

UNIDAD 7: USO DE BASES DE DATOS OBJETO-RELACIONALES

Módulo Profesional: Bases de Datos

Índice

RESUMEN INTRODUCTORIO	3
INTRODUCCIÓN	3
CASO INTRODUCTORIO	3
1. TECNOLOGÍA BDOR DE ORACLE	4
2. TIPOS DE DATOS DEFINIDOS POR EL USUARIO	6
2.1 Tipos de objetos	6
2.2 Tablas de objetos	10
2.3 Referencia entre objetos	12
2.4 Tipos de datos colección	13
2.5 Inserción y acceso a los datos	16
RESUMEN FINAL	34

RESUMEN INTRODUCTORIO

En la presente unidad se estudiarán conceptos con las bases de datos de tipo objeto relacional. Es importante destacar la presencia de la orientación a objetos en este tipo de bases de datos. Fundamentalmente, la unidad pretende adentrar al alumno información sobre la tecnología BDOR de Oracle y los tipos de datos que pueden ser definidos por el usuario de la base de datos como, por ejemplo, los tipos y las tablas de objetos, las referencias entre los diferentes objetos, los tipos de datos colección y su utilidad, así como algunas operaciones con ellos, como la inserción y el acceso a los datos.

INTRODUCCIÓN

Las unidades anteriores se han limitado a estudiar un tipo de bases de datos llamadas bases de datos relacionales. Existe un nuevo modelo de bases de datos, las objeto-relacionales, que pueden aportar muchas ventajas, entre las que se pueden destacar: el aprovechamiento más óptimo de la reutilización de código, el trabajo colaborativo y la interconexión de las bases de datos con otros lenguajes de programación orientados a objetos.

CASO INTRODUCTORIO

Paula mantiene una gran relación con todos y cada uno de los clientes que han acudido a su empresa y que han realizado peticiones de desarrollo de software. Uno de ellos, cuya actividad está íntimamente relacionada con el padrón municipal de un ayuntamiento, necesita revisar su infraestructura de bases de datos, ya que la actual funciona únicamente sobre tecnología relacional. Se han propuesto poder dar mayor flexibilidad y seguridad a toda la información almacenada. Otro de los requisitos es poder crear los propios tipos de datos que se van a almacenar y, además, implementar una serie de métodos que actúen directamente sobre ellos. Por ejemplo, la creación de un tipo de dato llamado "Persona", que recopile los diferentes datos que sea necesario almacenar en la base de datos. Paula no duda un instante, y propone a su cliente retomar contactos para estudiar la viabilidad de los cambios presentados. No le cabe duda de que la mejor de las soluciones es migrar la base de datos actual a una de tipo objeto relacional.

Al finalizar la unidad el alumnado será capaz de darle funcionalidad orientada a objetos a una base de datos relacional, conocerá los tipos de datos objeto en Oracle y será capaz de almacenar objetos en memoria gracias a las colecciones de objetos.

1. TECNOLOGÍA BDOR DE ORACLE

El término Base de Datos Objeto Relacional (BDOR) se usa para describir una base de datos que ha evolucionado desde el modelo relacional hacia otra más amplia que incorpora conceptos del paradigma orientado a objetos. Por tanto, un Sistema de Gestión Objeto-Relacional (SGBDOR) contiene ambas tecnologías: relacional y de objetos.

Una idea básica de las BDOR 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, las BDOR permiten crear métodos para esos tipos de datos. Con ello, este tipo de SGBD hace posible la creación de funciones miembro usando tipos de datos definidos por el usuario, lo que proporciona flexibilidad y seguridad.

Los SGBDOR permiten importantes mejoras en muchos aspectos con respecto a las BDR tradicionales. Estos sistemas gestionan tipos de datos complejos con un esfuerzo mínimo y albergan parte de la aplicación en el servidor de base de datos. Permiten almacenar datos complejos de una aplicación dentro de la BDOR sin necesidad de forzar los tipos de datos tradicionales. Son compatibles en sentido ascendente con las bases de datos relacionales tradicionales, tan familiares a multitud de usuarios. Es decir, se pueden pasar las aplicaciones sobre bases de datos relacionales al nuevo modelo sin tener que rescribirlas. Adicionalmente, se pueden ir adaptando las aplicaciones y bases de datos para que utilicen las funciones orientadas a objetos.



