5.8. SQL:99 y bases de datos objeto-relacionales

En SQL:99, también conocido como SQL3, la cuarta revisión del estándar SQL, se introdujeron los tipos estructurados definidos por el usuario. Esto significa que los atributos de una tabla pueden tener tipos no atómicos o estructurados, además de tipos atómicos como `CHAR`, `VARCHAR2`, `INTEGER`, etc. En una tabla puede haber columnas que contengan objetos, *arrays* o incluso tablas (tablas anidadas), y también referencias a objetos, de manera que en varios lugares diferentes de la base de datos puedan existir referencias al mismo objeto, almacenado en un único lugar en la base de datos. Estas características, por supuesto, se salen completamente del modelo relacional. Pero si no se utilizan en una base de datos, esta sigue siendo relacional.

Las bases de datos que implementan, en mayor o menor medida, estas características se pueden calificar como objeto-relacionales. Entre ellas cabe destacar Oracle.

Oracle implementa todas estas características en una capa de *software* sobre una base de datos relacional.

Recursos web

www

Las últimas versiones de la base de datos Oracle son 11g, 12c y 18c. Existen versiones XE de 11g y 18c. Son versiones simplificadas y limitadas que pueden utilizarse de manera gratuita y sin soporte técnico de Oracle. Para un primer contacto puede ser buena opción la versión 11g XE. El siguiente enlace dirige a la página de descargas para las distintas versiones de Oracle:

https://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html

El siguiente enlace proporciona la documentación más importante de la versión 12c:

https://docs.oracle.com/en/database/oracle/oracle-database/12.2/books.html

Es de especial interés la documentación de SQL, una vez que se haya completado el proceso de instalación y configuración inicial:

http://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlqr/sql-language-quick-reference.pdf
http://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/sql-language-reference.pdf

5.9. Características objeto-relacionales de Oracle

Oracle permite definir tipos de objetos (clases), tablas de objetos (que almacenan un objeto de una clase determinada en cada fila), tablas con columnas de objetos (que contienen objetos de una clase determinada en una columna), y tablas con columnas que contienen colecciones, tanto *arrays* como tablas (tablas anidadas). Pero en última instancia se trata de una base de datos relacional y todos estos datos se almacenan en una base de datos puramente relacional.

Recurso digital

En el anexo web 5.4, disponible en www.sintesis.com y accesible con el código indicado en la primera página del libro, encontrarás la "Guía del desarrollador objeto-relacional".

A continuación, se explicará la forma de definir, utilizando el SQL de Oracle, tipos estructurados definidos por el usuario, tales como los introducidos en SQL:99, entre ellos tipos de objetos, y se utilizarán para implementar el modelo de objetos de ejemplo que se ha venido utilizando hasta ahora y para introducir algunos datos de ejemplo.

5.9.1. Tipos de objetos

Se crean con `CREATE TYPE … AS OBJECT`. Un tipo de objeto es un tipo estructurado que tiene atributos y métodos, es decir, una clase. Hay que terminar las declaraciones de tipo con el carácter /. Se pueden crear tipos de objeto, que se utilizarán para la definición de las tablas, como sigue.

```
CREATE TYPE DatosProfesionales AS OBJECT (
    categoria CHAR(2),
    sueldo_bruto_anual NUMBER(8, 2)
);
/
CREATE TYPE Empleado AS OBJECT (
    dni CHAR(9),
    nom_emp VARCHAR2(60),
    datos_prof DatosProfesionales
    ) NOT FINAL;
/
```

El propósito de la opción `NOT FINAL` en la definición de `Empleado` se verá en breve.

En estas definiciones de tipos no tienen sentido restricciones sobre el valor que puedan tomar los atributos, como por ejemplo que no puedan tomar valor nulo. Estas se introducen como restricciones de integridad en tablas basadas en estos tipos, como se verá en breve.

5.9.2. Herencia

Se pueden definir clases como subclases de otras. No se admite herencia múltiple, solo simple. Solo se pueden definir subclases de una clase dada si se define como `NOT FINAL`. Se puede definir la clase `EmpleadoPlantilla` como subclase de `Empleado` como sigue:

```
CREATE TYPE EmpleadoPlantilla UNDER Empleado(
    num_emp CHAR(12)
);
/
```

Si no se hubiera definido la clase Empleado como `NOT FINAL`, se podría cambiar con:

```
ALTER TYPE Empleado NOT FINAL;
```

5.9.3. Objetos de fila y objetos de columna

Se puede definir una tabla con la misma estructura de una clase (es decir, de un *object type* o tipo de objeto). Cada fila de la tabla contendrá un objeto de la clase, lo que se conoce como *row object* u objeto de fila. La base de datos genera un OID o identificador único para cada objeto de fila.

