Ciclo Formativo DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Módulo 6 Acceso a Datos

Unidad Formativa 2

INICIATIVA Y COORDINACIÓN
IFP Innovación en Formación Profesional

Supervisión editorial y metodológica:
Departamento de Producto de Planeta Formación

Supervisión técnica y pedagógica:
Departamento de Enseñanza de **IFP** Innovación en Formación Profesional

Módulo: Acceso a Datos
UF 2 / Desarrollo de Aplicaciones Multiplataforma

MÓDULO 6

Unidad Formativa 2

Acceso a Datos

Esquema de contenido

OBJETIVOS

- Se saben identificar las ventajas e inconvenientes de las bases de datos que almacenan objetos.
- Se saben establecer y cerrar conexiones.
- Se sabe gestionar la persistencia de objetos.
- Se saben desarrollar aplicaciones que realizan consultas.
- Se saben modificar los objetos almacenados.
- Se saben gestionar las transacciones.

1. FORMAS DE ACCESO A UN FICHERO. CLASES ASOCIADAS

Como se ha comentado anteriormente, los SGBDR no son una alternativa óptima para almacenar objetos tal y como se entienden en la P00 (Programación Orientada a Objetos). Esto es debido a que el modelo 00 atiende a unas características que no son contempladas por el modelo relacional, por tanto, por la implementación que los SGBDR hacen de este modelo. Usar un SGBDR para almacenar objetos persistencia) incrementa significativamente la complejidad de un desarrollo de software ya que obliga a una conversión de objetos a tablas relacionales. Esta conversión implica: Mayor tiempo de desarrollo. El tiempo empleado en generar el código para la conversión de objetos a tablas y viceversa.

Mayor posibilidad de errores debidos a la traducción, ya que no siempre es posible traducir la semántica de la OO a un modelo relacional.

Mayor posibilidad de inconsistencias debidas a que el proceso de paso de modelo relacional a 00 y viceversa puede realizarse de forma diferente en las distintas aplicaciones.

Mayor tiempo de ejecución debido a la obligada conversión.

Por el contrario, si se usa un SGBD-00 los objetos se almacenan directamente en la base de datos, empleando

las mismas estructuras y relaciones que los lenguajes de P00. Así, el esfuerzo del programador y la complejidad del desarrollo software se reducen considerablemente y se mejora el flujo de comunicación entre todos los implicados en un desarrollo (usuarios, ingenieros software y desarrolladores).

Un SGBD-00 debe contemplar las siguientes características:

- Características propias de la OO. Todo sistema OO debe cumplir características como Encapsulación, Identidad, Herencia y Polimorfismo, junto con Control de tipos y Persistencia.
- Características propias de un SGBD. Todo sistema gestor de bases de datos debe permitir 5 características: Persistencia, Concurrencia, Recuperación ante fallos, Gestión del almacenamiento secundario y facilidad de Consultas.

Todas estas características están ampliamente explicadas en el Manifiesto de las bases de datos OO propuesto en diferentes etapas por los expertos en bases de datos más prestigiosos de los años 80 y 90:

- Atkinson en 1989 propuso el Manifiesto de los sistemas de bases de datos orientadas a objetos puras.
- Stonebraker en 1990 propuso el Manifiesto de los SGBD de tercera generación, que propone las características que deben tener los sistemas relacionales para almacenar objetos. Esta propuesta es justamente la que contemplan los actuales SGBD-OR. Estas características fueron ampliadas en 1995 por Daruven y Date.

El manifiesto de Atkinson expone las siguientes características que todo SGBD-OO debe implementar:

1. Almacén de Objetos complejos: los SGBD-OO deben permitir construir objetos complejos aplicando constructores sobre objetos básicos.
2. Identidad de los objetos: todos los objetos deben tener un identificador que sea independiente de los valores de sus atributos.

3. Encapsulación: los programadores solo tendrán acceso a la interfaz de los métodos, de modo que sus datos e implementación estén ocultos.
4. Tipos o clases: el esquema de una BDOO incluye únicamente un conjunto de clases (o un conjunto de tipos).
5. Herencia: un subtipo o una subclase heredará los atributos y métodos de su supertipo o superclase, respectivamente.
6. Ligadura dinámica: los métodos deben poder aplicarse a diferentes tipos (sobrecarga). La implementación de un método dependerá del tipo de objeto al que se aplique. Para proporcionar esta funcionalidad, el sistema deberá asociar los métodos en tiempo de ejecución.
7. Completitud de cálculos usando el lenguaje de manipulación de datos (Data Management Language (DMIL)).
8. El conjunto de tipos de datos debe ser extensible. Además, no habrá distinción en el uso de tipos definidos por el sistema y tipos definidos por el usuario.
9. Persistencia de datos: los datos deben mantenerse (de forma transparente) después de que la aplicación que los creó haya finalizado. El usuario no tiene que hacer ningún movimiento o copia de datos explícita para ello.
10. Debe ser capaz de manejar gran cantidad de datos: debe disponer de mecanismos transparentes al usuario, que proporcionen independencia entre los niveles lógico y físico del sistema.
11. Concurrencia: debe poseer un mecanismo de control de concurrencia similar al de los sistemas convencionales.
12. Recuperación: debe poseer un mecanismo de recuperación ante fallos similar al de los sistemas relacionales (igual de eficientes).
13. Método de consulta sencillo: debe poseer un sistema de consulta de alto nivel, eficiente e independiente de la aplicación (similar al SQL de los sistemas relacionales).

En resumen, los SGBD-OO nacen de la necesidad de proporcionar persistencia a los desarrollos hechos con lenguajes de programación 00. Hay varias alternativas para proporcionar la persistencia, sin embargo, las alternativas naturales utilizar un sistema gestor que permita conservar y explotar todas las posibilidades que la P00 permite. Esta alternativa la constituyen los SGBD-00. Estos sistemas, al igual que ocurre con los SGBDR, han sido ampliamente

estudiados para dar con la mejor solución posible y la más estandarizada. El manifiesto de Atkinson detalla muy bien ese estudio ya que define qué debe tener obligatoriamente todo sistema gestor que quiera llamarse SGBD-OO.

El manifiesto de Atkinson es básicamente una declaración de intenciones. Para garantizar que la industria de las bases de datos siga unas pautas comunes (estándares) para desarrollar SGBD-OO son necesarias organizaciones o grupos que completen las características que todo SGBD-OO debe tener y además exija su cumplimiento para favorecer la interoperabilidad entre sistemas de diferentes fabricantes. En los SGBDR, ISO y ANSI velan por el estándar SQL. En los SGBD-OO es ODMG (Object Data Management Group) la organización encargada de estandarizar todo lo relacionados con los SGBD-00.

1.1. ODMG (OBJECT DATA MANAGEMENT GROUP)

El ODMG (Object Data Management Group) es un consorcio industrial de vendedores de SGBD-00 que después de su creación se afilió al OMG (Object Management Group). El ODMG no es una organización de estándares acreditada en la forma en que lo es ISO o ANSI pero tiene mucha influencia en lo que a estándares sobre SGBD-00 se refiere. En 1993 publicó su primer conjunto de estándares sobre el tema: el ODMG-93, que en 1997 evolucionó hacia el ODMG 2.0. En enero de 2000 se publicó el ODMG 3.0. La última aportación referente a los SGBD-OO es la realizada por el OMG según ODMG 3.0 llamada "base de datos de 4° generación publicada en 2006.