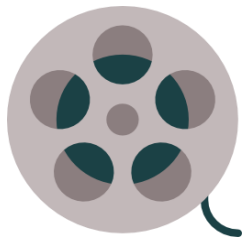
PARA SABER MÁS

Para ampliar información sobre la definición de las bases de datos objeto-relacionales:

http://es.wikipedia.org/wiki/Base_de_datos_objeto-relacional

Debido a los requerimientos de las nuevas aplicaciones, el sistema de gestión de bases de datos relacionales de Oracle, desde versión 8.i, ha sido significativamente extendido con conceptos del modelo de bases de datos orientadas a objetos. De esta manera, aunque las estructuras de datos que se utilizan para almacenar la información siguen siendo tablas, los usuarios pueden utilizar muchos de los mecanismos de orientación a objetos para definir y acceder a los datos.

Oracle proporciona mecanismos para que el usuario pueda definir sus propios tipos de datos, cuya estructura puede ser compleja, y se permite la asignación de un tipo complejo (dominio complejo) a una columna de una tabla. Además, se reconoce el concepto de objetos, de tal manera que un objeto tiene un tipo, se almacena en cierta fila de cierta tabla y tiene un identificador único (OID). Estos identificadores se pueden utilizar para referenciar a otros objetos y así representar relaciones de asociación y de agregación. Oracle también proporciona mecanismos para asociar métodos a tipos, y constructores para diseñar tipos de datos multivaluados (colecciones) y tablas anidadas.



VIDEO DE INTERÉS

Se recomienda visualizar el video contenido en el siguiente enlace donde se explican las diferencias entre el modelo objeto-relacional y el relacional:

<https://www.youtube.com/watch?v=yTI1TEiCJyI>

2. TIPOS DE DATOS DEFINIDOS POR EL USUARIO

Un tipo de dato define una estructura y un comportamiento común para un conjunto de datos de las aplicaciones. Los usuarios de Oracle pueden definir sus propios tipos de datos mediante dos categorías: tipos de objetos (object type) y tipos para colecciones (collection type).

Para construir los tipos de usuario se utilizan los tipos básicos provistos por el sistema y otros tipos de usuario previamente definidos.

2.1 Tipos de objetos

Representan una entidad del mundo real y se caracterizan por:

- Su nombre que sirve para identificar el tipo de los objetos.
- Sus atributos modelan la estructura junto con los valores de los datos de ese tipo. Cada atributo puede ser de un tipo básico o de usuario.
- Sus métodos (procedimientos o funciones) escritos en lenguaje PL/SQL (almacenados en la BDOR), o escritos en C (almacenados externamente).

Los tipos de objetos actúan como plantillas para los objetos de cada tipo. A continuación, se ve un ejemplo de cómo definir el tipo de dato Direccion_T en el lenguaje de definición de datos de Oracle, y cómo utilizar este tipo de dato para definir el tipo de dato de los objetos de la clase de Cliente_T.

DEFINICIÓN ORIENTADA A OBJETOS

```
define type Direccion_T:
tuple [calle:string,
        ciudad:string,
        prov:string,
        codpos:string]

define class Cliente_T
type tuple [clinum: integer,
            clinomb:string,
            direccion:Direccion_T,
            telefono: string,
            fecha-nac:date]
operations edad():integer
```

DEFINICIÓN EN ORACLE

```
CREATE TYPE direccion_t AS OBJECT (
    calle VARCHAR2(200),
    ciudad VARCHAR2(200),
    prov CHAR(2),
    codpos VARCHAR2(20));

CREATE TYPE cliente_t AS OBJECT (
    clinum NUMBER,
    clinomb VARCHAR2(200),
    direccion direccion_t,
    telefono VARCHAR2(20),
    fecha_nac DATE,
    MEMBER FUNCTION edad RETURN NUMBER,
    PRAGMA
    RESTRICT_REFERENCES(edad,WNDS));
```

Resultado tras la ejecución de la consulta



PARA SABER MÁS

Información sobre la sintaxis de la instrucción CREATE TYPE:

https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_8001.htm

Métodos: La especificación de un método se hace junto a la creación de su tipo, y debe llevar siempre asociada una directiva de compilación (PRAGMA RESTRICT_REFERENCES), para evitar que los métodos manipulen la base de datos o las variables del paquete PL/SQL. Tienen el siguiente significado:

- **WNDS:** no se permite al método modificar las tablas de la base de datos.
- **WNPS:** no se permite al método modificar las variables de PL/SQL.
- **RNDS:** no se permite al método leer las tablas de la base de datos.
- **RNPS:** no se permite al método leer las variables del paquete PL/SQL.