Un atributo de una tabla puede ser de un tipo de objeto. Esto es lo que se conoce como *object column* u objeto de columna. La base de datos no genera OID para objetos de columna.

Por ejemplo, se puede definir una tabla `empleados` cuyas filas contengan objetos del tipo de objeto `Empleado`, cuyo atributo `datos_prof` es del tipo `DatosProfesionales`, por lo que cada uno de estos objetos de fila contendrá un objeto de columna en `datos_prof`.

En la definición de la tabla se puede incluir la definición de clave primaria y restricciones sobre los valores que puedan tomar los atributos.

```
CREATE TABLE empleados OF Empleado(
    dni PRIMARY KEY,
    nom_emp NOT NULL
);
```

Se pueden insertar filas en la tabla de empleados de la siguiente forma:

```
INSERT INTO empleados VALUES (Empleado('56789012B', 'SAMPER', null));
INSERT INTO empleados VALUES (Empleado('76543210S', 'SILVA',
   DatosProfesionales('B1', 45200.00)));
INSERT INTO empleados VALUES (EmpleadoPlantilla('78901234X', 'NADALES', null,
   '604202'));
```

El ejemplo anterior muestra cómo en la tabla `empleados` se pueden introducir objetos de la clase `Empleado` y de su subclase `EmpleadoPlantilla`.

El operador `REF` permite obtener referencias a objetos de fila, es decir, sus OID.

```
SELECT REF(e) FROM empleados e;
```

Para objetos de columna no se crean referencias. La siguiente consulta es errónea:

```
SELECT REF(e.datos_prof) FROM empleados e;
```

5.9.4. Tipos de objetos con referencias a otros tipos de objetos

Las referencias permiten utilizar un mismo objeto en la definición de varios objetos. Por ejemplo: un mismo empleado de plantilla puede ser el jefe de proyecto de varios proyectos. Para reflejar esto en la base de datos, un objeto de tipo `EmpleadoPlantilla` se almacena una sola vez en la base de datos, en la tabla `empleados`, y en la tabla `proyectos` se almacenan varios objetos de tipo `Proyecto` que contienen una referencia a él. Guardar copias del mismo objeto de tipo `EmpleadoPlantilla` en varios objetos de tipo `Proyecto` no solo es ineficiente, sino erróneo. Una copia de un objeto es igual, pero no idéntica, no es el mismo objeto. Una referencia contiene el OID del objeto referenciado. El tipo `Proyecto` y la tabla `proyectos` se podrían definir de esta manera:

```
CREATE TYPE Proyecto AS OBJECT (
    nom_proy VARCHAR2(32),
    f_inicio DATE,
    f_fin DATE,
    jefe_proy REF EmpleadoPlantilla
);
/
CREATE TABLE proyectos OF Proyecto(
    nom_proy NOT NULL,
    f_inicio NOT NULL,
    jefe_proy SCOPE IS empleados
);
```

El jefe de proyecto de un proyecto se almacena como una referencia (`REF`) a un objeto de tipo `EmpleadoPlantilla` almacenado en la tabla empleados (`SCOPE IS empleados`).

Ahora se puede introducir un proyecto, incluyendo la referencia a su jefe de proyecto.

```
INSERT INTO proyectos
SELECT Proyecto('PAPEL ELECTRÓNICO', to_date('01/12/2018','dd/mm/YYYY'),
    NULL, TREAT(REF(e) AS REF EmpleadoPlantilla)) FROM empleados e WHERE
    DNI='78901234X';
```

El operador `TREAT` hace posible tratar la referencia recuperada como una referencia a un objeto de tipo `EmpleadoPlantilla` en lugar de como un objeto de su superclase `Empleado`.

Se puede obtener un objeto a partir de una referencia a él mediante el operador `DEREF`.

```
SELECT DEREF (p.jefe_proy)
FROM proyectos p WHERE p.nom_proy='PAPEL ELECTRÓNICO';
```

5.9.5. Tipos de datos de colección: VARRAY y tablas anidadas

Aparte de tipos de objeto, es posible definir distintos tipos de colecciones, y se pueden definir atributos con esos tipos. En particular, se pueden definir tipos de *arrays* y de tablas.

