MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Bases de datos orientadas a objetos II

19

ÍNDICE

OBJETIVOS

Realizar programas con acceso a bases de datos orientadas a objetos.

Insertar datos en una base de datos orientada a objetos.

Recuperar información de una base de datos orientada a objetos.

Borrar y modificar datos de una base de datos orientada a objetos.

/ 1. Introducción y contextualización práctica

En esta unidad vamos a continuar con el estudio de las bases de datos orientadas a objetos.

Vamos a conocer otro tipo de consultas, las consultas SODA, las cuales, son más avanzadas que las consultas QBE estudiadas en la unidad anterior, y, por tanto, permitirán realizar consultas más complejas.

Además, vamos a aprender a manejar tipos de datos estructurados dentro de una base de datos orientada a objetos, y cómo vamos a poder realizar consultas sobre ellos.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

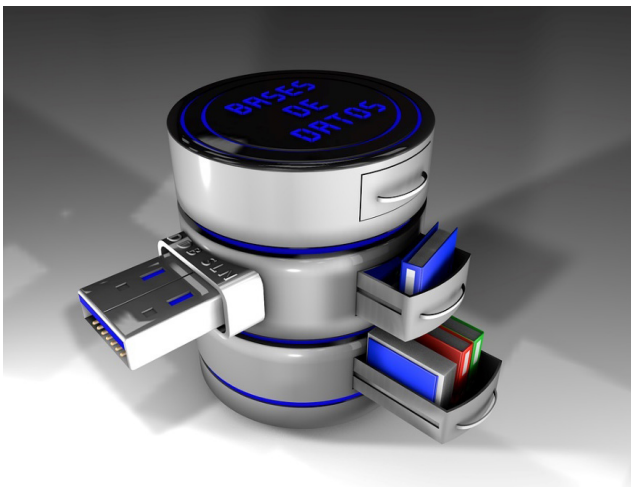

Fig. 1. Diferentes bases de datos.

Audio Intro. "¿QBE es insuficiente?"
http://bit.ly/2XwF5bW

/ 2. Tipos de consultas

En la unidad anterior hemos tenido nuestro primer contacto con las bases de datos orientadas a objetos, comprobando que éstas son utilizadas cuando necesitamos una base de datos embebida en nuestros proyectos.

Una base de **datos embebida** es una base de datos **no dependiente de un Sistema Gestor de Bases de Datos**, es decir, si utilizamos una base de datos SQL con MySQL vamos a necesitar obligatoriamente instalar el Sistema Gestor de Bases de Datos que la gestione, es este caso, podemos instalar una gran variedad, como MySQL Workbench, XAMP, Oracle, etc.

Cuando usamos una base de datos embebida esto no es problema, ya que esta **estará dentro de nuestro proyecto como un pequeño fichero**, es decir, lo que tenemos al utilizar **DB4O**, con el gran inconveniente de que si dicho fichero de base de datos se pierde perdemos la base de datos al completo.

También existen **bases de datos embebidas que pueden utilizar SQL**, como es el caso de **SQLite**, que creará un fichero de base de datos que podremos embeber en nuestro proyecto de una forma sencilla.

Volviendo a las bases de datos DB4O, éstas nos ofrecen la posibilidad de consultar la base de datos orientadas a objetos mediante **tres tipos diferentes de lenguajes de consulta de datos**:

1. ***Query By Example (QBE):*** Este lenguaje de consulta de datos lo hemos estudiado en la unidad anterior.

2. ***Native Queries (NQ):*** Son consultas nativas, y son la interfaz principal y la recomendada por los desarrolladores de DB4O.

3. ***Simple Object Data Access (SODA):*** Esta es la API interna. Se puede utilizar para una mayor retrocompatibilidad o para generar consultas dinámicamente. Es mucho más potente que las dos anteriores y mucho más rápida.

Fig. 2. Diseñando una base de datos.