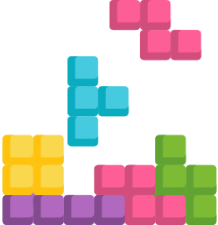


EJEMPLO PRÁCTICO

Los métodos se pueden ejecutar sobre los objetos de su mismo tipo. Si **x** es una variable PL/SQL que almacena objetos del tipo **Cliente_T**, entonces **x.edad()** calcula la edad del cliente almacenado en **x**. La definición del cuerpo de un método en PL/SQL se hace de la siguiente manera:

```
CREATE OR REPLACE TYPE BODY cliente_t AS
  MEMBER FUNCTION edad RETURN NUMBER IS
    a NUMBER;
    d DATE;
  BEGIN
    d:= today();
    a:= d.año - fecha_nac.año;
    IF (d.mes < fecha_nac.mes) OR
      ((d.mes = fecha_nac.mes) AND (d.dia < fecha_nac.dia))
    THEN a:= a-1;
    END IF;
    RETURN a;
  END;
END;
```

Constructores: En Oracle, todos los tipos de objetos tienen asociado por defecto un método que construye nuevos objetos de ese tipo de acuerdo a la especificación del tipo. El nombre del método coincide con el nombre del tipo, y sus parámetros son los atributos del tipo.



EJEMPLO PRÁCTICO

Las siguientes expresiones construyen dos objetos con todos sus valores:

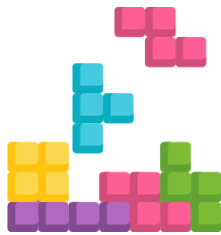
```
direccion_t('Avenida Sagunto', 'Puzol', 'Valencia', 'E-23523')  
cliente_t( 2347,  
  'Juan Pérez Ruíz',  
  direccion_t('Calle Eo', 'Onda', 'Castellón',  
    '34568'),  
  '696-779789',  
  12/12/1981)
```

Métodos de comparación: Para comparar los objetos de cierto tipo es necesario indicar a Oracle cuál es el criterio de comparación. Para ello, hay que escoger entre un método MAP u ORDER, debiéndose definir al menos uno de estos métodos por cada tipo de objeto que necesite ser comparado. La diferencia entre ambos es la siguiente:

Un método **MAP** sirve para indicar cuál de los atributos del tipo se utilizará para ordenar los objetos del tipo, y por tanto se puede utilizar para comparar los objetos de ese tipo por medio de los operadores de comparación aritméticos (<, >).

Un método **ORDER** utiliza los atributos del objeto sobre el que se ejecuta para realizar un cálculo y compararlo con otro objeto del mismo tipo que toma como argumento de entrada. Este método devolverá un valor negativo si el parámetro de entrada es mayor que el atributo, un valor positivo si ocurre lo contrario y un cero si ambos son iguales. El siguiente ejemplo define un orden para el tipo cliente_t diferente al anterior.

Sólo una de estas definiciones puede ser válida en un tiempo dado.



EJEMPLO PRÁCTICO

La siguiente declaración permite decir que los objetos del tipo **cliente_t** se van a comparar por su atributo clinum:

```
CREATE TYPE cliente_t AS OBJECT (  
    clinum NUMBER,  
    clinomb VARCHAR2(200),  
    direccion direccion_t,  
    telefono VARCHAR2(20),  
    fecha_nac DATE,  
    MAP MEMBER FUNCTION ret_value RETURN NUMBER,  
    PRAGMA RESTRICT_REFERENCES(ret_value, WNDS, WNPS,  
    RNPS, RNDS), /  
    *instrucciones a PL/SQL*/  
    MEMBER FUNCTION edad RETURN NUMBER,  
    PRAGMA RESTRICT_REFERENCES(edad, WNDS));  
CREATE OR REPLACE TYPE BODY cliente_t AS  
    MAP MEMBER FUNCTION ret_value RETURN NUMBER IS  
    BEGIN  
        RETURN clinum  
    END;  
END;
```

```
CREATE TYPE cliente_t AS OBJECT (  
    clinum NUMBER,  
    clinomb VARCHAR2(200),  
    direccion direccion_t,  
    telefono VARCHAR2(20),  
    fecha_nac DATE,  
    ORDER MEMBER FUNCTION  
    cli_ordenados (x IN clientes_t) RETURN INTEGER,  
    PRAGMA RESTRICT_REFERENCES(  
    cli_ordenados, WNDS, WNPS, RNPS, RNDS),  
    MEMBER FUNCTION edad RETURN NUMBER,  
    PRAGMA RESTRICT_REFERENCES(edad, WNDS));  
CREATE OR REPLACE TYPE BODY cliente_t AS  
    ORDER MEMBER FUNCTION cli_ordenados (x IN cliente_t)  
        RETURN INTEGER IS  
    BEGIN  
        RETURN clinum - x.clinum; /*la resta de los dos números  
    clinum*/  
    END;  
END;
```

