UDE
UNIVERSIDAD DE LA EMPRESA

Bases de Datos 3

- Docentes: Federico Gómez, Diego Siri
- Carreras: Licenciatura & Ingeniería en Informática
- Año de la carrera: 3º

UDE
UNIVERSIDAD DE LA EMPRESA

Capítulo 6:
Introducción a los Frameworks de Persistencia

UNIVERSIDAD DE LA EMPRESA

Introducción:

En este capítulo realizaremos una introducción al concepto de ***frameworks de persistencia*** como herramientas de soporte al ***manejo de persistencia*** en el desarrollo de aplicaciones.

Conforme las aplicaciones han ido creciendo en complejidad y en tamaño, ha ido surgiendo la necesidad de crear herramientas que ***faciliten*** al desarrollador el manejo de la persistencia de los datos.

Durante el curso, hemos visto que es el desarrollador quien debe escribir el código fuente necesario para acceder al mecanismo de persistencia utilizado (pudiendo hacerlo en ***1, 2 o 3 capas***).

Los frameworks de persistencia procuran ***quitar*** al desarrollador la responsabilidad de escribir el código fuente de acceso al medio de persistencia, ***delegando*** dicha responsabilidad al framework.

UNIVERSIDAD DE LA EMPRESA

¿Qué es un Framework?

Un ***framework*** se define, a nivel conceptual, como una ***plataforma***, ***entorno*** o ***marco*** de referencia.

En el contexto del ***desarrollo de aplicaciones***, un framework es una estructura de soporte en la cual otro proyecto de software puede ser organizado y desarrollado. Dicha estructura suele incluir los siguientes elementos:

- Bibliotecas predefinidas que resuelven operaciones relevantes, quitando al desarrollador la responsabilidad de implementarlas.
- Facilidades para el desarrollo de nuevas piezas de código fuente (clases, si el lenguaje es OO) tales como plugins, asistentes, etc.

La idea (informal) de un framework es la siguiente:

"*Programar lo importante, el framework resuelve lo tedioso*"

UDE
UNIVERSIDAD DE LA EMPRESA

¿Qué es un Framework de Persistencia?

Un ***framework de persistencia*** es un tipo especial de ***framework*** que se especializa en la gestión (semi) automatizada del acceso al mecanismo concreto de persistencia utilizado por una aplicación.

Por tratarse de ***frameworks***, poseen los elementos enumerados anteriormente. Por ser ***de persistencia***, hacen hincapié en los siguientes aspectos:

- Las bibliotecas predefinidas se encargan de implementar las operaciones necesarias para acceder al ***medio de persistencia*** que el framework utiliza.
- El desarrollador hace uso de las operaciones anteriores ***sin*** necesitar conocer cómo es el código fuente que el framework usa internamente para acceder al medio de persistencia.

UNIVERSIDAD DE LA EMPRESA

¿Qué tipos de Framework de Persistencia existen?

En principio, sería posible utilizar ***frameworks de persistencia*** que usen ***cualquier*** mecanismo de almacenamiento (BD relacionales, archivos planos, archivos binarios, archivos XML, etc.)

No obstante, las ***bases de datos relacionales*** siguen siendo el medio de persistencia más utilizado. Por lo tanto, la mayoría de los ***frameworks de persistencia*** existentes utilizan BD relacionales como medio de almacenamiento.

En resumen, podemos hacer la siguiente clasificación:

- ***Frameworks***
  - ***Frameworks de persistencia***
    - ***Frameworks de persistencia sobre BD relacionales***

      (éstos últimos son los que nos interesan)

UNIVERSIDAD DE LA EMPRESA

Frameworks de Persistencia sobre BD relacionales:

Como su nombre lo indica, los ***frameworks de persistencia sobre BD relacionales*** persisten la información en BD relacionales.

Internamente, se encargan de implementar las sentencias ***SQL*** que acceden a la(s) base(s) de datos utilizada(s) por la aplicación. El programador ***no necesita*** escribir sentencias SQL. Lo que hace es utilizar las bibliotecas provistas por el framework.