Entre muchas otras especificaciones el estándar ODMG define el modelo de objetos que debe ser soportado por el SGBD-00. ODMG se basó en el modelo de objetos del OMG (Object Management Group) que sigue una arquitectura de núcleo-componentes. Por otro lado, el lenguaje de base de datos es especificado mediante un lenguaje de definición de objetos (ODL) que se corresponde con el DDL de los SGBD relacionales, un lenguaje de manipulación de objetos (OML) y un lenguaje de consulta (OQL), que equivale al archiconocido SQL de ANSI-ISO. La arquitectura propuesta por ODMG incluye además un método de conexión con lenguajes tan populares como Small talk, Java y C. En las siguientes secciones se definirán con más precisión el modelo ODMG y los lenguajes ODL, OML y OQL.

1.2. EL MODELO DE DATOS ODMG

El modelo de objetos ODMG permite que los diseños OO y las implementaciones usando lenguajes OO sean portables entre los sistemas que lo soportan. El modelo de datos dispone de unas primitivas de modelado. Estas primitivas subyacen en la totalidad de los lenguajes orientados a objetos puros (como Eiffel, Small talk, etc.) y en mayor o menor medida en los híbridos (por ejemplo: Java, C, etc.).

Las primitivas básicas de una base de datos orientada a objetos son los objetos y los literales.

- Un objeto es una instancia de una entidad de interés del mundo real. Los objetos necesitan un identificador único (Identificador de Objeto (OID).
- Un literales un valor específico. Los literales no tienen identificadores. Un literal no tiene que ser necesariamente un solo valor, puede ser una estructura o un conjunto de valores relacionados que se guardan bajo un solo nombre (por ejemplo, enumeraciones).

Los objetos se dividen en tipos. Sin ser estrictos en la definición, un tipo se puede entender como una clase en POO. Los objetos de un mismo tipo tienen un mismo comportamiento y muestran un rango de estados común:

- El comportamiento se define por un conjunto de operaciones que pueden ser ejecutadas por un objeto del tipo (métodos en P00).
- El estado de los objetos se define por los valores que tienen para un conjunto de propiedades. Las propiedades pueden ser:
- Atributos. Los atributos toman literales por valores y son accedidos por operaciones del tipo getvalue y setvalue (como exige la OO pura, y nunca se accede a ellos directamente).
- Relaciones entre el objeto y uno o más objetos. Son propiedades que se definen entre tipos de objetos, no entre instancias. Las relaciones pueden ser uno-a-uno, uno-a-muchos o muchos-a-muchos.

Un tipo tiene una interfaz y una o más implementaciones. La interfaz define las propiedades visibles externamente y las operaciones soportadas por todas las instancias del tipo. La implementación define la representación física de las instancias del tipo y los métodos que implementan las operaciones definidas en la interfaz.Los tipos pueden tener las siguientes propiedades:

- Supertipo. Los tipos se pueden jerarquizar (herencia simple). Todos los atributos, relaciones y operaciones definidas sobre un supertipo son heredadas por los subtipos. Los subtipos pueden añadir propiedades y operaciones adicionales para proporcionar un comportamiento especializado a sus instancias. El modelo contempla también la herencia múltiple, y en el caso de que dos propiedades heredadas coincidan en el subtipo, se redefinirá el nombre de una de ellas.
- Extensión: es el conjunto de todas las instancias de un tipo dado. El sistema puede mantener automáticamente un índice con los miembros de este conjunto incluyendo una declaración de extensión en la definición de tipos. El mantenimiento de la extensión es opcional y no necesita ser realizado para todos los tipos.
- Claves: propiedad o conjunto de propiedades que identifican de forma única las instancias de un tipo (OID). Las claves pueden ser simples (constituidas por una única propiedad) o compuestas (constituidas por un conjunto de propiedades).

1.3. EL MODELO DE DATOS ODMG

ODL (lenguaje de definición de objetos) es un lenguaje para definir la especificación de los tipos de objetos en sistemas compatibles con ODMG. ODL es el equivalente al DDL (lenguaje de definición de datos) de los SGBD relacionales. ODL define los atributos y las relaciones entre tipos y especifica la signatura de las operaciones. ODL se utiliza para expresar la estructura y condiciones de integridad sobre el esquema de la base de datos: mientras que, en una base de datos relacional, DDL define las tablas, los atributos en la tabla, el dominio de los atributos y las restricciones sobre un atributo o una tabla, en una base de datos orientada a objetos ODL define los objetos, métodos, jerarquías, herencia y el resto de elementos del modelo OO.

Una característica importante que debe cumplir (según ODMG) un ODL es ofrecer al diseñador de bases de datos un sistema de tipos semejantes a los de otros lenguajes de programación OO. Los tipos permitidos son:

- Tipos básicos: incluyen los tipos atómicos (Boolean, Float, Short, Long, Double, Chart, etc.) y las enumeraciones.
- Tipos de interfaz o estructurados: son tipos complejos obtenidos al combinar tipos básicos por medio de los siguientes constructores de tipos:
  - Conjunto (Set<tipo>) denota el tipo cuyos valores son todos los conjuntos finitos de elementos del tipo.
  - Bolsa (Bag<tipo>) denota el tipo cuyos valores son bolsas o multiconjuntos de elementos del tipo. Una bolsa permite a un elemento aparecer más de una vez, a diferencia de los conjuntos, por ejemplo (1,2,1) es una bolsa pero no un conjunto.
  - Lista (List<tipo>) denota el tipo cuyos valores son listas ordenadas finitas conteniendo 0 o más elementos del tipo. Un caso especial lo constituye el tipo String que es una abreviatura del tipo List<char>.
  - Array (Array<tipo, i>) denota el tipo cuyos elementos son arrays de i elementos del tipo.

Por tanto, con la ayuda de ODL se puede crear el esquema de cualquier base de datos en un SGBD-OO que siga el estándar ODMG. Una vez creado el esquema, usando el propio gestor o un lenguaje de programación se pueden crear, modificar, eliminar y consultar objetos que satisfagan ese esquema.

El siguiente ejemplo muestra la definición de un esquema usando ODL para el SGBD-OOMatisse. En el ejemplo mostrado abajo se definen dos tipos complejos llamados Libro y Autor:

Ejemplo

Un Libro tiene como atributos título de tipo básico String, año y páginas de tipo básico Integer. Un Autor tiene como atributos apellidos, nombre y nacionalidad de tipo String y edad de tipo Short.

Entre ambos tipos hay relaciones definidas como conjuntos Set: un Libro es escrito por un conjunto de autores y un Autor escribe un conjunto de libros.

```
interface Libro {
/* Definición de atributos */
attribute string título;
attribute integer año;
attribute integer paginas;
atribute enum Posibles Encuadernaciones (Dura, Bolsillo) tipo;
/* Definición de relaciones */
relationship Set.<Autor> escrito_por inverse Autor: : escribe;
interface Autor{
/* Definición de atributos */
 atribute string apellidos;
attribute string nombre;
attribute string nacionalidad;
attribute short edad;
/* Definición de relaciones */
relationship setro escribe inverse Libros::escrito por;
}
```

1.4. OML (LENGUAJE DE MANIPULACIÓN DE OBJETOS)

Una importante peculiaridad de ODMG es que no define ningún lenguaje de manipulación de objetos (OMIL). El motivo es claro: dejar descansar esta tarea en los propios lenguajes de programación, es decir, que sean los lenguajes de programación los que puedan acceder a los objetos y modificarlos, cada uno con su sintaxis y sus posibilidades. El objetivo es no diferenciar en la ejecución de un programa entre objetos persistentes almacenados en una base de datos y objetos no persistentes creados en memoria.

