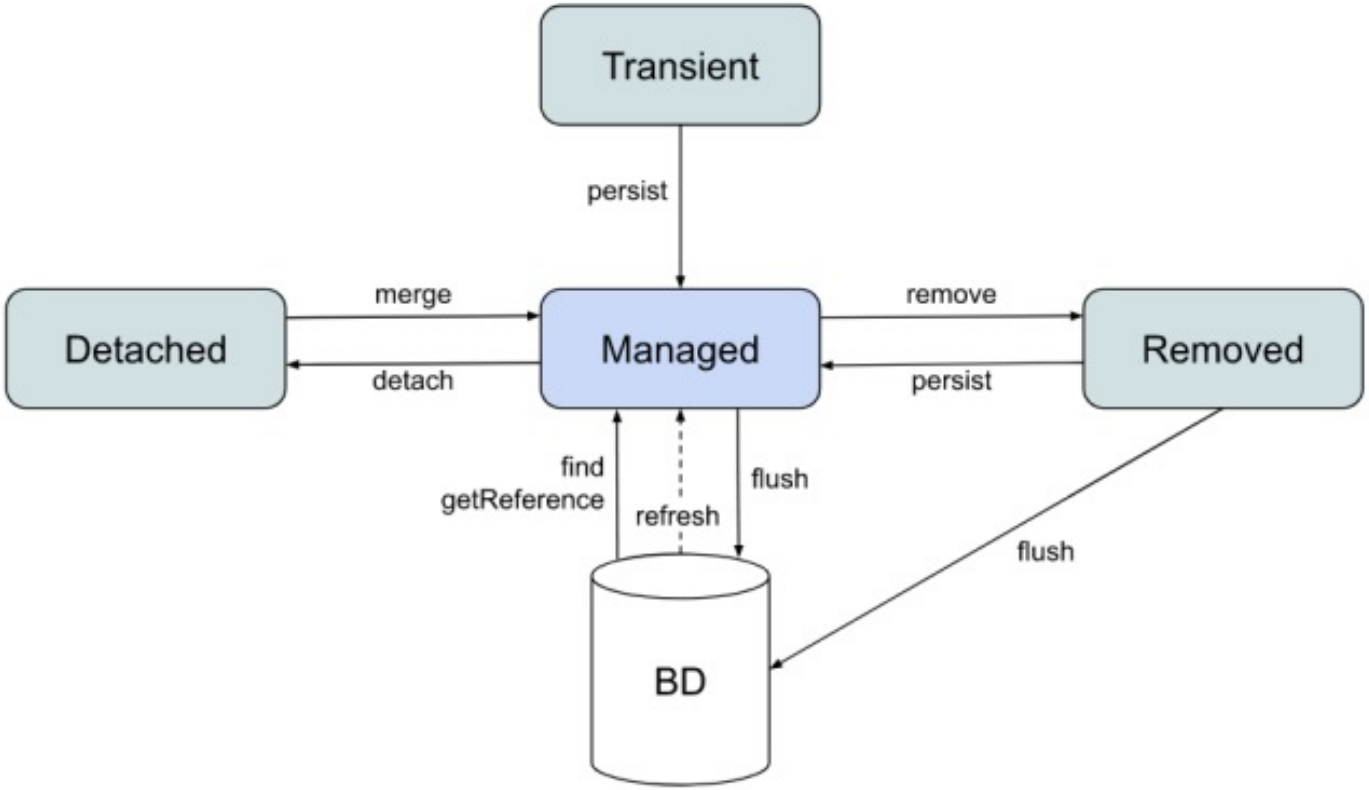
- Esto se consigue con una caché de entidades, llamada «**caché de primer nivel**

- Cada contexto tiene su propia caché de primer nivel de uso exclusivo que garantiza el aislamiento entre contextos \ sesiones \ transacciones.