Lo anterior ***facilita*** el desarrollo de software. El programador se concentra mayormente en escribir el código fuente a nivel de la ***aplicación***, dejando al framework la responsabilidad del código fuente de acceso al ***medio de persistencia***.

Los frameworks de persistencia sobre BD relacionales usualmente están concebidos para el desarrollo de aplicaciones en ***3 capas***.

UDE
UNIVERSIDAD DE LA EMPRESA

ORM (Object-Relational Mapping):

Es un conjunto de técnicas para realizar el ***mapeo*** (traducción) entre el ***modelo relacional de BD*** y el ***modelo OO***. Las técnicas que lo componen fueron presentadas en el capítulo 5.

Como se vio en dicho capítulo, el ***mapeo OO – BD*** es un problema ***abierto*** que ***no*** posee una solución ideal hasta la fecha.

Los ***frameworks de persistencia sobre BD relacionales*** utilizan las técnicas anteriores en forma ***interna***. Según qué framework se utilice, las técnicas pueden ser ***automáticas*** o bien el desarrollador puede indicarle cuál(es) desea utilizar en cada relación.

La gran mayoría de los frameworks de persistencia hacen uso de las técnicas ***ORM***. Por tal razón, se los conoce como ***frameworks de persistencia ORM***, o simplemente ***frameworks ORM***.

UNIVERSIDAD DE LA EMPRESA

Características de los Frameworks ORM:

Los frameworks ORM poseen las siguientes características:

- A partir de las clases de la capa ***lógica*** y de las relaciones entre ellas, generan automáticamente el esquema de tablas de la BD.
- Según el framework que se utilice, el ***mapeo*** de las relaciones puede ser automático o asistido (esto es, se puede "forzar" una determinada representación de una relación).
- Las ***claves*** de las tablas de la BD son definidas en base a las indicaciones del programador.
- Proveen facilidades para ***manejo de transacciones*** (en algunos casos, incluso pueden traer predefinido un ***pool de conexiones***, sobre todo cuando se usan en contextos de aplicaciones web).

UDE
UNIVERSIDAD DE LA EMPRESA

Ventajas de los Frameworks ORM:

Los frameworks ORM poseen las siguientes ventajas:

1) El desarrollador solamente se concentra en tareas inherentes al ***diseño OO***, dejando al framework el ***diseño de la BD***.

Por ejemplo, ***sin*** frameworks ORM, el desarrollador debe realizar:

- Diagrama de clases ***conceptual*** en UML.
- Diagrama de clases de ***implementación*** en UML.
- Modelo entidad-relación (***MER***).
- Normalización y diseño de tablas de la BD.

En cambio, ***con*** frameworks ORM, el desarrollador debe realizar:

- Diagrama de clases ***conceptual*** en UML.
- Diagrama de clases de ***implementación*** en UML.
- Indicar como realizar (algunos de) los mapeos OO – BD.

UDE
UNIVERSIDAD DE LA EMPRESA

Ventajas de los Frameworks ORM (continuación):

Los frameworks ORM poseen las siguientes ventajas:

2) El desarrollador ***no*** necesita escribir sentencias ***SQL*** (incluso podría no necesitar saber escribir SQL). Las mismas son generadas internamente por el framework.

3) El desarrollador ***no*** necesita manejar directamente ***Conexiones*** ***PreparedStatements***, ***ResultSets***, ***Transacciones***, etc.

4) Cuando ***no*** se utilizan frameworks, el código es repetitivo (se hace *INSERT*, *SELECT*, *UPDATE*, etc. para cada clase ***DAO*** del modelo. Al incorporar el uso del framework, el desarrollador no repite sentencias SQL (ni siquiera las escribe).

5) ***Reducen*** el tiempo de desarrollo de las aplicaciones.

UNIVERSIDAD DE LA EMPRESA

Desventajas de los Frameworks ORM:

Los frameworks ORM poseen las siguientes desventajas:

1) Las sentencias ***SQL*** generadas internamente por el framework pueden resultar en una ejecución lenta e ineficiente. Suelen ser menos óptimas que aquellas expresamente escritas por los desarrolladores.

2) El hecho de que se está persistiendo en una **BD *relacional*** termina viéndose reflejado en la capa ***lógica*** de un modo u otro, perjudicando levemente la separación en capas.