Lo que formalmente sugiere ODMG es definir un OML que sea la extensión de un lenguaje de programación de forma que se puedan realizar las operaciones típicas de creación, eliminación, modificación e identificación de objetos desde el propio lenguaje como se haría con objetos que no fueran persistentes.

Para mostrar cómo se pueden modificar objetos directamente con un lenguaje de programación, en la Sección 3.3 se trabajará con Java para realizar modificaciones en una base de datos gestionada con Matisse.

1.5. OQL (LENGUAJE DE CONSULTAS DE OBJETOS)

OQL (lenguaje de consultas de objetos) es un lenguaje declarativo del tipo de SQL que permite realizar consultas sobre bases de datos orientadas a objetos, incluyendo primitivas de alto nivel para conjuntos de objetos y estructuras.

Tanto la definición de las bases de datos OO como de OQL fue posterior a las bases de datos relaciones y a SQL. De hecho, SQL ya estaba más que extendido y aceptado por los desarrolladores y clientes de bases de datos cuando apareció OQL. Por este motivo OQL no quiso reinventar la rueda e intentó definirse lo más parecido a la sintaxis usada en SQL (Select-From-Where) dentro de las posibilidades. Así los nuevos usuarios potenciales del OQL no apreciarían en este lenguaje diferencias significativas con respecto a SQL y obtendrían una curva de aprendizaje más rápida.

En los sistemas relacionales SQL es el estándar de consulta, sin embargo, la implementación que los sistemas comerciales hacen de SQL puede variar de unos a otros (por ejemplo, MS-Access utiliza o para usarlo como comodín mientras que Oracle utiliza %). En OQL ocurre igual, en algunas implementaciones comerciales el estándar OQL no se corresponde exactamente con la implementación realizada. En esta sección mostraremos el OQL de la base de datos Matisse. Éste, básicamente, cumple con todas las características de OqL (ODMG) pero no satisface todo estrictamente.

La Figura 3.2 muestra un ejemplo de diagrama de clases para una biblioteca. Las consultas OQL mostradas se ejemplarizarán sobre ese mismo diagrama. Un ejemplo de OQL que devuelva el título de todos los artículos cuyo número de páginas sea mayor de 30 sería:

Ejemplo

OQL 1

```
select a.titulo from articulo a where a. paginas>30;
```

En el siguiente ejemplo se muestra una consulta para obtener todos los nombres y apellidos (sin repetir) de los autores cuyos apellidos empiezan por "Mura".

Ejemplo

OQL2

```
select distinct al. nombre, a.apellidos from alltor a
where apellidos like "Mura";
```

En la consulta anterior, para cada objeto de la colección que cumple la condición se muestra el valor de los atributos incluidos en el select. El resultado por defecto es un tipo bag de tipo string y muestra todos los valores, aunque estén duplicados. Sin embargo, si se utiliza distinct el resultado es de tiposet ya que se eliminan los duplicados.

Además, OQL permite hacer reuniones entre objetos. La siguiente consulta obtiene el título de los libros escritos por todos los autores cuyo apellido empieza por "Mura%".

Ejemplo

OQL3

```
select L. titulo, a ... nombre, al. apellidos from autor a, Libro l where a. escribe=l. OID and apellidos like 'Murai";
```

La consulta anterior reúne los objetos autor con los objetos libro usando la relación escribe y el identificador único de objeto (OID).

Como se puede observar en la consulta, la sintaxis de OQL es idéntica a SQL. Sin embargo, esto ocurre solo en consultas sencillas, cuando se necesita explotar las características concretas de OO las consultas no tienen una sintaxis tan similar a SQL.

Por ejemplo, la siguiente consulta obtiene como resultado el título de los libros escritos por un autor y el nombre y apellidos. Sin embargo, el nombre y apellidos del autor no se obtienen invocándolos directamente, sino a través del método dameNombreyApellidos () definido en la clase Autor.

Ejemplo

OQL4

```
select. l. titulo, a. dameNombreyApellidos () from autor a, Libro l where a . escribe=l . ÔID;
```

Esta manera tan natural de incluir los métodos de los objetos en la propia consulta es de gran utilidad en OQL ya que no es necesario que el lenguaje posea primitivas de modificación, con la simple invocación de los métodos se puede consultar y modificar el estado (valores de las variables) de los objetos.

OQL tiene una sintaxis más rica que permite explotar todas las posibilidades de la OO. Sin embargo, no es objeto de este capítulo tratar OQL en profundidad. Para más información sobre OQL (con Matisse) se puede consultar la guía para desarrolladores de Matisseo.

2. SISTEMAS GESTORES DE BASE DE DATOS ORIENTADAS A OBJETOS

Aunque la oferta no es tan extensa como ocurre con los SGBD relacionales también hay una oferta significativa de SGBD-OO en el mercado. Como en el caso de los sistemas relacionales existen:

- Sistemas privativos, como ObjectStore,o Objectivity/DBoo Versanto
- Sistemas bajo licencias de software libre como Matisseo (versión para desarrolladores) y dib4oo (versión liberada bajo licencia por Versant).

En esta sección se pretende mostrar los pormenores de un SGBD-OO pero desde la perspectiva de una solución como Matisse. Este SGBD-OO es una alternativa que respeta en gran medida el estándar ODMG, por lo que es una buena referencia para probar los diferentes lenguajes de definición, manipulación y consulta de objetos descritos en la sección anterior.

Matisse, de ADB Inc., es un SGBD-00 que da soporte para ser manejado con C, Eiffel, Java y NET. Según sus autores, Matisse está orientado a trabajar con gran cantidad de datos con una rica estructura semántica. Además del control de transacciones, acceso, etc., propio de cualquier sistema gestor relacional y no relacional, Matisse tiene ventajas para la gestión propias de la OO. Algunas de ellas son:

- Técnicas para fragmentar objetos grandes en varios discos para optimizar así el tiempo de acceso. Una ubicación optimizada de los objetos en los discos.
- Un mecanismo automático de duplicación que proporciona una solución software a los fallos de tolerancia del hardware: los objetos (en lugar de los discos en sí) se pueden duplicar especularmente en varios discos, con recuperación automática en caso de fallo del disco.
- Un mecanismo de versiones de objetos incorporado.
- Soporte para una arquitectura cliente-servidor en la cual un servidor central gestiona los datos para un número posiblemente elevado de clientes, que mantienen una "reserva" de objetos a los que se haya accedido recientemente.
- Con respecto a la implementación del modelo OO, Matisse ofrece un mecanismo de optimización de acceso a objetos relacionados. Por ejemplo, si una clase como Libro posee como atributo Autor, Matisse mantendrá, si así se solicita, los enlaces inversos de forma automática de modo que será posible no solo acceder a los autores de un libro sino también a los libros de un autor determinado.

2.1. INSTALACIÓN DE MATISSE

Matisse puede descargarse de su web oficial para diferentes sistemas. La instalación es sencilla, un ejecutable que no necesita de ninguna configuración salvo la básica de cada sistema. Todo lo que ofrece Matisse está bien documentado en el sitio oficial. La versión con la que se trabaja en los ejemplos de este capítulo es Matisse 9.0.5 para Windows 7.

Una vez arrancado Matisse se accede a un sencillo entorno de gestión.