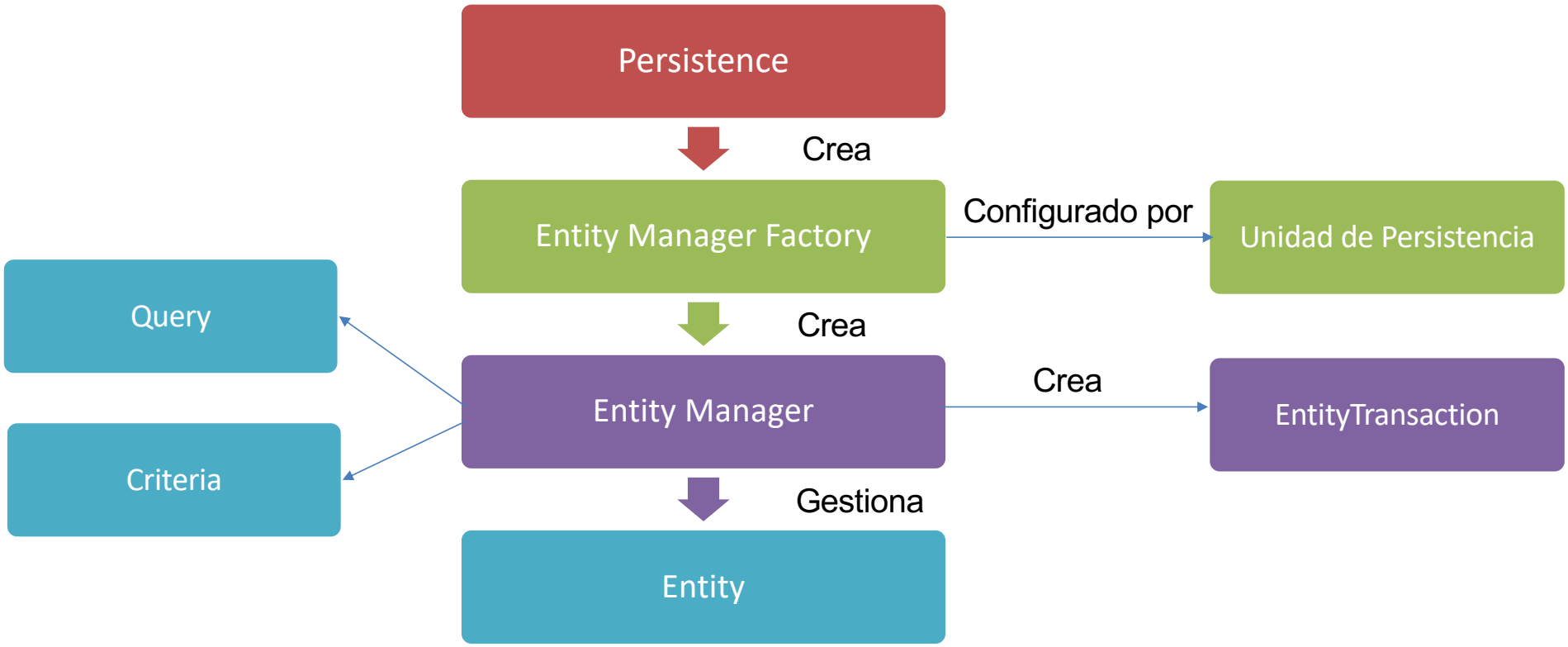
Ciclo de vida

- Las instancias de entidades pueden tener cuatro estados desde el punto de vista del contexto de persistencia.
  - *Nuevo (Transient)*. Se ha instanciado una entidad nueva. Nunca ha formado parte de un contexto y no tiene vinculación alguna con la base de datos.
  - *Persistente\gestionado (Persistent, Managed)*. La entidad forma parte del contexto y es gestionada por el mismo, sus cambios se propagarán a la base de datos bajo ciertas circunstancias.
  - *Desligado (Detached)*. La entidad ya no está en el contexto. En consecuencia, sus cambios no se sincronizarán con la base de datos. Todas las entidades quedan desligadas cuando se cierra el contexto.
  - *Eliminada (Removed)*. Se trata de una entidad que existía en el contexto, pero que ha sido designada para ser borrada.

Ciclo de vida



Jakarta Persitence



Unidades de persistencia

- Cada unidad de persistencia debe tener un nombre.
  - Los nombres de las unidades de persistencia deben ser únicos dentro de su ámbito.

- Las unidades de persistencia se nombran para permitir la diferenciación de una *EntityManagerFactory* de otra.

- Esto le da a la aplicación control sobre qué configuración o unidad de persistencia se utilizará para operar en una entidad en particular.

- Define el conjunto de todas las clases que están relacionadas o agrupadas por la aplicación y que deben asignarse a una sola base de datos.

Unidades de persistencia

- Una unidad de persistencia es una agrupación lógica que incluye:
  - Una EntityManagerFactory y sus EntityManagers, junto con su información de configuración.
  - El conjunto de clases gestionadas incluidas en la unidad de persistencia y gestionadas por los EntityManagers de la EntityManagerFactory.
  - Mapeo de metadatos (en forma de anotaciones de metadatos y/o metadatos XML) que especifica el mapeo de las clases a la base de datos.