3) Dependiendo del framework, el programador puede necesitar escribir sentencias usando un lenguaje de consulta propio del framework (que luego es internamente traducido a SQL). O sea, termina siendo necesario manejar algún lenguaje de consultas.

UNIVERSIDAD DE LA EMPRESA

Evolución de los Frameworks ORM:

Los frameworks ORM fueron surgiendo con el fin de simplificar el manejo de la capa de ***persistencia*** e incrementar la ***productividad*** en el desarrollo de aplicaciones.

Se han desarrollado múltiples frameworks de persistencia (en breve nombraremos algunos), unos más exitosos o eficaces que otros. Inicialmente, cada framework definía sus propias estrategias para realizar el ***mapeo OO – BD***.

Con el tiempo, algunos mecanismos se han ido estandarizando y han ido surgiendo diversas ***API***'s que buscan uniformizar la manera en que se realizan los mapeos. No obstante, a la fecha no existe ningún estándar globalmente aplicado. Es un tema que continúa en pleno desarrollo e investigación.

UNIVERSIDAD DE LA EMPRESA

JPA – Java Persistence API:

JPA es una API utilizada para realizar mapeos ORM utilizando el lenguaje de programación ***Java***.

Esta API busca unificar y estandarizar la manera en que funcionan los mapeos ORM. En su creación, se combinaron ideas usadas por distintos frameworks ORM en forma previa al surgimiento de JPA.

Actualmente hay diversos frameworks que siguen la especificación dada por JPA, como ser: *Hibernate*, *Toplink*, *Glassfish*, ***EclipseLink*** (éste último es el que usaremos en el práctico, ver el instructivo correspondiente en el sitio web del curso).

De modo análogo a **JDBC**, **JPA** define una serie de interfaces que los distintos fabricantes de frameworks luego implementan y que podemos utilizar en nuestras aplicaciones.

UDE
UNIVERSIDAD DE LA EMPRESA

JPA – Java Persistence API (continuación):

La versión inicial de JPA (1.0) fue liberada en Mayo de 2006. La versión actual es la 2.1 y fue liberada en Abril de 2013. Su principal propósito fue unificar características presentes en algunos frameworks populares, pero aún no consensuadas en la 1.0.

En JPA, los mapeos ORM se realizan mediante ***anotaciones*** en las clases de la capa ***lógica*** que se desea persistir. Las anotaciones son ***etiquetas*** que se añaden al código Java para ser tomadas en cuenta en tiempo de ***ejecución***.

El uso de anotaciones es lo que libera al programador de tener que codificar explícitamente las sentencias SQL de acceso a la BD. En contraparte, su uso es a su vez lo que, tarde o temprano, termina haciendo visible en el código el hecho de que se persiste en **BD**.

JPA – Java Persistence API (continuación):

Presentamos a continuación algunos ejemplos **aislados** de JPA, a efectos de entender el concepto de uso de anotaciones:

```
@Entity
public class Pelicula
{
   @id
   @GeneratedValue
   private Long id;
   private String titulo;
   private int duracion;
   ...
}
```

UNIVERSIDAD DE LA EMPRESA

JPA – Java Persistence API (continuación):

En este primer ejemplo destacamos los siguientes aspectos:

- La anotación **@Entity** indica al framework que todo objeto de la clase **Película** será ***persistido***. Esto ocasiona que el framework genere una tabla en la BD que los almacene.
- Por defecto, el framework generará una tabla con el mismo nombre de la clase (**Película**) para almacenar los objetos.
- La anotación **@id** indica al framework que el atributo **id** será la clave primaria que identifique a cada objeto. Existen también formas más complejas para definición de claves.
- La anotación **@GeneratedValue** indica al framework que el valor del atributo **id** será autogenerado.