La ventana principal tiene tres partes bien diferenciadas:

El árbol de la izquierda muestra las bases de datos creadas. Desplegando cada una se puede acceder a los elementos (espacio de nombres, clases, métodos, atributos, etc.) de cada base de datos.

La parte de la derecha sirve para ejecutar los lenguajes ODL y OQL sobre las bases de datos y mostrar los resultados de estas y otros comandos que Matisse permite.

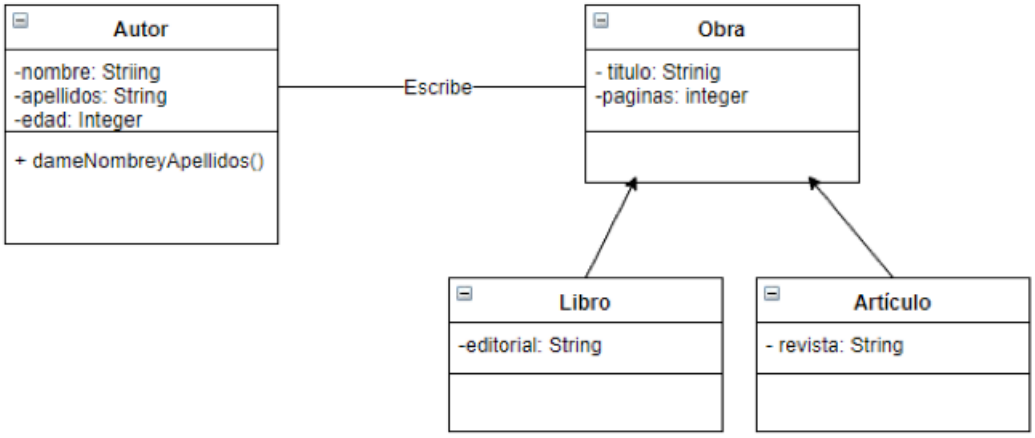
Al igual que ocurre con muchos de los sistemas gestores empleados en este libro no se pretende explicar en detalle el funcionamiento de Matisse, sino únicamente los aspectos esenciales para entender el acceso a datos desde un lenguaje de programación (Java). Por tanto, la siguiente sección da a conocer cómo se puede crear un esquema de bases de datos nuevo sobre el que más adelante se trabajará con el acceso mediante Java. Si el lector está interesado en los pormenores de este sistema gestor, puede acceder a la web oficial para recabar información más detallada.

2.2. CREANDO UN ESQUEMA CON MATISSE

En las secciones siguientes se trabajará sobre un ejemplo concreto para mostrar una aplicación de acceso a SGBDOO. Sin embargo, para ello es necesario previamente utilizar el entorno de Matisse para crear el esquema básico de base de datos. A continuación se muestra cómo crear ese esquema.

El modelo de ejemplo que se muestra en la Figura 3.2 es con el que se trabajará en las siguientes secciones y representa una simplificación de una Biblioteca. El modelo contiene:

- Una clase Autor, que representa a todos los posibles autores de una obra literaria. Sus atributos son nombre, apellidos y edad. Además tiene un método dame NombreyApellidos.0 que devuelve la concatenación del nombre y la edad en una única cadena.
- Una clase Obra, que representa a los diferentes tipos de creaciones que puede hacer un autor. Tiene como atributos título y páginas.
- Una clase Libro, que hereda de Obra y añade un atributo más llamado editorial que representa a la editorial que publica el libro.
- Una clase Artículo, que también hereda de Obra y añade otro atributo llamado revista que representa a la revista que publica el artículo.



Para crear este esquema en Matisse se deben seguir los siguientes pasos:

1. Crear una nueva base de datos (Database). Para ello se selecciona la opción de menú Server -> New Database. Entonces saldrá una ventana que pedirá el nombre de la nueva base de datos. En este ejemplo, como muestra la imagen anterior, la base de datos se llamará Prueba.

2. Creada la base de datos, esta aparece en la estructura de árbol pero con una cruz blanca sobre fondo rojo. Eso 4l indica que hay que arrancar la base de datos para poder utilizarla. Esto se hace pulsando con el botón derecho del ratón sobre el nombre de la base de datos y seleccionando la opción Start en el menú contextual que aparece.

3. Arrancada la base de datos aparecerá una estructura en el árbol similar a la de la figura siguiente.

Narnespaces representa el espacio de nombre en el cual se crearán las clases asociadas. Para una buena organización de la información es interesante definir un espacio de nombre que no cree ambigüedad sobre dónde están situados los elementos de la base de datos."

Clases albergará la estructura propia de tipos definidos. En el ejemplo será (Autor, Libro, Revista y Obra).

Methods contendrá los métodos definidos en cada clase. Hay que recordar que en un SGBD-OO (Atkinson en la Sección 2. 1) ge deben poder almacenar objetos con sus atributos y sus métodos asociados. Indexes contiene los Indexes creados sobre los objetos (creados por el usuario) para optimizar las consultas"

4. Crear una Namespace para la estructura Biblioteca. Para ello se pulga con el botón derecho del ratón sobre Namespaces y se selecciona la opción New Namespace, que aparece en el menú contextual. En la parte derecha aparecerá una plantilla de ayuda para escribir la sentencia ODL create namegpace. La Figura 3.5 muestra cómo crear el nuevo espacio de nombres (biblioteca).

5. Creado el espacio de nombres, IO siguiente es crear la estructura definida en la Figura 3.2. Para ello se pulsa en Classes y se selecciona la opción de menú New Class. Al igual que con los narnespace, en la parte derecha saldrá una plantilla con la sintaxis para crear una nueva clase. Esta plantilla utiliza un lenguaje propio de Matisse pero que luego puede ser exportado a ODL (ODMG). La Figura 3.6 muestra el código necesario para la creación de la clase Autor y se puede observar en ella cómo es necesario seleccionar de la lista (In narnespace) el namespace biblioteca para que la clase se cree en ese egpae10 de nombres y no en la raíz (root).

Las otras clases del ejemplo se crean de la misma manera con el siguiente código. Debemos observar que en el código no se especifica la relación entre Obra y Autor ni se define el método dameNombreyApellidos), estos elementos se definirán en el siguiente paso.

Ejemplo

```
CREATE CLASS Obra (
titulo string,
 paginas Integer
)
CREATE CLASS Libro
INHERIT Obra
(
editorial String
)
```

```
CREATE CLASS Articulo
INHERIT Obra (
revista String
)
```

6. Por último, falta por crear las relaciones entre Obra y Autor (escribe) y el método dameNombreyApellidos) que devuelve ambos atributos de Autor concatenados. Para ello se usa el mismo procedimiento que para crear las clases. En el árbol se selecciona la clase a la que se quiere crear una nueva relación, por ejemplo, Autor, y se pulsa con el botón derecho del ratón. En el menú contextual se selecciona Alter Class -> Add Relationship. Entonces saldrá una plantilla en la parte derecha con la sintaxis para añadir una nueva relación. El siguiente código es el necesario para crear primero una relación entre Autor y Obra (escribe) y segundo la inversa entre Obra y Autor (escrita por). Esto optimizará la recuperación de objetos. Es importante recordar que se debe poner el namespaces en Biblioteca para que el sistema sepa que las clases que se quieren relacionar están bajo ese espacio de nombres.