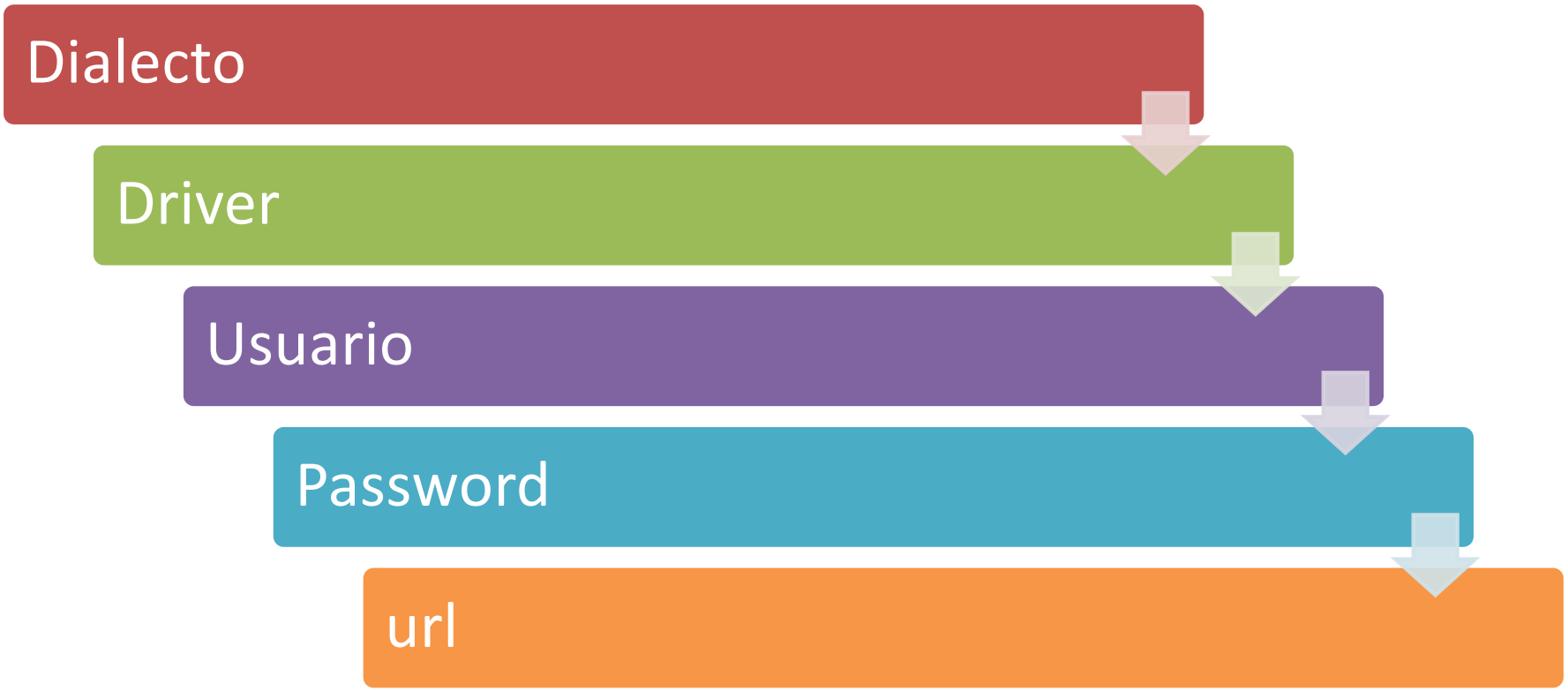
Unidades de persistencia

Objeto	Objeto API	Descripción
Persistence	Persistence	Clase Bootstrap utilizada para obtener un entity manager factory
Entity Manager Factory	EntityManagerFactory	Objeto configurado utilizado para obtener entity managers
Persistence Unit		Configuración nombrada que declara las clases de entidad y la información del almacén de datos
Entity Manager	EntityManager	Objeto API principal utilizado para realizar operaciones y consultas en entidades
Persistence Context		Conjunto de todas las instancias de entidad administradas por un administrador de entidad específico

persistence.xml

- Un documento XML que define una o más unidades de persistencia.
- esquema XML:
  - *https://jakarta.ee/xml/ns/persistence*
    https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd
- El archivo debe colocarse en el directorio META-INF.
- El archivo o directorio JAR cuyo directorio META-INF contiene el archivo persistence.xml se denomina raíz de la unidad de persistencia.