Los tipos de *array* (`VARRAY`) son similares a los *arrays* de Java. Los elementos guardados en un `VARRAY` se pueden recuperar indicando un índice, siendo 1 el índice del primer elemento. Por

ejemplo, para cada empleado se podría guardar hasta un máximo de tres teléfonos, modificando la definición del tipo `Empleado` como sigue:

```
CREATE TYPE TelefonosEmpleado AS VARRAY(2) OF CHAR(14);
/
ALTER TYPE Empleado ADD ATTRIBUTE telefonos TelefonosEmpleado CASCADE;
```

La palabra reservada `CASCADE` se utiliza para que los cambios realizados en el tipo se propaguen a todos los tipos y tablas en los que este se utiliza. Esto implica, aparte del cambio de los tipos, la actualización de los contenidos de las tablas para adaptar los datos existentes a la nueva definición de los tipos.

Se podría añadir un nuevo empleado en la tabla de objetos `empleados` definida anteriormente de la siguiente forma:

```
INSERT INTO empleados VALUES( Empleado('89012345E', 'ROJAS', null,
    TelefonosEmpleado('654321098','959456789'))
);
```

Se pueden definir tipos de tablas. Si se define una columna de una tabla con un tipo de tabla, cada fila de la tabla contendrá en esa columna una tabla (tabla anidada o *nested table*). Por ejemplo, se podría añadir a la tabla `proyectos` una columna para la lista de empleados asignados al proyecto.

```
CREATE TYPE TablaAsigEmpleados AS TABLE OF REF Empleado;
/
ALTER TYPE Proyecto ADD ATTRIBUTE asig_empleados TablaAsigEmpleados CASCADE;
```

Se puede añadir un nuevo proyecto con la lista de empleados asignados de la siguiente manera:

```
UPDATE proyectos
SET asig_empleados=TablaAsigEmpleados
(
(SELECT REF(e) FROM empleados e WHERE DNI='56789012B'),
(SELECT REF(e) FROM empleados e WHERE DNI='76543210S')
)
WHERE nom_proy='PAPEL ELECTRÓNICO';
```

Actividad propuesta 5.8

Se quiere tener para cada asignación de un empleado a un proyecto la fecha de inicio y de fin. Elimina la columna `asig_empleados` de la tabla `proyectos` y añade una nueva columna con igual nombre que contenga también una tabla anidada con los empleados asignados al proyecto, pero en cada fila de esta tabla, además de la referencia al empleado, debe haber columnas con las fechas de inicio y de fin de asignación al proyecto (`f_ini` y `f_fin`). La referencia al empleado no puede ser nula, ni la fecha de inicio. Su clave primaria debe estar compuesta por la referencia

al empleado y la fecha de inicio. Se trata de definir un nuevo tipo `AsigEmpleado` que incluya la referencia al empleado y las dos fechas, y utilizarlo para definir el tipo `TablaAsigEmpleados`.

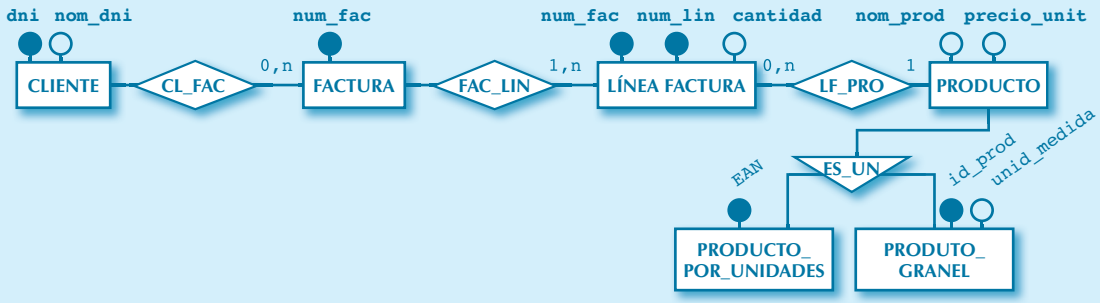
Las características objeto-relacionales de Oracle ofrecen muchas posibilidades. Con esta breve introducción se ha tratado solo de introducir las características fundamentales de las BDOR basadas en SQL:99, y de dar una idea de cómo se puede crear con Oracle un esquema objeto-relacional con diversos tipos de objetos y diversos tipos de relaciones entre ellos.