2.1. QBE vs SODA

Ya sabemos que existen tres tipos diferentes de consultas que podemos lanzar a las bases de datos DB4O. Nosotros nos vamos a centrar en las que ya hemos estudiado o vamos a hacerlo: las consultas en lenguaje QBE, y las consultas en lenguaje SODA.

QBE es la forma más básica de consultar la **base de datos orientada a objetos**. Ésta es sencilla porque funciona presentando un ejemplo y se recuperarán los datos que coincidan con él.

QBE tiene las siguientes **limitaciones**:

- No permite realizar consultas avanzadas con los operadores *AND, OR, NOT...*
- Hay que proporcionarle un ejemplo, y eso tiene muchas limitaciones y algún inconveniente, como, por ejemplo, la pérdida de tiempo que conlleva montar los ejemplos.
- No nos permite preguntar por objetos cuyo valor de un campo numérico sea 0, *String*, vacíos o campos con valor *null*.
- Se necesita de un constructor que construya los objetos no inicializados.

Las consultas **SODA** son una **biblioteca que presenta la ventaja de una ejecución más rápida de las mismas y permite generar consultas dinámicas**. Para crear una consulta SODA es necesario un objeto del tipo *Query*, el cual puede ser generado llamando al método *query*() del objeto de tipo *ObjectContainer*. Una vez creado el *query* le podremos ir añadiendo *Constraints* (restricciones) para ir modelando el resultado que necesitamos obtener.

Una vez que se tiene ya modelada la consulta se ejecuta llamando al método *execute*() del objeto tipo *Query*.

En los siguientes apartados vamos a ver cómo podemos generar consultas con SODA y para ello vamos a necesitar las siguientes nuevas clases:

Clase	Funcionalidad
Constraint	Esta clase nos va a permitir agregar restricciones a las consultas SODA.
Query	Esta clase nos va a permitir crear las consultas SODA.

Tabla 1. Clases necesarias para las consultas SODA.

/ 3. Caso práctico 1: "Migrando consultas"

Planteamiento: Pilar y José han recibido una nueva práctica sobre el proyecto del programa gestor de las barras de bar. En este caso tienen que realizar una serie de consultas SODA que anteriormente no podían realizar con QBE, para así equiparar el programa con bases de datos orientadas a objetos al de bases de datos relacionales.

Nuestros amigos ya tenían anteriormente creadas una serie de consultas QBE muy simples, pero en clase han visto que las consultas que se pueden realizar en QBE se pueden realizar también en SODA. "¿Deberíamos migrar las consultas QBE que ya teníamos hechas a SODA, antes de hacer las nuevas consultas SODA?", le pregunta Pilar a José, mientras que éste le responde que él cree que sí, ya que si no, pudiera haber problemas de incompatibilidad al utilizar dos tipos de consultas.

Nudo: ¿Qué opinas al respecto? ¿Crees que, como dice Pilar, deberían migrar las consultas ya hechas a SODA?, o que como piensa José, ¿deberían dejarlas en QBE?

Desenlace: Cuando usamos una base de datos orientada a objetos embebida como es el caso de DB4O podemos realizar consultas de las dos formas, tanto con QBE como son SODA.

En principio, no tendríamos ninguna incidencia si mantenemos consultas en los dos diferentes lenguajes, aunque lo ideal sería dejarlas en uno solo, en este caso, SODA que es más completo que QBE y cubre todas sus consultas, para así, tener una uniformidad en todo el proyecto.

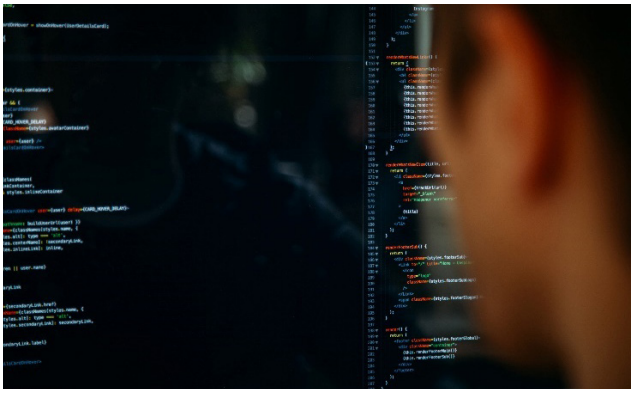

Fig. 3. Consultando la base de datos.