JPA – Java Persistence API (continuación):

Veamos un segundo ejemplo de JPA:

```
@Entity
@Table(name = “TABLA_PELICULAS”)
public class Pelicula
{
   @id
   @GeneratedValue
   @Column(name = “ID_PELICULA”)
   private Long id;
   private String titulo;
   private int duracion;
   ...
}
```

UNIVERSIDAD DE LA EMPRESA

JPA – Java Persistence API (continuación):

En este segundo ejemplo destacamos los siguientes aspectos:

- La anotación **@Table(name = “TABLA_PELICULAS”)** permite definir expresamente el nombre de la tabla en donde se desea almacenar a las películas.
- La anotación **@Column(name=“ID_PELICULA”)** permite definir expresamente el nombre de la columna donde se almacena un determinado atributo.
- Si no se especifican expresamente, el framework normalmente utilizará los nombres de la clase y de los atributos como nombres de tablas y de columnas.

JPA – Java Persistence API (continuación):

Veamos un tercer ejemplo de JPA:

```
@Entity
public class Dueño
{
    @id
    @GeneratedValue
    private Long id;

    @OneToMany
    private List<Mascota> mascotas;
    ...
}
```

UNIVERSIDAD DE LA EMPRESA

JPA – Java Persistence API (continuación):

En este tercer ejemplo destacamos los siguientes aspectos:

- La anotación **@OneToMany** fuerza una cierta representación para una relación de **1** a **0..***. Indica que cada dueño debe tener una lista conteniendo a todas sus mascotas.
- Por defecto, JPA define el siguiente mapeo en la BD:

  Dueño (idDueño, camposDueño)
  Mascota (idMascota, camposMascota)
  Dueño-Mascota (idDueño, idMascota)

- Sin embargo, del capítulo 5 sabemos que un mejor mapeo (en términos del modelo OO) para esta relación sería el siguiente:

  Dueño (idDueño, camposDueño)
  Mascota (idMascota, camposMascota, idDueño)

JPA – Java Persistence API (continuación):

- La elección de un buen mapeo (en términos del modelo OO) implica una decisión de ***diseño***, por lo que los mapeos generados por defecto no siempre son adecuados.
- JPA permite definir expresamente el segundo mapeo mediante la siguiente anotación:

```
@OneToMany
@JoinColumn(name = “idDueño”)
private List<Mascota> mascotas;
```

- La anotación anterior agrega el identificador del dueño como clave foránea en la tabla de Mascotas, forzando de este modo la representación deseada.

JPA – Java Persistence API (continuación):

Veamos un cuarto ejemplo de JPA:

```
@Entity
@NamedQueries
 (@NamedQuery (name = “personasPorApellido”,
               query = “SELECT p FROM Persona p
                        WHERE p.ced = :ced”))
public class Persona
{
   @id
   @GeneratedValue
   private Integer id;
   ...
}
```

UNIVERSIDAD DE LA EMPRESA

JPA – Java Persistence API (continuación):

En este cuarto ejemplo destacamos los siguientes aspectos:

- La anotación **@NamedQueries** permite definir un conjunto de consultas usando **JPQL**, el lenguaje de Consultas de JPA, que es similar a SQL, pero manipula ***clases*** en lugar de ***tablas***.
- La anotación **@NamedQuery** permite definir una consulta en **JPQL** que luego puede ser utilizada desde aquellas clases que vayan a persistir los datos de las personas.
- En JPQL es posible definir consultas ***con*** parámetros como ***sin*** parámetros, de modo similar a JDBC.

UDE
UNIVERSIDAD DE LA EMPRESA

JPA – Alternativa al uso de anotaciones:

En todos los ejemplos anteriores, el uso de ***anotaciones*** refleja en el código de las clases de la capa ***lógica*** aspectos expresamente propios de la capa de ***persistencia***. Esto perjudica la transparencia en términos de la ***separación en capas***.

Una alternativa al uso de anotaciones es la definición de ***archivos XML*** en lugar de anotaciones. Todo lo relativo a los mapeos es definido en dichos archivos, liberando así al código fuente de la presencia de anotaciones que opacan la separación en capas.

De todos modos, aspectos vinculados al ***diseño*** de los mapeos necesarios o el uso de **JPQL** en la resolución de requerimientos eventualmente terminan condicionando el ***diseño OO*** al hecho de que se está persistiendo en una ***BD relacional***.