Si un tipo de objeto no tiene definido ninguno de estos métodos, Oracle es incapaz de deducir cuándo un objeto es mayor o menor que otro. Sin embargo, sí puede determinar cuándo dos objetos del mismo tipo son iguales. Para ello, el sistema compara el valor de los atributos de los objetos uno a uno:

- Si todos los atributos son no nulos e iguales, Oracle indica que ambos objetos son iguales.
- Si alguno de los atributos no nulos es distinto en los dos objetos, entonces Oracle dice que son diferentes.
- En otro caso, Oracle dice que no puede comparar ambos objetos.

2.2 Tablas de objetos

Una vez definidos los tipos, éstos pueden utilizarse para definir nuevos tipos, tablas que almacenen objetos de esos tipos, o para definir el tipo de los atributos de una tabla. Una tabla de objetos es una clase especial de tabla que almacena un objeto en cada fila y que facilita el acceso a los atributos de esos objetos como si fueran columnas de la tabla.

La diferencia entre la primera y la segunda tabla es que la primera almacena objetos con su propia identidad (OID) y la segunda no es una tabla de objetos, sino una tabla con una columna con un tipo de datos de objeto. Es decir, la segunda tabla tiene una columna con un tipo de datos complejo, pero sin identidad de objeto.



EJEMPLO PRÁCTICO

Se puede definir una tabla para almacenar los clientes de este año y otra para almacenar los de años anteriores de la siguiente manera:

```
CREATE TABLE clientes_año_tab OF cliente_t  
(clinum PRIMARY KEY);  
CREATE TABLE clientes_antiguos_tab (  
año NUMBER,  
cliente cliente_t);
```

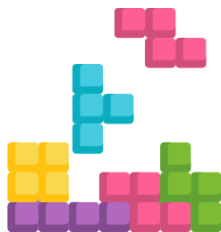
Además de esto, Oracle permite considerar una tabla de objetos desde dos puntos de vista:

Como una tabla con una sola columna cuyo tipo es el de un tipo de objetos.

Como una tabla que tiene tantas columnas como atributos los objetos que almacena.

Por ejemplo, se puede ejecutar una de las dos instrucciones siguientes. En la primera instrucción, la tabla `clientes_año_tab` se considera como una tabla con varias columnas cuyos valores son los especificados. En el segundo caso, se la considera como con una tabla de objetos que en cada fila almacena un objeto.

Las reglas de integridad, de clave primaria, y el resto de propiedades que se definan sobre una tabla, sólo afectan a los objetos de esa tabla, es decir no se refieren a todos los objetos del tipo asignado a la tabla.



EJEMPLO PRÁCTICO

En esta instrucción la cláusula **VALUE** permite visualizar el valor de un objeto:

```
INSERT INTO clientes_año_tab VALUES(
2347,
'Juan Pérez Ruíz',
direccion_t('Calle Castalia', 'Onda', 'Castellón', '34568'),
'696-779789',
12/12/1981);
SELECT VALUE(c) FROM clientes_año_tab c
WHERE c.clinomb = 'Juan Pérez Ruíz';
```



PARA SABER MÁS

En esta web se puede ampliar información sobre las bases de datos objeto-relacionales con algunos ejemplos de creación de tablas de objetos:

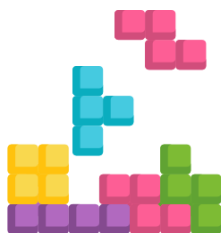
http://www3.uji.es/~mmarques/e16/teoria/lib_cap9.pdf

2.3 Referencia entre objetos

Los identificadores únicos asignados por Oracle a los objetos que se almacenan en una tabla, permiten que éstos puedan ser referenciados desde los atributos de otros objetos o desde las columnas de tablas.