/ 4. Consultas avanzadas

Lo primero que debemos conocer del lenguaje de consultas SODA es cómo recuperar todos los objetos de un tipo concreto de la base de datos.

Para contextualizar lo que vamos a hacer, y entenderlo un poco mejor, basándonos en una analogía con SQL lo que estamos intentando implementar es lo que hacíamos en SQL con *SELECT*:

```
SELECT * FROM nombretabla
```

Solo que, en **lugar de obtener los datos de una tabla, los vamos a obtener de la base de datos orientada a objetos.**

Lo primero que debemos hacer es crear un objeto de tipo *Query*, para poder lanzarle las consultas SODA:

```
Query consulta = bd.query();
```

El método *query*() nos va a devolver un objeto de tipo *Query* al que le podremos lanzar las consultas SODA que necesitemos.

A partir de este momento, lo que tendremos que hacer es ir **agregando restricciones a la consulta**, como si de un *WHERE* en SQL se tratara. La primera restricción que tenemos que agregar siempre va a ser en la que le indicaremos a la consulta, el tipo de objeto que queremos rescatar.

Para ello tendremos que utilizar el **método *constraint*** del objeto de tipo *Query* y pasarle como parámetro el nombre de la clase de los objetos que queremos rescatar seguido de *.class*. Por ejemplo, si queremos rescatar objetos de tipo *Alumno*:

```
consulta.constrain(Alumno.class);
```

Esto nos va a permitir rescatar únicamente objetos de ese tipo, ya que podremos tener más de un tipo de objetos en la base de datos.

Una vez implementado esto, ejecutamos el **método *execute*** y nos devolverá un **objeto de tipo *ObjectSet*** con los objetos rescatados, al igual que con las QBE.

```
ObjectContainer bd = null;

try {
    bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "alumnos.db4o");
    Query consulta = bd.query();
    consulta.constrain(Alumno.class);
    ObjectSet res = consulta.execute();
    while (res.hasNext()) {
        System.out.println(res.next());
    }
} catch (Exception ex) {
    System.out.println("Error: " + ex.toString());
} finally {
    bd.close();
}
```

Fig. 4. Consulta SODA.

Audio 1. "Recordatorio de bases de datos"
http://bit.ly/3oANTcL

4.1. Consultas avanzadas con restricciones

Imaginemos ahora que queremos rescatar de la base de datos todos los **objetos de tipo alumno cuya edad sea superior a 18**, lo cual, en términos de SQL sería hacer:

```
SELECT * FROM alumno WHERE edad > 18
```

Esta consulta ya no sería posible de realizar con QBE, pero son SODA sí. Para ello necesitamos un objeto de tipo *Constraint*, en el que podremos definir todo tipo de restricciones para nuestras consultas.

En los **objetos de tipo *Constraint*** disponemos del **método *descend***, el cual nos permitirá agregar una restricción a un atributo del objeto. Seguidamente con el método *constraint* le podremos indicar el valor de la restricción y según queramos que sea dicha restricción podemos aplicar los siguientes **métodos**:

- ***smaller():*** Este método nos permitirá crear una restricción para valores más pequeños.
- ***greater():*** Este método nos permitirá crear una restricción para valores más grandes.
- ***equal():*** Este método nos permitirá crear una restricción para valores iguales.
- ***not():*** Este método nos permitirá crear una restricción para valores distintos al indicado.