Ejemplo

```
ALTER CLASS Autor
 ADD RELATIONSHIP escribe
RELATIONSHIP SET ( Obra)
INVERSE Obra. escrito_por;

ALTER CLASS Obra
ADD RELATIONSHIP escrito por
RELATIONSHIP SET ( Autor)
INVERSE Autor. Escribe;
```

De la misma manera se añade un método, sin embargo en este caso se selecciona la clase Autor y pulsando en el botón derecho se selecciona la opción de menú Alter Class -> Add Method. El código para añadir un método a Autoro que concatene el nombre y el apellido será:

Ejemplo

```
CREATE METHOD dameNombrey Apellidos ()
RETURNS String
FOR Alltor
--
-- Describe your method here
--
BEGIN
return CONCAT (nombre, apellidos);
END;
```

3. INTERFAZ DE PROGRAMACIÓN DE APLICACIONES DE LA BASE DE DATOS

3.1. PREPARANDO EL CÓDIGO JAVA

Para poder entender el proceso para acceder a los objetos almacenados en las bases de datos OO hay que tener siempre presente que lo que se busca es tener la sensación de trabajar solo con objetos, sin tener que pensar en si están almacenados en una base de datos o están creados en memoria. Es decir, se busca un acceso a objetos totalmente transparentes.

Por lo tanto, cuando se trabaja con SGBD-00 hay que olvidarse de buscar algo similar a las típicas API de acceso a datos de sistemas relacionales como ODBC o JDBC. Lo que se tiene que buscar es una manera de que el SGBD-00 genere en un lenguaje de programación (Java en este caso) las clases que componen el esquema de base de datos. Esta es la clave del problema, lo que beneficia el trabajo con SGBD-OO desde lenguajes de programación OO y lo que lo diferencia a su vez del acceso a datos en sistema relacionales.

La librería de clases creada por el SGBD-00 en el lenguaje de programación seleccionado (en nuestro caso Java) puede ser integrada en un proyecto en el cual, usando la librería adecuada, se puede abordar la persistencia de un objeto con solo invocar a un método del mismo. El mecanismo por el cual el contenido de un objeto se almacena en la base de datos es totalmente transparente al programador. Solo tiene que pedir que se haga persistente y el objeto se hará, solo hay que pedir que se recupere y el objeto aparecerá en memoria como si de cualquier otro objeto se tratase.

A continuación, se creará con Matisse la estructura de clases en Java que representa el esquema Biblioteca de la base de datos RAMA.

1. Seleccionar la opción de menú Schema -> Generate Code. En el siguiente menú hay que seleccionar RAMA para que su esquema se convierta en clases Java.

   Las opciones de generación son las siguientes:

   - Project directory (directorio de proyecto): se utiliza para indicar dónde está ubicada la carpeta del proyecto Java en el que se integrará el código Java generado. Esto es útil para que las clases creadas se coloquen en la carpeta deseada y dentro del paquete seleccionado y así no tener que hacer importaciones posteriores.
   - Database namespace: se utiliza para indicar de qué espacio de nombres almacenados en la base de datos se quieren sacar las clases de las que se generará el código. En este caso será del namespace Biblioteca.
   - Language Package: sirve para indicar el paquete Java en el que se quieren agrupar las clases que se generarán.
   - SQL method call: indica si se desea hacer llamadas a SQL para invocar a los métodos definidos en la base de datos. En el ejemplo Biblioteca se creó un método en la clase Autor llamado dameNombreyApellidos.0. Seleccionando la opción SQL method call, Matisse genera el código necesario para que cuando el usuario quiera invocar este método de la clase Autor se llame directamente a su código ODL almacenado en la base de datos. De alguna manera, los métodos creados en la base de datos son como procedimientos almacenados (típicos de los SGBDR), que siempre deben ser ejecutados en el servidor de base de datos por razón de

optimización. Por esto es por lo que el código generado en el método no se traduce a Java en la generación de las clases Java, sino que solo se pone el código para conectar a la base de datos y ejecutarlo desde el propio Matisse. En las siguientes secciones se concretará este código y su acceso desde Java.

2. Una vez generadas las clases, todas se ubicarán en el paquete seleccionado dentro del directorio del proyecto seleccionado. Esas clases ya estarán listas para ser incorporadas a un proyecto y poder realizar operaciones de modificación y consulta.

3. Para que el proyecto reconozca las clases y métodos creados por Matisse es necesario incluir en el proyecto la librera matisse,jar. En el caso de que la instalación de Matisse se haga en la unidad C:\, bajo sistema Windows, la ruta donde se localiza la librería sería: C:\Products \Matisse \lib\.

3.2. AÑADIENDO OBJETOS

Una vez creadas las clases, la manera de añadir objetos a la base de datos es sencilla, solo hay que crear en Java objetos y luego llamar para almacenarlos. Evidentemente, para saber dónde se tienen que almacenar los objetos es necesario previamente establecer una conexión con la base de datos mediante su dirección (localhost si trabajo en local) y su nombre (RAMA en el ejemplo). El objeto necesario para la persistencia es:o

- Mt Database. Este objeto gestiona el acceso a Matisse. Sus métodos más destacados son:
  - Mt Database : constructor de la clase. Crea la conexión a la base de datos indicada por la dirección del servidor Matisse, el nombre de la base de datos y el espacio de nombres al que se quiere acceder dentro de la base de datos.
  - OpenO: abre la base de datos definida en el constructor.
  - start Transaction.0: crea una transacción para que lo que se haga hasta el final de la transacción sea atómico y si algo falla a media ejecución de la transacción se deshagan todos los cambios.
  - Commit 0: da por finalizada la transacción y materializa los cambios hechos en los objetos en la base de datos.
  - Close(): cierra la base de datos.
- Mt Exception: atiende las excepciones que se produzcan.

El siguiente código muestra el método creaCObjetos que tiene como parámetros el servidor Matisse (hostiname) y el nombre de la base de datos (dibname). El resto del código es utilizar Java para crear objetos sin pensar en su posible persistencia en un SGBD-OO. Únicamente hay que recalcar que la clase que contenga este código debe importar el paquete biblioteca (en el ejemplo de la Figura 3.8 es matisseoo.biblioteca) que es el que contiene las clases Java generadas de Matisse.

Una vez ejecutado este código Java, en Matisse se puede ver que los objetos se han ejecutado correctamente. Para ello se pulsa en el árbol en la clase de la que se quiere ver los objetos y en el menú se selecciona View data. Entonces aparecerán los valores de los objetos creados.

3.3. ELIMINANDO OBJETOS

El borrado de objetos se hace utilizando el método deep Remove() definido en todos los objetos de Matisse. Cuando Matisse crea el código Java correspondiente a una clase definida en su base de datos, le añade un método deep RemoveO que permite eliminar el objeto de la base de datos.

El siguiente código de ejemplo muestra el borrado de dos objetos Obra que haya en la base de datos. El código borra los dos primeros de la lista sin especificar cuáles son. Como puede observarse, el código conecta y desconecta usando los mismos métodos de Mt Database vistos en el ejemplo anterior. Para el borrado se utiliza el método Obra. deep RemoveO sobre dos de los objetos de la lista de Obras rescatados. Para saber cuántos objetos de tipo Obra hay en la base de datos utiliza el método Obra.getInstanceNumberdb) de la clase Obra (creado automáticamente también al generar el código Java de la clase Obra desde Matisse).

Es interesante resaltar que el ejemplo utiliza un iterador Mt Object Iterator«Obras iter = Obra.<Obras instance Iterator(db) para recorrer los objetos. La clase MObject Iterator está definida en matissejar y se utiliza como cualquier clase Iterator de Java.