El tipo de datos proporcionado por Oracle para soportar esta facilidad se denomina REF. Un atributo de tipo **REF** almacena una referencia a un objeto del tipo definido, e implementa una relación de asociación entre los dos tipos de objetos. Estas referencias se pueden utilizar para acceder a los objetos referenciados y para modificarlos; sin embargo, no es posible operar sobre ellas directamente. Para asignar o actualizar una referencia se debe utilizar siempre **REF** o **NULL**. Cuando se define una columna de un tipo a REF, es posible restringir su dominio a los objetos que se almacenen en cierta tabla. Si la referencia no se asocia a una tabla, sino que sólo se restringe a un tipo de objeto, se podrá actualizar a una referencia a un objeto del tipo adecuado con independencia de la tabla donde se almacene. En este caso su almacenamiento requerirá más espacio y su acceso será menos eficiente.

Cuando se borran objetos de la base de datos, puede ocurrir que otros objetos que referencien a los borrados queden en estado inconsistente. Estas referencias se denominan dangling references, y Oracle proporciona el predicado llamado **IS DANGLING** que permite comprobar cuándo sucede esto.



EJEMPLO PRÁCTICO

El siguiente ejemplo define un atributo de tipo **REF** y restringe su dominio a los objetos de cierta tabla:

```
CREATE TABLE clientes_tab OF cliente_t;
CREATE TYPE ordenes_t AS OBJECT (
    ordnum NUMBER,
    cliente REF clientes_t,
    fechpedido DATE,
    direcentrega direccion_t);
CREATE TABLE ordenes_tab OF ordenes_t (
PRIMARY KEY (ordnum),
SCOPE FOR (cliente) IS clientes_tab);
```

2.4 Tipos de datos colección

Para poder implementar relaciones 1:N, Oracle permite definir tipos colección. Un dato de tipo colección está formado por un número indefinido de elementos, todos del mismo tipo.

De esta manera, es posible almacenar en un atributo un conjunto de tuplas en forma de array (**VARRAY**), o en forma de tabla anidada.

Al igual que los tipos objeto, los tipos colección también tienen por defecto unas funciones constructoras de colecciones cuyo nombre coincide con el del tipo. Los argumentos de entrada de estas funciones son el conjunto de elementos que forman la colección separados por comas y entre paréntesis, y el resultado es un valor del tipo colección.

En Oracle es posible diferenciar entre un valor nulo y una colección vacía. Para construir una colección sin elementos se puede utilizar la función constructora del tipo seguida por dos paréntesis sin elementos dentro.

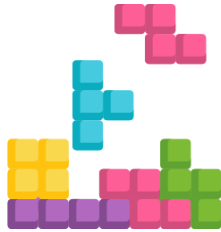
El tipo VARRAY: Un array es un conjunto ordenado de elementos del mismo tipo. Cada elemento tiene asociado un índice que indica su posición dentro del array. Oracle permite que los VARRAY sean de longitud variable, aunque es necesario especificar un tamaño máximo cuando se declara el tipo VARRAY.

Se puede utilizar el tipo VARRAY para:

- Definir el tipo de dato de una columna de una tabla relacional.
- Definir el tipo de dato de un atributo de un tipo de objeto.
- Para definir una variable PL/SQL, un parámetro, o el tipo que devuelve una función.

Cuando se declara un tipo VARRAY no se produce ninguna reserva de espacio. Si el espacio que requiere lo permite, se almacena junto con el resto de columnas de su tabla, pero si es demasiado largo (más de 4000 bytes) se almacena aparte de la tabla como un BLOB.

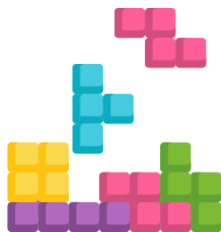
La principal limitación del tipo VARRAY es que en las consultas es imposible poner condiciones sobre los elementos almacenados dentro. Desde una consulta SQL, los valores de un VARRAY solamente pueden ser accedidos y recuperados como un bloque. Es decir, no se puede acceder individualmente a los elementos de un VARRAY. Sin embargo, desde un programa PL/SQL sí que es posible definir un bucle que itere sobre los elementos de un VARRAY.