Aquí hay que tener mucho cuidado, ya que el nombre del atributo al que le queremos aplicar la restricción deberá llamarse exactamente igual a como se haya llamado en la clase, es decir, si queremos aplicar una restricción a la edad de un objeto de tipo *Alumno*, tendremos que comprobar en la clase **Alumno** cómo se llama el atributo que representa la edad, y transcribirlo exactamente igual en el método *descend*.

Para obtener los objetos de tipo *alumno* que tengan una edad mayor de 18 años podemos escribir el código de la siguiente figura:

```
bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "alumnos.db4o");
Query consulta = bd.query();

consulta.constrain(Alumno.class);
Constraint restriccion1 = consulta.descend("edad").constrain(18).greater();

ObjectSet res = consulta.execute();
while (res.hasNext()) {
    System.out.println(res.next());
}
```

Fig. 5. Consulta SODA con restricción.

4.2. Consultas avanzadas con concatenación

En el caso de cuando queríamos **concatenar más de una restricción en una consulta SQL,** podíamos usar los **operadores *AND* y *OR*,** de forma que si queríamos obtener los alumnos cuya edad fuese mayor que 18, o su nota final mayor que 7, podríamos usar:

```
SELECT * FROM alumno WHERE edad > 18 OR notafinal > 7
```

El lenguaje de consultas SODA también nos va a permitir realizar este tipo de consultas de forma muy simple.

Para ello debemos crear tantas restricciones con objetos de tipo *Constraint* como deseemos de la forma que hemos visto en el punto anterior.

Para poder **unir** dichas **restricciones con los operadores *OR* y *AND*** podemos hacerlo mediante los siguientes métodos:

- ***or(Constraint):*** Este método nos permitirá unir restricciones mediante el operador OR. Tendremos que pasarle la *Constraint* principal.
- ***and(Constraint):*** Este método nos permitirá unir restricciones mediante el operador AND. Tendremos que pasarle la *Constraint* principal.

Podremos concatenar tantas restricciones con los operadores *AND* y *OR* como necesitemos.

Si queremos rescatar los alumnos cuya edad sea mayor que 18 años o su nota final sea mayor que 7, podremos utilizar el código de la siguiente figura.

```
bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "alumnos.db4o");
Query consulta = bd.query();

consulta.constrain(Alumno.class);
Constraint restriccionedad = consulta.descend("edad").constrain(18).greater();
consulta.descend("notafinal").constrain(7).greater().or(restriccionedad);

ObjectSet res = consulta.execute();
while (res.hasNext()) {
    System.out.println(res.next());
}
```

Fig. 6. Consulta SODA concatenada.

Como podemos observar, hemos creado en primer lugar la consulta de la edad, y posteriormente hemos aplicado la consulta de la nota final a través del método *or*.

4.3. Consultas avanzadas con ordenación

La última de las operaciones que vamos a estudiar es la **ordenación de datos en las consultas**.

Cuando en SQL queremos obtener los alumnos cuya edad sea mayor que 18 y estén ordenados por nombre podemos utilizar la siguiente consulta:

```
SELECT * FROM alumno WHERE edad > 18 ORDER BY nombre
```

El lenguaje de consultas SODA también nos va a permitir realizar este tipo de consultas compartiendo la misma filosofía.

Para ello, como en los casos anteriores, debemos crear tantas restricciones con objetos de tipo *Constraint* como deseemos.

Para poder ordenar los datos tenemos los siguientes **métodos**:

- ***orderDescending():*** Este método nos va a permitir ordenar de forma descendente.
- ***orderAscending():*** Este método nos va a permitir ordenar de forma ascendente.

Para poder aplicar la restricción de ordenación deberemos descender con el método *descend* al atributo por el que queremos ordenar, pudiendo ordenar por más de un criterio.

Si queremos obtener los alumnos cuya edad sea mayor que 18 y estén ordenados por nombre podemos escribir el código de la siguiente figura.

```
bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "alumnos.db4o");
Query consulta = bd.query();

consulta.constrain(Alumno.class);
Constraint restriccionedad = consulta.descend("edad").constrain(18).greater();
consulta.descend("nombre").orderAscending();

ObjectSet res = consulta.execute();
while (res.hasNext()) {
    System.out.println(res.next());
}
```

Fig. 7. Consulta SODA con ordenación.