Resumen

- Como solución al desfase objeto-relacional, es decir, al conjunto de dificultades que plantea la persistencia de objetos en bases de datos relacionales, han surgido diversas soluciones. Entre ellas, las BDO, las BDOR y la correspondencia objeto-relacional.
- Las BDO permiten almacenar directamente objetos.
- La organización ODMG desarrolló estándares para BDO, entre los que están ODL y OQL. No se incluye entre ellos ningún lenguaje para manipulación de objetos, sino *language bindings* para lenguajes de programación orientados a objetos –entre ellos Java–, que hacen posible la persistencia transparente. Es decir, se gestionan de igual manera los objetos persistentes y transitorios, y en el momento de confirmar una transacción se reflejan los cambios realizados sobre los objetos en la base de datos.
- ODL permite especificar los atributos de los objetos, sus métodos y las relaciones entre objetos, que pueden ser de tipo uno a uno, uno a mucho o muchos a muchos.
- En SQL:99 se introdujeron tipos estructurados definidos por el usuario, entre los que están los tipos objetos (clases), además de tipos de colecciones, tablas anidadas y referencias.
- Las BDOR, como Oracle, se basan en SQL:99 y amplían la funcionalidad de una base de datos relacional con tipos estructurados definidos por el usuario, entre los que están los objetos.

Ejercicios propuestos

1. En el siguiente diagrama E-R se representan las relaciones entre clientes, facturas y productos incluidos en las facturas. Cada línea de factura viene identificada por la línea de factura y un número secuencial dentro de la propia factura. Interesa guardar ambos, dado que aparecen en las facturas impresas y el orden de las líneas es relevante. Los productos son de dos tipos: los que se venden por unidades, identificados por su EAN, y los que se venden a granel (por kilos, litros, etc.), identificados por un código que puede contener letras y números. Para estos últimos interesa guardar la unidad de medida a la que se aplica el precio unitario. Crea un esquema de objetos para él, la definición del esquema de objetos en ODL de Matisse, e importa el esquema en una nueva base de datos en Matisse.



2. Crea un programa en Java que cree varios clientes y productos y varias facturas para varios clientes, cada una de ellas con varias líneas. Debe haber alguna factura que contenga líneas para productos tanto por unidades como a granel. Será necesario generar previamente las *stub classes*.
3. Añade un método a la clase `Factura` que devuelva su importe neto, que será el resultado de sumar el importe para cada línea, que a su vez será el resultado de multiplicar el precio unitario del producto por la cantidad. Escribe un programa que lo utilice para mostrar todos los datos de todas las facturas, incluyendo el importe neto.
4. Escribe un programa que borre una factura, que debe recuperar a partir de su número de factura. Antes de borrar la factura, debe borrar sus líneas.
5. Cambia el método `deepRemove()` de la clase `Factura` para que borre una factura, borrando previamente todas sus líneas y, por último, la propia factura.
6. Escribe un programa que, utilizando el `driver` JDBC de Matisse, obtenga todas las facturas de un cliente identificado por su DNI, y muestre para cada una su número de factura y su importe total, que se obtendrá con el método desarrollado en una actividad anterior.
7. Crea un esquema objeto-relacional en Oracle para el diagrama E-R anterior. Las relaciones de uno a muchos se implementarán mediante referencias a objetos en lugar de mediante claves foráneas. Las líneas de factura se almacenarán en una tabla anidada dentro de la propia tabla de facturas, y en cada línea de factura habrá una referencia al producto.
8. Crea para Oracle, mediante sentencias de SQL, los mismos datos que se crearon en actividades anteriores para Matisse.

   Borra en Oracle, mediante sentencias de SQL, la misma o las mismas facturas que se borraron en una actividad anterior mediante un programa en Java en Matisse.

ACTIVIDADES DE AUTOEVALUACIÓN

1. El objeto del estándar ODMG 3.0 es:
   - ☐ *a)* Definir las características que debe cumplir una BDO.
   - ☐ *b)* La persistencia de objetos utilizados en lenguajes orientados a objetos.
   - ☐ *c)* La persistencia de objetos en BDO u objeto-relacionales.
   - ☐ *d)* Ninguna de las opciones anteriores es correcta.

2. En SQL:99 se introdujeron:
   - ☐ *a)* Tipos de objetos definidos por el usuario que son clases e interfaces.
   - ☐ *b)* Mecanismos para facilitar la transición de las actuales bases de datos relacionales a futuras BDO.
   - ☐ *c)* Tipos estructurados definidos por el usuario, entre ellos objetos, arrays y tablas anidadas.
   - ☐ *d)* Mecanismos para implementar la persistencia de objetos en bases de datos relacionales mediante el establecimiento automático de correspondencias objeto-relacionales.