UNIVERSIDAD DE LA EMPRESA

JPA – Persistiendo objetos a través del framework:

Una vez definidas las clases cuyos objetos se desea persistir (mediante la anotación **`@Entity`**), para hacerlo es necesario usar las interfaces **`EntityManagerFactory`** y **`EntityManager`**.

- La interface **`EntityManagerFactory`** define el nexo entre la aplicación y el framework de persistencia. Cumple un rol que es equivalente a la interface **`DriverManager`** de **JDBC**.
- La interface **`EntityManager`** permite persistir y recuperar datos del medio de persistencia. Cumple un rol equivalente a las interfaces **`Connection`**, **`Statement`**, **`PreparedStatement`** de **JDBC** (unificadas dentro de **`EntityManager`**).

Cada framework concreto de **JPA** (*Hibernate*, ***EclipseLink***, etc.) provee su propia implementación para estas interfaces, de igual modo que cada driver provee la suya para cada interface de **JDBC**.

UDE
UNIVERSIDAD DE LA EMPRESA

JPA – Persistiendo objetos a través del framework (cont.):

Para establecer la conexión con el DBMS (a través del framework), lo primero es crear una instancia del **`EntityManagerFactory`**:

```
String nom = "pruebaJPA";
EntityManagerFactory factory =
   Persistence.createEntityManagerFactory(nom);
```

JPA usa como único ***archivo de configuración*** un archivo llamado **`Persistence.xml`**, dentro del cual se indican ***driver***, ***url***, ***user*** y ***password*** de la base de datos a utilizar, lo cual se hace bajo un nombre vinculable desde el código (**`pruebaJPA`** en el ejemplo).

Observación: Ver instructivo en el sitio web del curso para más detalle sobre el archivo **`Persistence.xml`**.

UNIVERSIDAD DE LA EMPRESA

JPA – Persistiendo objetos a través del framework (cont.):

Luego, para persistir y/o recuperar objetos persistidos, se le pide un **`EntityManager`** al **`EntityManagerFactory`**. Por ejemplo:

```
EntityManager manager =
   factory.createEntityManager();
...
Pelicula p1 = new Pelicula();
p1.setTitulo("Star Wars VIII: The Last Jedi");
p1.setDuracion(2);
manager.persist(p1);
Pelicula p2 =
  (Pelicula) manager.find(Pelicula.class, 222);
```

Observación: El acceso a la **BD** queda ***encapsulado*** dentro de los métodos del **`EntityManager`**.

UNIVERSIDAD DE LA EMPRESA

JPA – Persistiendo objetos a través del framework (cont.):

También se ejecutan consultas (en particular ***named queries***) a través del **`EntityManager`**. Por ejemplo:

```
Query query = manager.createNamedQuery
      (peliculasPorDuracion, Persona.class);
query.setParameter (“duracion”, 2);
@SupressWarnings (“unchecked”)
List <Pelicula> lista = query.getResultList();
/* ejecuta la consulta y devuelve una lista de
   objetos como resultado, no hay ResultSet */
```

Observación:

Todos los accesos a la BD usando **JPA** se realizan ahora donde antes se realizaban usando **JDBC**. Por ejemplo, dentro de las clases ***DAO*** si la aplicación es en ***3 capas***.

UNIVERSIDAD DE LA EMPRESA

Conclusiones:

Los ***frameworks de persistencia*** (en particular ***frameworks ORM***) incrementan la ***productividad*** en el desarrollo de aplicaciones y procuran quitar al desarrollador la responsabilidad de codificar el código fuente de acceso al ***medio de persistencia***.

No obstante, no llegan a eliminar ***totalmente*** dicha responsabilidad. Sigue siendo necesario definir expresamente aspectos de la capa de persistencia (definir ***mapeos***, codificar sentencias **JPQL**), aún cuando se utilice ***XML*** en lugar de ***anotaciones***.

Al igual que en las ***aplicaciones en 3 capas*** (donde el acceso a la persistencia es ***explícito***), sigue habiendo aspectos del ***diseño OO*** condicionados al hecho de que se persiste en una ***BD relacional*** (representación de ***relaciones*** entre clases, eficiencia en cantidad de ***accesos*** a la BD).