Vídeo 1. "Consultas SODA"
https://bit.ly/3i5BqLY

/ 5. Consultas en tipos de datos estructurados

Ya hemos podido comprobar que con el lenguaje de consultas SODA podemos realizar cualquier consulta que necesitemos de una forma muy sencilla, pudiendo emular muchos de los comportamientos de SQL.

Pero ¿qué ocurre cuando utilizamos la composición de clases en nuestro proyecto? es decir, cuando dentro de una de nuestras clases tenemos otros objetos como variables de clase.

En este caso de dice que tenemos un **tipo de dato estructurado** y el lenguaje de consultas SODA nos va a permitir trabajar con ellos de forma habitual tal y como lo hemos aprendido.

Vamos a definir a continuación, qué clases vamos a utilizar para exponer el funcionamiento de las consultas SODA con los tipos de datos estructurados.

Trabajaremos con la clase Fecha, la cual va a tener los siguientes atributos:

- **dia:** Este atributo va a representar el día de la fecha. Será del tipo *int*.
- **mes:** Este atributo va a representar el mes de la fecha. Será del tipo *int*.
- **anio:** Este atributo va a representar el año de la fecha. Será del tipo *int*.

También utilizaremos una clase Persona, la cual va a tener los siguientes atributos:

- **nombre:** Este atributo va a representar el nombre de la persona. Será del tipo *String*.
- **apellidos:** Este atributo va a representar los apellidos de la persona. Será del tipo *String*.
- **fechanacimiento:** Este atributo va a representar la fecha de nacimiento de la persona. Será del tipo *Fecha*.

Como podemos observar, dentro de la clase *Persona* tenemos un objeto del tipo Fecha, lo cual la convierte en un tipo de dato estructurado.

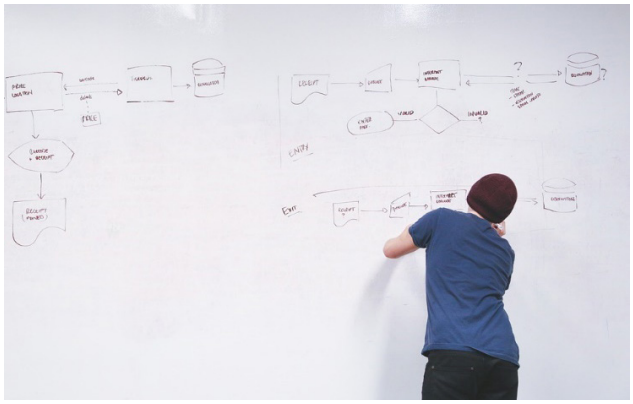

Fig. 8. Datos estructurados.

5.1. Consulta de datos estructurados

Las consultas SODA con tipos de datos estructurados funcionan de la misma forma que con datos no estructurados.

A través del método *descend* podremos descender a los atributos a los que queramos aplicar la restricción oportuna.

Como vimos anteriormente, lo primero que debemos hacer es **crear un objeto de tipo *Query*, para poder lanzarle las consultas SODA**:

```
Query consulta = bd.query();
```

Igual que ya hemos hicimos para casos anteriores, lo siguiente a implementar sería ir **agregando restricciones** a la consulta, como si de un *WHERE* en *SQL* se tratara.

La primera restricción que tenemos que agregar siempre va a ser en la que le indicaremos el tipo de objeto que queremos rescatar a la consulta.

La diferencia aquí radica en cómo podemos acceder a los atributos del objeto que está dentro de nuestro objeto.

Para poder acceder a los atributos del objeto interior tendremos que realizar dos *descend*, uno para acceder al objeto, indicando el nombre del mismo, y seguidamente otro para acceder al atributo del objeto en cuestión.