3. Un *languaje binding* tal como los especificados en ODMG 3.0:
   - ☐ *a)* Interactúa con un DML con unas funcionalidades mínimas.
   - ☐ *b)* Permite gestionar de la misma forma objetos persistentes y transitorios, utilizando los mecanismos ya existentes en un lenguaje de programación.
   - ☐ *c)* Hace innecesario un DML, pero requiere determinadas extensiones en la sintaxis de un lenguaje orientado a objetos de propósito general.
   - ☐ *d)* No garantiza la completitud computacional propuesta en el manifiesto de Atkinson y otros en 1989.

4. Un *traversal path* en ODL:
   - ☐ *a)* Solo se define para relaciones de uno a muchos o de muchos a muchos.
   - ☐ *b)* Se define en extremos de una relación con multiplicidad mayor que 1.
   - ☐ *c)* Se definen en ambos extremos de una relación binaria.
   - ☐ *d)* Se implementan siempre mediante colecciones, ordenadas o desordenadas.

5. El lenguaje OQL:
   - ☐ *a)* Tiene las sentencias UPDATE y DELETE de SQL, adaptadas para objetos.
   - ☐ *b)* Permite hacer consultas sobre objetos persistentes almacenados en una BDO.
   - ☐ *c)* Solo se puede utilizar desde un *language binding*.
   - ☐ *d)* Es completamente independiente de los *language binding*, porque estos solo sirven para crear, modificar y borrar objetos, pero no para consultarlos.

6. Una implementación del *language binding* de Java de ODMG 3.0 garantiza que, al realizarse una operación commit para confirmar una transacción:
   - ☐ *a)* Se graben todos los cambios realizados en objetos persistentes, pero se ignoren los cambios realizados en cualquier objeto transitorio.
   - ☐ *b)* Se graben los cambios realizados sobre cualquier objeto persistente modificado, y también sobre cualquier objeto transitorio alcanzable siguiendo las referencias desde ellos.
   - ☐ *c)* Si siguiendo las referencias desde cualquier objeto persistente modificado se alcanza un objeto no persistente, se produzca un error y se deshagan todos los cambios realizados.
   - ☐ *d)* Se graben todos los cambios realizados durante ella, tanto en objetos persistentes como transitorios, y todos estos últimos se convierten en persistentes.

7. La base de datos Matisse:
   - ☐ *a)* Es una base de datos posrelacional, y solo hasta cierto punto una BDO.

- ☐ *b)* Es en última instancia una base de datos relacional sobre la que se implementan los objetos en una capa de *software*.
- ☐ *c)* Es una BDO que tiene su propio *driver* JDBC, pero en una transacción no se pueden mezclar cambios hechos con JDBC y con los métodos de las *stub classes* del Java *binding*.
- ☐ *d)* Es una BDO que tiene además un *driver* JDBC.

8. El lenguaje SQL de Matisse:
   - ☐ *a)* Proporciona una implementación completa de SQL:99.
   - ☐ *b)* Solo proporciona operaciones de consulta, dado que las operaciones de creación, modificación y borrado se realizan con el Java *binding*.
   - ☐ *c)* No es realmente SQL, aunque sí es similar a SQL en muchos aspectos.
   - ☐ *d)* No es SQL, pero incluye una implementación de OQL y, al igual que OQL, es similar a SQL.

9. Oracle 12c:
   - ☐ *a)* Es una BDOR, es decir, es a la vez una BDO y una base de datos relacional.
   - ☐ *b)* Es una base de datos relacional sobre la que se implementan objetos en una capa *de soft*ware.
   - ☐ *c)* Es conforme a ODMG 3.0.
   - ☐ *d)* Todo lo anterior es cierto.

10. La base de datos Oracle:
    - ☐ *a)* Permite tablas de objetos y columnas de objetos.
    - ☐ *b)* Permite tablas de objetos, pero no columnas de objetos, aunque sí columnas que contengan referencias a objetos.
    - ☐ *c)* Permite tablas de objetos siempre que no contengan referencias a otros objetos.
    - ☐ *d)* Permite tablas anidadas, siempre que no sean tablas de objetos.

SOLUCIONES:

1. c	**5.** b	**9.** b	
2. c	**6.** b	**10.** a	
3. b	**7.** d		
4. c	**8.** c		