Ejemplo

```
public Static void borra Objetos (String hostiname, String dibname)

{

try {

MtDatabase = new MtDatabase (hostname, dbname, new Mt PackageÖbject Factory (“”, “biblioteca” ) ) ; db.open();

do. StartTransaction ();

// Lista todos los objetos obra con el método get InstanceNumber System.out.println(“\n” + Obra. getInstanceNumber (db) + “ Obra (s) en la DB. “ ) ; //Crea un Iterador (propio de Java) MtObjectIterator<obra > iter = Obra. <Obra>instanceIterator (db) ;

System. out.println (“Borra dos Obras”); while (iter .hasNext () ) { Obra [] obras = iter . next ( 2 ) ; System. out.println (“Borrando “ + obras. length + ‘’ Obra (s) ...”) ; for (int i-0; i < obras.length; i-H ) { //borra definitivamente el objeto obrasi).deepRemove (); // Solo borra dos y lo deja break; } iter. close () ; //materializa los cambios y cierra la BD db.commit (); db.close();

System.out.println(“An HEcho.”); } catch (MtException mte) { System. Out“.print. In (“MEException : “ + mte.getMessage () );

Otra opción de borrado es eliminar todos los objetos de una clase. Para esta acción se utiliza el método get Class(db). remove All Instances(); por ejemplo, si se desearan borrar todos los objetos de la clase Obra se usaría Obra.get Class(db).

removeAll‘Instances0;
```

3.4. MODIFICANDO OBJETOS

Con lo visto en las secciones anteriores, modificar objetos almacenados en la base de datos es simple. Solo hay que encontrar el objeto buscado, por ejemplo, recorriendo con un iterador Mt Object Iterator, y una vez encontrado cambiar sus propiedades con los métodos set de cada objeto.

El siguiente código muestra un método de modificación: cambia la edad de los autores cuyo nombre sea Haruki. La nueva edad se le pasa como parámetro.

Los parámetros de entrada del método serán la dirección del servidor Matisse, el nombre de la base de datos, el nombre del Autor buscado y la nueva edad para ser modificada.

El código muestra el proceso:

Ejemplo

```
public static void Modifica Objeto (String hostiname, String dibname, String nombre, String nueva Edad){
 int in Autores=0;
 try {
MtDatabase db = new MtDatabase (hostname, dbname, new MtpackageObject Factory (“”, “ biblioteca“ ) ) ;
db.open();
do. StartTransaction ();
// Lista cuántos objetos Obra con el método get InstanceNumber
System.out.println(“\n” + Autor.getInstanceNumber (db) + * Autores en la DB .. “ ) ;
nAutores= (int) Autor.get InstanceNumber (dib);
//Crea un Iterador (propio de Java)
MtobjectItErator«Autor» iter = Autor. «Autor»instance Iterator (dib);
System. out.printlin (“recorro el iterador de uno en uno y cambio cuando encuentro “nombre” “);
while (iter .hasNext () ) {
Autor [] autores = iter.next (nAutores);
for (int is 0; i < autores.length; it ) {
//Busca una autor con nombre nombre
if (autores (i).get Nombre () . CompareTo (nombre)==0) autores (i. set Edad (nueva Edad):
iter. close () ;
//materializa los cambios y cierra la BD,
db.commit ();
db.close();
System.out.println(\niecho.”); } catch (MtException mte) {
System. out.print ln (“Mt Exception : “ + mte.getMessage ()) ;
```

3.5. CONSULTANDO OBJETOS CON OQL

La última de las operaciones básicas sobre una base de datos es la ejecución de consultas. En esta sección se muestran las posibilidades de ejecutar consultas OQL sobre Matisse y desde Java. Las consultas OQL pueden ejecutarse directamente sobre el propio entorno de Matisse, sin embargo, esta sección se centra en la utilización de Java para enviar consultas OQL usando JDBC al servidor Matisse y para tratar los resultados con código Java. Para entender bien este modo de ejecución es necesario que el lector esté familiarizado con el acceso a datos con JDBC, visto en el Capítulo 2.

Una peculiaridad de Matisse es que llama a su lenguaje de consulta con el acrónimo SQL en vez de llamarlo OQL como cabría esperar de un SGBD-00. Los creadores de Matisse entienden que su lenguaje de consulta OQL es tan sencillo y similar al famoso SQL que prefieren utilizar el significado de sus siglas (Structured Query Language) para denominarlo y así hacerlo más familiar para los nuevos usuarios del entorno. En esta sección se llamará al lenguaje OQL por diferenciar con su sintaxis real, aunque en el entorno Matisse y en toda su documentación lo llaman SQL.

Hay dos maneras de ejecutar consultas sobre Matisse. La primera de ellas es utilizando el editor de consultas del propio entorno. La segunda es utilizando el Java JDBC desde el propio código del programa.

3.5.1. Consultas desde JDBC de Java

La solución para ejecutar consultas desde Java es utilizar JDBC. Como se vio en el Capítulo 2, JDBC es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se acceda. Para ejecutar una consulta desde JDBC de Matisse se siguen los siguientes pasos:

Crear un objeto MDatabase con los parámetros de conexión a la base de datos Matisse. Estos parámetros son, principalmente, la dirección del servidor y el nombre de la base de datos. Un ejemplo de conexión es mostrado en la siguiente línea de código:

Mt Database dibcon = new Mt Database (hostiname, dibname) ;

Esta manera de conexión con la base de datos es la misma que se ha seguido en los ejemplos anteriores.

Se abre una conexión y se crea un objeto Statement, que es el que ejecutará la consulta. El siguiente código muestra estas dos operaciones:

dbcon.open(); Statement stimt = db.com. createStatement ();

Si se usa Prepared Statement se puede utilizar getJDBCConnection.0; para obtener y trabajar con una conexión fuente JDBC.

Connection db.con = dbcon. get JDBCConnection();

3 Ejecutar la consulta OQL con el método executeQuery 0 de Statement. El resultado de la consulta se guarda en un objeto de tipo ResultSet que podrá ser recorrido para obtener los valores devueltos por la consulta (método mext:0). El siguiente código muestra el uso de execute Query 0:

ResultSet rset = sitmt.executa Query (“select l. titulo, a. nombre, a.apellidos from autor a, Libro 1 Where a ... escribe=l ... OID”) ;

Una vez ejecutada una consulta y obtenidos los datos del objeto Result Set, tanto los objetos ResultSet y

Statement como la conexión deben ser cerrados con sus respectivos métodos close () para evitar problemas en la ejecución.

Los pasos descritos para ejecutar consultas con JDBC sirven de marco general para ejecutar consultas sobre cualquier sistema gestor de bases de datos relacional u objetual. Es decir, en el ResultSet se almacenan los valores devueltos por la consulta y estos se recorren de la misma manera con independencia de si esos valores son devueltos por un sistema relacional u objetual.

Sin embargo, el resultado de una consulta OQL no tiene por qué ser únicamente un valor o conjunto de valores (título, nombre, apellidos, etc.) sino que puede ser un objeto almacenado en la base de datos. Esta es una importante diferencia con respecto a las bases de datos relacionales, en las cuales el resultado devuelto por una consulta nunca será un objeto.