Por ejemplo, si queremos recuperar todas las personas que hayan nacido en el mes de septiembre, podremos hacerlo mediante el código que se muestra es la siguiente figura.

```
bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "alumnos.db4o");
Query consulta = bd.query();

consulta.constrain(Alumno.class);
Constraint restriccionedad = consulta.descend("edad").constrain(18).greater();
consulta.descend("nombre").orderAscending();

ObjectSet res = consulta.execute();
while (res.hasNext()) {
    System.out.println(res.next());
}
```

Fig. 9. Consulta SODA estructurada I.

Como observamos en el código, primero descendemos al objeto *"fechanacimiento"* y después al atributo *"mes"*.

Imaginemos ahora que necesitamos obtener objetos de tipo *Persona*, pero con restricciones con los operadores de mayor, igual, etc.,

Esto es perfectamente posible llevarlo a cabo con datos de tipo estructurado, y se implementaría con los mismos métodos que hemos visto anteriormente.

Si queremos **aplicar** alguno de **estos métodos a un tipo de dato estructurado**, en primer lugar, tenemos que **descender al objeto para así poder seguir descendiendo a los atributos de dicho objeto**.

Todas estas restricciones las vamos a poder concatenar sin problema alguno con los operadores *AND* y *OR*, representados con los métodos *and* y *or* como hemos visto anteriormente.

Podremos concatenar restricciones de los atributos del objeto con los de sus objetos internos sin ningún tipo de problema.

También vamos a poder **ordenar los objetos que rescatemos que sean tipos de datos estructurados**, mediante los métodos de ordenación ascendente y descendente que hemos visto en los puntos anteriores.

Vamos a poder **ordenar tanto por datos del objeto estructurado, como por datos de los atributos de la clase**, o incluso ordenar por **varios atributos estando unos en el objeto estructurado u otros en la clase**.

Imaginemos que queremos obtener todas las personas cuyo apellido sea Martínez y cuyo mes de nacimiento sea agosto, además, deberán estar ordenadas de mayor a menor por el día de la fecha de nacimiento.

Para esto podemos realizar la consulta que se muestra en la siguiente figura.

```
bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(),
        "personas.db4o");
Query consulta = bd.query();

consulta.constrain(Persona.class);

Constraint restricape = consulta.descend("apellido").constrain("Martínez");
consulta.descend("fechanacimiento").descend("mes").constrain(8).and(restricape);
consulta.descend("fechanacimiento").descend("dia").orderDescending();

ObjectSet res = consulta.execute();
while (res.hasNext()) {
    System.out.println(res.next());
}
```

Fig. 10. Consulta SODA estructurada II.

Vídeo 2. "Consultas estructuras con SODA"

https://bit.ly/2XB6UzP

/ 6. Caso práctico 2: "¿Cuándo se deberá usar cada tipo de base de datos?"

Planteamiento: Nuestros amigos han terminado por fin el proyecto del gestor de la barra de un bar mediante bases de datos orientadas a objetos, consiguiendo que tenga exactamente la misma funcionalidad que en su versión con bases de datos relacionales. "Estoy muy contento, después de tanto tiempo ya tenemos este proyecto terminado y totalmente funcional", le dice José a Pilar.

Pilar le responde que ella también está muy satisfecha y que una vez que están en producción ambos proyectos no se puede saber cuál es el que utiliza cada una de las bases de datos. Pilar le dice a José que esto la hace pensar, "si se puede hacer lo mismo con una base de datos orientada a objetos que con una relacional, ¿cuándo utilizar una y cuando otra?".

Nudo: ¿Qué piensas al respecto? ¿Cuándo crees que deberemos utilizar una base de datos orientada a objetos o una relacional? ¿Hay proyectos en los que utilizar una de las dos sea mejor o más óptimo?

Desenlace: El mundo del desarrollo de software es muy amplio, al igual que el de las bases de datos. Uno de los primeros pasos a realizar antes de empezar a programar un proyecto, como ya sabemos, es su diseño, y es en esta fase de diseño en la que decidiremos qué tipo de base de datos utilizar.