Persistence.xml



Persistence.xml

EntityManagerFactory

- La interfaz *jakarta.persistence.EntityManagerFactory* se proporciona para obtener objetos EntityManager para una unidad de persistencia con nombre.

Obtener un Entity Manager

- El método estático *createEntityManagerFactory()* en la clase **Persistence** devuelve un EntityManagerFactory para el nombre de unidad de persistencia especificado.

- El nombre de la unidad de persistencia especificada *colegio* identifica la configuración que determina los parámetros de conexión que los *EntityManager* generarán desde esta Factory que usarán al conectarse a la base de datos.

- Una vez que se tienen un *EntityManagerFactory*, se puede obtener fácilmente un *EntityManager* de ella.

Obtener un Entity Manager

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("colegio");

EntityManager em = emf.createEntityManager();
```

Persistir entidad

- Persistir una entidad es la operación de tomar una entidad transitoria, o una que aún no tiene ninguna representación persistente en la base de datos, y almacenar su estado para que pueda recuperarse más tarde.

- Esta es realmente la base de la persistencia: crear un estado que pueda sobrevivir al proceso que lo creó.

```
Alumno alumno = new Alumno(matricula);
em.persist(alumno);
```

Persistir entidad

- Llamar al método ***persist()*** es todo lo que se requiere para conservar el estado de la entidad en la base de datos.

- Si el *EntityManager* encuentra un problema al hacer esto, lanzará una excepción sin marcar de tipo *PersistenceException*.

- Cuando se complete la llamada a *persist()*, alumno se habrá convertido en una entidad administrada dentro del contexto de persistencia del administrador de la entidad.

Localizar una entidad

- Una vez que una entidad está en la BD, lo siguiente que normalmente se quiere hacer es encontrarla nuevamente.

```
em.find(Alumno.class, matricula);
```

- Se pasa la clase de la entidad que se busca y la clave principal que identifica la entidad en particular

- Esto es toda la información que necesita el EntityManager para encontrar la instancia en la BD, y cuando se completa la llamada, el alumno será una entidad administrada, lo que significa que existirá en el contexto de persistencia actual asociado con el EntityManager.

Localizar una entidad

- Pasar la clase como un parámetro también permite parametrizar el método de búsqueda y devolver un objeto del mismo tipo que se pasó, ahorrando una conversión adicional.

- En el caso de que no se haya encontrado el objeto, la llamada al método ***find()*** simplemente devuelve *null*.

Eliminar una entidad

- La eliminación de una entidad de la bd no es tan común como podría pensarse.
- Muchas aplicaciones nunca eliminan objetos, o si lo hacen, simplemente marcan los datos como obsoletos o ya no son válidos y luego simplemente los mantienen fuera de la vista de los clientes.
- Se requiere de algo que da como resultado que se realice una instrucción DELETE en una o más tablas.
- Para eliminar una entidad, la propia entidad debe gestionarse, lo que significa que está presente en el contexto de persistencia. La aplicación que llama ya debería haber cargado o accedido a la entidad y ahora está emitiendo un comando para eliminarla.
- Normalmente, esto no es un problema dado que, en la mayoría de los casos, la aplicación habrá provocado que se administre como parte del proceso para determinar que este era el objeto que deseaba eliminar.

Eliminar una entidad

```
Alumno alumno = findAlumno(matricula);
if (alumno != null) {
     em.remove(alumno);
}
```

- Primero se busca la entidad usando el método ***find()***, que devuelve una instancia administrada de Alumno, y luego se elimina la entidad usando la llamada ***remove()*** en el Eentity Manager.

- Se debería verificar la existencia del alumno antes de ejecutar la llamada ***remove()***

Actualizar una entidad

- Existen formas diferentes de actualizar una entidad

- Si no se tiene una referencia a la entidad administrada, primero se debe obtenerla usando ***find()*** y luego realizar las operaciones de modificación en la entidad administrada.

```
Alumno alumno = em.find(Alumno.class, matricula);
if (alumno != null) {
     alumno.setEstatura(alumno.getEstatura() + raise);
}
```

Actualizar una entidad

- Note la diferencia entre esta operación y las demás.

- En este caso, se llama al entity manager para modificar el objeto, sino se llama directamente al objeto mismo.

- Por esta razón, es importante que la entidad sea una instancia administrada; de lo contrario, el proveedor de persistencia no tendrá forma de detectar el cambio y no se realizarán cambios en la representación persistente del alumno.

Mapeo de Entidades

- Si el nombre de la tabla por defecto no es el nombre de la clase, o si ya existe una tabla que contiene el estado en su base de datos con un nombre diferente, se debe especificar el nombre de la tabla.

- Debe usarse la anotación @*Table* e incluir el nombre de la tabla mediante el elemento de nombre.