Las consultas OQL permiten devolver referencias a los objetos almacenados con el método REFO. Por ejemplo, la siguiente consulta devuelve todos los objetos tipo Autor almacenados en el espacio de nombres biblioteca:

Ejemplo

OQL 5

SELECT RIEF (a) from biblioteca . Autor a;

En este caso, el ResulSet contendrá referencias a objetos que deberán ser mapeados a objetos Java para poder obtener la información que cada uno contiene.

El siguiente código muestra un ejemplo de recuperación de objetos Autor ejecutando la consulta OQL 5 anterior.

Se define el método ejecutarOQLO que tiene como parámetros la dirección del servidor (hostiname) y el nombre de la base de datos (dbname).

```
public Static void ejecutarOQL (String hostinama, String dibname)
{
Mt Database dibcon = new Mt Database (hostiname, doname) ;
//Abre una conexión a la base de datos
dbcon.open();
try{
// Crea una instancia de Statement
Statement stint = db.con. CreateStatement();
// Asigna una consulta CQL. Utiliza REF () para obtener el objeto directamente en vez de //obtener valores concretos (que también podría ser).
String commandText = "SELECT REF(a) from biblioteca. Autor a; ";
// Ejecuta la consulta y obtiene un ResultSet
ResultSet riset = Stmt.executeQuery (command Text);
Autor a1i ;
// Recorre el rset uno a uno
while (rset.next ()) {
```

```
 // Obtiene el objeto Autor
a1 = (Autor) rset.getObject (1) ;
// Imprime los atributos de cada objeto en forma de tabla.
System.out.println (
"Autor: " + String. format ($16s", al.getNombre ()) + String. format ($16s", al.getApellidos ()) + "
Páginas: " + String. format ("$16s", al. get Edad ()));
// Cierra las conexiones.
rset. close();
stmt.close();
dbcon.close();
}
catch (SQLException e)
{
System.out.println ("SQLException: " + e.getMessage () ) ;
}
}
```

El código sigue los pasos de conexión, creación de Statement y ejecución de consultas con executeQuery0 vistos previamente. La principal diferencia es cómo trata a los objetos almacenados en el Result Set. Al saber que el resultado de la consulta debe devolver objetos de tipo Autor, se crea una variable Autor a 1, que será la que sirva de auxiliar para almacenar los objetos devueltos en el ResultSet. El método get Object0 de ResultSet es el que permite recuperar el objeto especificado y almacenarlo en al con el molde pertinente. El siguiente código extraído del ejemplo muestra esta recuperación:

```
a1 = (Autor) riset.getObject (1) ;
```

Una vez el objeto es referenciado en al ya se pueden recuperar los valores de sus atributos o incluso ejecutar los métodos como cualquier otro objeto almacenado en memoria.

Este ejemplo muestra solo una de las muchas posibilidades que permite la ejecución de consultas OQL sobre Matisse y su posterior tratamiento en Java. La Guía para programadores de Java para Matisseo es un detallado y avanzado manual que muestra con ejemplos todas las posibilidades de estas tres herramientas: OQL, Java y Matisse.

4. CARACTERÍSTICAS DE LAS BASES DE DATOS OBJETO-RELACIONALES

El término "base de datos objeto-relacional" se usa para describir una base de datos que ha evolucionado desde el modelo relacional hasta una base de datos híbrida, que contiene la tecnología relacional y la orientada a objetos.

Durante muchos años se ha debatido sobre si las bases de datos orientadas a objetos son las sucesoras de las relacionales y sobre si la solución objeto-relacional es una alternativa de compromiso que tarde o temprano dará paso a la orientación a objetos pura.

Los partidarios de los sistemas objeto-relacionales esgrimen varias razones para demostrar que el modelo objetorelacional es la opción más adecuada. Estas se pueden resumir de la siguiente manera:

Las bases de datos objeto-relacionadas, tales como Oracla8i (y sus sucesoras), son compatibles con las bases de datos relacionales.

Los usuarios están muy familiarizados con los sistemas relacionales (MySQL, Oracle, Informix, Ms-SQLServer, PostgreSQL, etc.). Los usuarios pueden pasar sus aplicaciones actuales sobre bases de datos relaciones a modelo relacional sin tener que reescribirlas. Posteriormente se pueden ir adaptando las aplicaciones y bases de datos para que utilicen las funciones orientadas a objetos.

En cualquier caso, los SGBD-OR siempre serán una alternativa híbrida que intenta dar las ventajas de la OOpero sin abandonar el contrastado modelo relacional.

Un SGBD-OR se diferencia básicamente de un SGBDR en que define el tipo de datos Objeto. Una idea básica de los SGBDR es que el usuario pueda crear sus propios tipos de datos, para ser utilizados en aquella tecnología que permita la implementación de tipos de datos predefinidos. Además, los SGBDR permiten crear métodos para esos tipos de datos. Con ello se hace posible la creación de funciones miembro usando tipos de datos definidos por el usuario, lo que proporciona flexibilidad y seguridad.

Por tanto, un tipo de dato define una estructura y un comportamiento común para el conjunto de datos de una aplicación. Los usuarios pueden definir sus propios tipos de datos mediante dos categorías: tipos de objetos y colecciones.

Un tipo de objeto representa una entidad del mundo real y se compone de los siguientes elementos: Su nombre, que sirve para identificar el tipo de los objetos.

Sus atributos, que modelan la estructura y los valores de los datos de ese tipo. Cada atributo puede ser de un tipo de datos básico o de un tipo de usuario.

Sus métodos, que especifican el comportamiento. Son procedimientos y funciones escritos en un lenguaje de programación soportado por el SGBD-OR que se utilice. En el caso de utilizar Oracle, se podría hacer en PL/ SQL o Java.

Con un tipo de objeto se pueden proporcionarlos principios básicos de la OO (abstracción, encapsulación y herencia de tipos), con lo cual se puede aplicar la sobrecarga de operadores y la ligadura dinámica.

4.2. GESTIÓN DE OBJETOS SQL: ANSI SQL 1999

En esta sección se muestra cómo se pueden crear objetos en un SGBD-OR. En los ejemplos mostrados se utiliza la sintaxis de Oracle y las posibilidades que soporta para especificar los objetos y el acceso a los mismos. Sin embargo, también se hará una pequeña referencia a la sintaxis definida en el estándar SQL.

Para crear un objeto con Oracle se utiliza la sentencia CREATETYPE. El siguiente código muestra la creación de un objeto persona que tiene dos atributos: nombre, de tipo texto de longitud 30, y teléfono, de tipo texto de longitud 20.

Ejemplo

```
CREATE TYPE persona AS OBJECT
(
nombre VARCHAR2 (30), telefono VARCHIAR2 (20)
);
```

La sintaxis que se utilizaría en ANSI SQL.99 para definir objetos sería:

define type persona: tuple (nombre: string, telefono: string)

Además de objetos simples también se pueden crear objetos más complejos donde el tipo de datos de un atributo es un tipo objeto. El siguiente ejemplo muestra un tipo estudiante que tiene dos atributos: id estudiante, de tipo texto de longitud 9, y datos personales, que es de tipo persona.

CREATE TYPE estudiante NAS OBJECT (ide studiante varchar? (9), datos_personales persona) ;

Una vez definidos objetos simples o complejos, estos pueden utilizarse como tipos para los datos almacenados en tablas relacionales. Dos son las formas que Oracle ofrece para crear tablas que alberguen objetos:

Tablas de objetos: son tablas en las que cada objeto se almacena en una fila. Estas tablas facilitan el acceso a los atributos de los objetos ya que muestran cada atributo del objeto como si fueran columnas de esa tabla. Es importante destacar que una ventaja de crear tablas de objetos es que cada objeto conserva su identificador de objeto (OID) para poder hacer referencias a ese identificador desde otros objetos o tablas. En la Sección 3.3.5 el método ejecutarOQLO utiliza la función REF para obtener el identificador de un objeto y luego poder recuperarlo como una clase Java. Esta misma idea se podría hacer en Oracle con objetos recuperados de una tabla de objetos, ya que los objetos sí conservan el OID.