Las bases de datos orientadas a objetos son bastante más complejas de utilizar que las relacionales. Parece como si mentalmente estuviéramos mejor preparados para visualizar la clasificación de los datos de forma relacional, por lo que normalmente en los proyectos hay como una tendencia natural para usar una base de datos relacional.

Esto no quita que las orientadas a objetos se usen en gran medida, además de que existen gestores de bases de datos que soportan ambas, como por ejemplo Oracle.

Un punto a tener en cuenta, es que actualmente está a la orden del día el manejo del Big Data, para el cual, las bases de datos relacionales no son muy eficientes, así que se utilizará otro tipo de base de datos.

Fig. 11. Consultando una base de datos.

/ 7. Eliminación y actualización de datos de tipo estructurado

Como hemos visto en puntos anteriores, es posible recuperar datos de tipos estructurados aplicando el lenguaje de consultas SODA.

El siguiente paso natural es saber si, ya que se pueden rescatar, podemos realizar operadores de borrado y de actualización de este tipo de datos.

La respuesta, como se cabría esperar, es que sí. Si queremos **eliminar objetos de tipo estructurado** tendremos que seguir los mismos pasos que estudiamos en unidades anteriores para eliminar objetos normales:

1. **Seleccionar los objetos que queremos eliminar.** Para esto tendremos que diseñar la consulta SODA que nos permita obtener los objetos que deseamos.

2. **Eliminar los objetos**. Para eso podemos utilizar el **método *delete***, pasándole el objeto que queremos eliminar.

Por otra parte, si queremos **actualizar la información objetos de tipo estructurado**, igualmente, tendremos que seguir los mismos pasos que ya conocemos de unidades anteriores para actualizar objetos:

1. **Seleccionar los objetos que queremos actualizar.** Para esto tendremos que diseñar la consulta SODA que nos permita obtener los objetos que deseamos.

2. **Actualizar la información de los objetos.** Para ello tendremos que recorrer todos y cada uno de los objetos a actualizar y cambiar sus datos mediante los métodos ***set* pertinentes**, para después volver a insertarlos con el **método *store***, para que así, se actualicen dentro de la base de datos orientada a objetos.

Fig. 12. Actualizando datos.

/ 8. Aplicando el Modelo Vista-Controlador

Ya conocemos de unidades anteriores el Modelo Vista-Controlador. Si recordamos, este es un patrón de diseño de software que divide nuestro proyecto en tres paquetes:

- **Modelo:** Donde estarán **todas las clases que van a representar los modelos de datos.** En nuestro caso, aquí es donde crearemos las clases de las que crearemos los objetos que vamos a almacenar dentro de nuestra base de datos orientada a objetos.

- **Controlador:** Donde estarán **todos los controladores que van a gestionar nuestros datos**. En nuestro caso, aquí es donde vamos a crear el gestor de la conexión con la base de datos orientada a objetos y los gestores de los objetos de nuestra base de datos.

- **Vista:** Donde estarán **todas las clases que representen las ventanas de interfaz gráfica**, en el caso de que las usemos. Si en un proyecto no hay interfaz gráfica, este paquete estará vacío.

Este tipo de patrón es ampliamente usado en el desarrollo de *software* y es altamente recomendable empezar a utilizarlo cuando antes, ya que hará que nuestros proyectos estén ordenados y sean muy fáciles de entender.

En la siguiente figura podemos ver el esquema de un proyecto que utiliza bases de datos orientadas a objetos y usa el Modelo Vista-Controlador.