```
@Entity
@Table(name="EMP")
public class Employee { ... }
```

Mapeo de Entidades

- La anotación @*Table* brinda la capacidad no solo de nombrar la tabla en la que se almacena el estado de la entidad, sino también de nombrar un esquema o catálogo de base de datos.

```
@Entity @Table(name="EMP", schema="HR")
public class Employee { ... }
```

Metadatos de la entidad

- Además de su estado persistente, cada entidad de persistencia tiene algunos *metadatos* asociados que la describen.
- Pueden existir como *parte del archivo de clase*, o pueden almacenarse fuera de la clase, pero no se conservan en la base de datos.
- Permite que la capa de persistencia *reconozca, interprete y administre* correctamente la entidad desde el momento en que se carga hasta su invocación en tiempo de ejecución.
- Los metadatos de la entidad se pueden especificar de dos maneras
  - **Anotaciones**
  - **XML**

Campos y Propiedades

- El estado persistente de una entidad se representa mediante variables de instancia.

- El estado de la entidad está disponible para los clientes solo a través de los métodos de la entidad, es decir, métodos de acceso (métodos getter/setter) u otros métodos de negocio.

- El estado persistente de una entidad es accesido en tiempo de ejecución por el proveedor de persistencia a través de accesores de propiedades de estilo JavaBeans ("acceso a propiedades") o mediante variables de instancia ("acceso a campos").

Campos y Propiedades

- Las variables de instancia de una clase deben ser privadas, protegidas o de visibilidad de paquete, independientemente de si se utiliza acceso de campo o acceso de propiedad.

- Cuando se utiliza el acceso a la propiedad, los métodos de acceso a la propiedad deben ser públicos o protegidos.

- Los campos persistentes con valor de colección y las propiedades deben ser del tipo java.util.Collection, java.util.List, java.util.Map o java.util.Set.
  - La aplicación puede utilizar cualquier tipo de implementación de colección para inicializar campos o propiedades antes de que la entidad se vuelva persistente.

Campos y Propiedades

- Los campos persistentes de una entidad pueden ser de los siguientes tipos:
  - Cualquier tipo primitivo
  - java.lang.String
  - Otros tipos que implementan la interfaz java.io.Serializable (tipos de contenedor primitivo, java.math.BigInteger, java.math.BigDecimal, java.util.Date, java.util.Calendar, java.sql.Date, java.sql .Hora, java.sql.Timestamp, byte[], Byte[], char[], Carácter[], java.time.LocalDate, java.time.LocalTime, java.time.LocalDateTime, java.time.OffsetTime, java .time.OffsetDateTime)
  - Tipos definidos por el usuario que implementan la interfaz java.io.Serializable
  - Enums
  - Entity types
  - Collections of entity types
  - Embeddable classes
  - Collections of basic and embeddable types

Mapeo de Columnas

- Se pueden especificar varios elementos de anotación como parte de *@Column*, pero la mayoría de ellos se aplican solo a la generación de esquemas.

- El elemento de imporancia es de ***name***, que es solo una cadena que especifica el nombre de la columna a la que se ha asignado el atributo.

- Se usa cuando el nombre de columna predeterminado no es apropiado o no se aplica al esquema que se está usando.

Mapeo de Columnas

```
@Entity public class Employee {
    @Id
    @Column(name="EMP_ID")
    private long id;
    private String name;
    @Column(name="SAL")
    private long salary;
    @Column(name="COMM")
    private String comments; // ...
}
```

Field access

- Si se aplica la anotación @Id un atributo, se usa el modo de acceso por atributo (*field access*) y la implementación de JPA debe ser capaz de acceder a los campos sin pasar por los *getters* y *setters*.

- Todos los campos deben declararse como protected, package o private.

- Las propiedades de la clase se vuelven persistentes de forma predeterminada y se asignan a columnas con el mismo nombre.

Field access

```
@Entity public class Employee {
    @Id private long id;
    private String name;
    private long salary;
    public long getId() { return id; }
    public void setId(long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public long getSalary() { return salary; }
    public void setSalary(long salary) { this.salary = salary; }
}
```

Property access

- El modo de acceso propiedad (property access) se activa si @Id se usa en un método *getter*
- Deben existir métodos getter y setter para las propiedades persistentes.
- El tipo de propiedad está determinado por el tipo de retorno del método getter y debe ser el mismo que el tipo del único parámetro pasado al método setter.
- Ambos métodos deben tener visibilidad pública o protegida.
- Las anotaciones de asignación para una propiedad deben estar en el método getter.

Property access