Un ejemplo para crear tablas de objeto podría ser una tabla de objetos persona llamada contactos que albergara los datos de las personas de contacto de una agenda en un teléfono móvil. La sintaxis en Oracle sería:

CREATE TABLE contactos OF persona;

Si sobre esta tabla se desea obtener el nombre y el teléfono de todas las personas cuyo nombre empiece por M, la consulta SQL necesaria sería:

SELECT C. nombre, c. telefono FROM COntaCtOS C Where c. nombre like “M:&” ;

Si se quiere insertar datos en la tabla contactos se podría hacer con la siguiente sintaxis:

INSERT INTO contactos WALUES ("Nieves Calama', '9675 70691');

Tabla con columnas de tipo objeto: son tablas en donde uno o más atributos son de un tipo objeto. De alguna manera estas tablas son las tablas típicas de un sistema relacional pero en donde una o más columnas son de un tipo objeto. En este caso, los objetos almacenados en las columnas no conservan su OID y, por tanto, no pueden ser referenciados por otras columnas u objetos, ni recuperar un objeto a través de su OID.

Si se desea crear una tabla con los estudiantes admitidos en una determinada actividad formativa, la sentencia necesaria en Oracle para crear una tabla con columnas de tipo objeto sería:

CREATE TABLE admitidos (fecha: date, solo estudia estudiante);

La tabla admitidos tiene una columna fecha de tipo date y un atributo admitido de tipo estudiante. Una consulta SQL que recupere el nombre y la id estudiante sería:

SELECT a. solo estudia.i.d estudiante, a. solo estudia. datos personales. nombre FROM admitidos a;

Si se desea insertar datos en la tabla admitidos, la sintaxis podría ser:

INSERT INTO admitidos WALUES (12/09/2012', estudiante ("98B", persona ("Nieves Calama", "" 96.757.0691)));

4.2.1. Referencias

Las relaciones entre objetos se establecen mediante columnas o atributos de tipo REF Con atributos de este tipo se pueden establecer relaciones uno a muchos entre objetos. En el caso de Oracle se asigna unidentificador único (OID) a cada objeto almacenado en una tabla de objetos. En otros objetos o tablas se pueden definir atributos o columnas de tipo REF que almacenen OID de los objetos con los que se quieran relacionar. En Oracle las relaciones pueden estar restringidas mediante la cláusula SCOPE o mediante una restricción de integridad referencial (REFERENTIAL). Cuando se restringe mediante SCOPE, todos los valores almacenados en la columna REF apuntan a objetos de la tabla indicada en la cláusula. Sin embargo, puede ocurrir que haya valores que apunten a objetos que no existan. La restricción mediante REFERENTIAL obliga a que las referencias sean siempre a objetos que existen en la tabla referenciada.

Por ejemplo, la siguiente tabla define Departamento, que alberga la estructura departamental de un centro educativo. Los atributos son el nombre del departamento y una referencia a la persona que lo dirige, que es de tipo Persona.

CREATE TABLE Departamento (Nom Dept. WARCHIAR (30), Jefe REIF persona) ;

En las consultas SQL se puede utilizar la función REFO) y DEREFO para devolver el OID de un objeto u obtener el valor de los objetos a los que apunta la referencia.

Métodos

Como es sabido, los métodos son funciones o procedimientos que se pueden declarar en la definición de un tipo de objeto para implementar el comportamiento que se desea para dicho tipo de objeto. Las aplicaciones llaman a los métodos para invocar su comportamiento. En Oracle los métodos son escritos en PL/SQL o en Java, y en este caso se almacenan en las bases de datos como procedimientos almacenados

(al igual que ocurre en Matisse como se vio en la Sección 3.3.1). Los métodos escritos en otros lenguajes se almacenan externamente.

El siguiente ejemplo muestra la creación de un tipo racional con numerador y denominador como atributos de tipo entero, y un método llamado normaliza. El código de los métodos no es incluido en la propia definición del tipo objeto sino que es posteriormente añadido con CREATETYPEBODY.

Ejemplo

```
CREATE TYPE racional AS OBJECT (
numerador INTEGER,
denominador INTEGER,
MEMBER PROCEDURE normaliza,
);
CREATE TYPE BODY racional AS MEMBER PROCEDURE normaliza TS g INTEGER; BEGIN g := gcd (num, den) ; — — num := num ll g; den : = den / g; END normaliza;
END;
```

Actividad

2.1. ¿Hay alguna diferencia significativa entre el modelo OO que sigue un lenguaje de programación como Java y el modelo OO propuesto por ODMG para bases de datos OO?

2.2. ¿Todos los elementos definidos en el modelo de datos ODMG se pueden encontrar en Java?

2.3. ¿Se podría decir que, con lo visto, Java es compatible al 100% con el modelo ODMG?

2.5. ¿Siguen las bases de datos relacionales la misma filosofía respecto a OML que sigue ODMG?

2.6. Pon algunos ejemplos de cómo se modifican atributos y tablas usando OML de un SGBDR.

2.7. ¿Es posible que la ausencia de un OML en los SGBD-OO diferencie mucho la modificación de objetos a como se hace en los SGBDR con OML?

2.8. Diseñar en OQL las siguientes consultas sobre el modelo dado en la figura del capítulo.

2.8.1. Obtener el título de todas las obras almacenadas

2.8.2. Obtener el título de todos los artículos cuyo autor tenga por nombre "Nikolai" y por apellido, "Gogol".

2.8.3. Obtener el título y revista de todos los artículos cuyo resultado de invocar al método dameNombreyApellidos() sea "Nikolai Gogol"

2.9. Instalar en local el sistema gestor Matisse. Una vez instalado y siguiendo los pasos descritos en esta sección crear una base de datos según la figura del capítulo.

2.10. Utilizar el código anterior para definir métodos que permitan crear los diferentes objetos Obra, Artículo, Libro y Autor, todos ellos de clases creadas según el modelo de biblioteca de la Figura del capítulo. A cada método de inserción se le debe pasar como parámetro los valores que serán asignados a los atributos de las clases. Por ejemplo, el método para crear un nuevo libro podría tener la siguiente signatura:

public Static void crea Libro (String titulo, String editorial, Int paginas)

2.10.1. Además de los métodos correspondientes para crear Libros, Obra, Artículo y Autor es necesario crear los métodos que relacionen objetos Autor con objetos de tipo Obra y viceversa.

2.10.2. Una vez creados varios objetos de diferente tipo, hay que comprobar desde el entorno de Matisse que están guardados correctamente.

2.11. Siguiendo los pasos mostrados, utilizar el editor de consultas de Matisse para probar las consultas creadas como ejemplo en la sección 2.1.5

2.12. Crea un método que permita recuperar objetos con una consulta OQL. Los parámetros que debe tener el método son:

2.12.1. La consulta OQL que recuperará un objeto,

2.12.2. El tipo de objeto que se recupera con la consulta. Este parámetro será necesario para saber qué tipo de objeto se debe crear para albergar el resultado de la consulta.

Índice