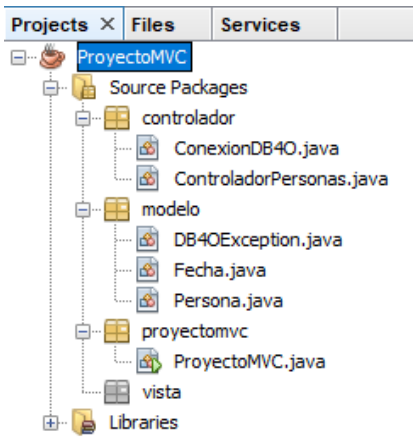

Fig. 13. Aplicando el Modelo Vista-Controlador.

Podemos ver, que como no utilizamos ningún tipo de vista gráfica el paquete de la vista está vacío.

/ 9. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad hemos profundizado en las bases de datos orientadas a objetos.

Hemos visto en qué consisten las consultas SODA, comprobando que son mucho más versátiles que las QBE ya que permiten realizar consultas más complejas.

Hemos aprendido también a realizar las mismas consultas que con QBE, pero utilizando SODA.

Además, hemos estudiado el procedimiento para consultar los tipos de datos estructurados, que son los objetos que tienen como atributo de la clase otro objeto. Hemos conocido cómo se comportan y cómo podemos aplicar las consultas SODA sobre ellos, dentro de una base de datos orientada a objetos.

Es importante recordar de esta unidad que las consultas SODA pueden realizar consultas que las QBE, por definición, no pueden, por lo que conviene usar consultas SODA en gran medida.

Resolución del caso práctico de la unidad

Cuando estamos trabajando con bases de datos, sean del tipo que sean, siempre vamos a necesitar lanzar una serie de consultas sobre ellas para recuperar información.

Como ya vimos en la unidad de bases de datos relacionales, puede haber consultas muy sencillas, pero otras no tanto, y todas deben poder ejecutarse.

Las consultas QBE, vistas en la unidad anterior, nos permitían realizar consultas básicas, seleccionar todos los objetos de un tipo, seleccionar todos los objetos de un tipo que cumplan con una o varias condiciones y no mucho más.

Cuando necesitamos ejecutar consultas más complejas ya no podemos usar QBE, sino que tendremos que usar las consultas SODA, las cuales nos permiten realizar consultas mucho más complejas que nos permitirán satisfacer las necesidades del desarrollo que estemos llevando a cabo.

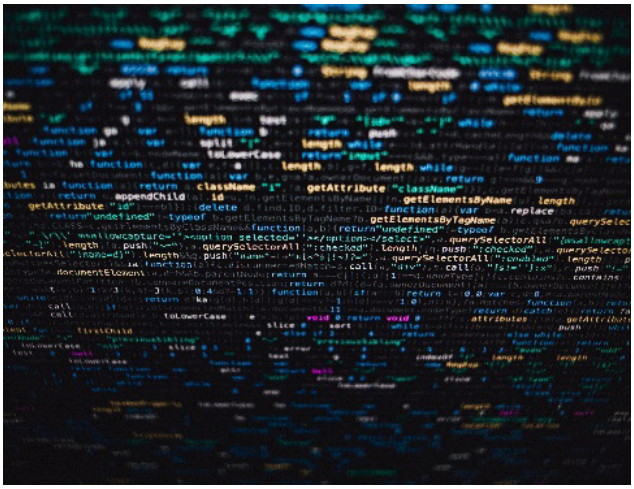

Fig. 14. Información.

/ 10. Bibliografía

Webgrafía

Colaboradores de Wikipedia. (2020, mayo 25). Base de datos - Wikipedia, la enciclopedia libre. Recuperado 3 de junio de 2020, de https://es.wikipedia.org/wiki/Base_de_datos

Colaboradores de Wikipedia. (2019, diciembre 21). Base de datos orientada a objetos - Wikipedia, la enciclopedia libre. Recuperado 3 de junio de 2020, de https://es.wikipedia.org/wiki/Base_de_datos_orientada_a_objetos

Isabelvalera55, M. (s. f.). Programación orientada a objetos. Oracle y SQL Server. - Monografias.com. Recuperado 3 de junio de 2020, de https://www.monografias.com/trabajos4/basesdatos/basesdatos.shtml