```
@Entity public class Employee {
    private long id;
    private String name;
    private long wage;
    @Id public long getId() { return id; }
    public void setId(long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public long getSalary() { return wage; }
    public void setSalary(long salary) { this.wage = salary; }
}
```

Mixed Access

- La anotación ***@Access*** es útil cuando necesita realizar una transformación simple en los datos al leer o escribir en la base de datos.

- En este caso se definirán un par de métodos getter/setter para realizar la transformación y usar el acceso de propiedad para ese atributo.

```
@Entity
@Access(AccessType.FIELD)
public class Employee { ... }
```

Mixed Access

- El siguiente paso es anotar el campo o propiedad adicional con la anotación @Access, pero esta vez especificando el tipo de acceso opuesto al especificado en el nivel de clase.

- Puede parecer un poco redundante, por ejemplo, especificar el tipo de acceso de AccessType.PROPERTY en una propiedad persistente porque es obvio que se trata de una propiedad, pero al hacerlo indica que lo que está haciendo no es un descuido.

```
@Access(AccessType.PROPERTY) @Column(name="PHONE")
protected String getPhoneNumberForDb() { ... }
```

Tipos enumerados

- Los valores de un tipo enumerado son ***constantes*** que se pueden manejar de manera diferente según las necesidades de la aplicación.

- Tienen una ***asignación ordinal*** implícita que está determinada por el orden en que fueron declarados, ***no se puede modificar en tiempo de ejecución*** y se puede usar para representar y almacenar los valores del tipo enumerado en la base de datos.

- Con la anotación ***@Enumerated*** se puede utilizar como criterio alternativo el nombre. Resulta más legible, pero menos eficiente porque requiere de una columna de mayor tamaño: con la posición solo se almacena un numérico con uno o dos dígitos.

Tipos enumerados

```
public enum EmployeeType {
    FULL_TIME_EMPLOYEE, // -> 0
    PART_TIME_EMPLOYEE, // -> 1
    CONTRACT_EMPLOYEE // -> 2
}

@Entity public class Employee {
    @Id private long id;
    private EmployeeType type; // ...
}
```

Tipos Temporales

- Son el conjunto de tipos basados en el tiempo que se pueden usar en asignaciones de estado persistentes.

- La lista de tipos temporales admitidos incluye los tres tipos de
  - java.sql
    - java.sql.Date
    - java.sql.Time
    - java.sql.Timestamp
  - java.util
    - java.util.Date
    - java.util .Calendario.

Tipos Temporales

- Los tipos *java.sql* no necesitan tratamiento adicional y actúan como cualquier otro tipo de mapeo simple

- Los tipos *java.util* necesitan metadatos adicionales para indicar cuál de los tipos java.sql de JDBC se debe utilizar al comunicarse con el controlador JDBC.

- Se una la anotación ***@Temporal*** y se especificando el tipo JDBC como un valor del tipo enumerado TemporalType.
  - DATE
  - TIME
  - TIMESTAMP

Tipos Temporales

```
@Entity public class Employee {
    @Id private long id;
    @Temporal(TemporalType.DATE)
    private Calendar dob;
    @Temporal(TemporalType.DATE)
    @Column(name="S_DATE")
    private Date startDate;
    // ...
}
```

Mixed Access

- El campo o propiedad correspondiente al que se está haciendo persistente debe marcarse como *transitorio* para que las reglas de acceso predeterminadas no provoquen que el mismo estado persista dos veces.

- El campo en el que se almacena el estado de la propiedad persistente en la entidad debe anotarse con ***@Transient***

@Transient private String phoneNum;

Estado transitorio

- Los atributos que forman parte de una entidad persistente pero que no pretenden ser persistentes pueden modificarse con el modificador transitorio en Java o anotarse con la anotación @*Transient*.

- Si se especifica alguno, el proveedor no aplicará las reglas de asignación predeterminadas al atributo en el que se especificó.

Estado transitorio

```
@Entity
public class Employee {
    @Id
    private int id;
    private String name;
    private long salary;
    @Transient
    private String convertedName;
}
```

Contacto

Dr. Omar Mendoza González

omarmendoza564@aragon.unam.mx