EJEMPLO PRÁCTICO

Las siguientes declaraciones crean un tipo para una lista ordenada de precios, y un valor para dicho tipo:

```
CREATE TYPE precios AS VARRAY(10) OF NUMBER(12);
precios('35', '342', '3970');
```



EJEMPLO PRÁCTICO

En el siguiente ejemplo, se define un tipo de datos para almacenar una lista ordenada de teléfonos: el tipo **list** (ya que en el tipo **set** no existe orden). Este tipo se utiliza después para asignárselo a un atributo del tipo de objeto **cliente_t**.

DEFINICIÓN ORIENTADA A OBJETOS

```
define type Lista_Tel_T:
  list(string);

define class Cliente_T:
  tuple [clinum: integer,
        clinomb:string,
        direccion:Direccion_T,
        lista_tel: Lista_Tel_T];
```

DEFINICIÓN EN ORACLE

```
CREATE TYPE lista_tel_t AS
  VARRAY(10) OF VARCHAR2(20) ;

CREATE TYPE cliente_t AS OBJECT (
  clinum NUMBER,
  clinomb VARCHAR2(200),
  direccion direccion_t,
  lista_tel lista_tel_t );
```

Tablas anidadas: Una tabla anidada es un conjunto de elementos del mismo tipo sin ningún orden predefinido. Estas tablas solamente pueden tener una columna que puede ser de un tipo de datos básico de Oracle, o de un tipo de objeto definido por el usuario. En este último caso, la tabla anidada también puede ser considerada como una tabla con tantas columnas como atributos tenga el tipo de objeto.

El siguiente ejemplo declara una tabla que después será anidada en el tipo **ordenes_t**. Los pasos de todo el diseño son los siguientes:

Se define el tipo de objeto **linea_t** para las filas de la tabla anidada.

```
define type Linea_T:
tuple [linum:integer,
item:string,
cantidad:integer,
descuento:real];
CREATE TYPE linea_t AS OBJECT (
linum NUMBER,
item VARCHAR2(30),
cantidad NUMBER,
descuento NUMBER(6,2));
```

Se define el tipo colección tabla `lineas_pedido_t` para después anidarla.

```
CREATE TYPE lines_pedido_t AS TABLE OF linea_t ;
```

Esta definición permite utilizar el tipo colección **lines_pedido_t** para:

- Definir el tipo de dato de una columna de una tabla relacional.
- Definir el tipo de dato de un atributo de un tipo de objetos.
- Para definir una variable PL/SQL, un parámetro, o el tipo que devuelve una función.

Se define el tipo objeto `ordenes_t` y su atributo pedido almacena una tabla anidada del tipo `lineas_pedido_t`.

```
define class Ordenes_T:
tuple [ordnum:integer,
cliente:Clientes_T,
fechpedido:date,
fechentrega:date,
pedido:set (Linea_T),
direcentrega:Direccion_T];
```

```
CREATE TYPE ordenes_t AS OBJECT (
ordnum NUMBER,
cliente REF cliente_t,
fechpedido DATE,
fechentrega DATE,
pedido lines_pedido_t,
direcentrega direccion_t);
```

Se define la tabla de objetos `ordenes_tab` y se especifica la tabla anidada del tipo `lineas_pedido_t`.

```
CREATE TABLE ordenes_tab OF ordenes_t
(ordnum PRIMARY KEY,
SCOPE FOR (cliente) IS clientes_tab)
NESTED TABLE pedido STORE AS pedidos_tab);
```

Este último paso es necesario realizarlo porque la declaración de una tabla anidada no reserva ningún espacio para su almacenamiento. Lo que se hace es indicar en qué tabla (pedidos_tab) se deben almacenar todas las líneas de pedido que se representen en el atributo pedido de cualquier objeto de la tabla ordenes_tab. Es decir, todas las líneas de pedido de todas las órdenes se almacenan externamente a la tabla de órdenes, en otra tabla especial. Para relacionar las tuplas de una tabla anidada con la tupla a la que pertenecen, se utiliza una columna oculta que aparece en la tabla anidada por defecto. Todas las tuplas de una tabla anidada que pertenecen a la misma tupla tienen el mismo valor en esta columna (**NESTED_TABLE_ID**).

A diferencia de los VARRAY, los elementos de las tablas anidadas (NESTED_TABLE) sí pueden ser accedidos individualmente, y es posible poner condiciones de recuperación sobre ellos. En la próxima sección se estudia, una forma conveniente de acceder individualmente a los elementos de una tabla anidada mediante un cursor anidado. Además, las tablas anidadas pueden estar indexadas.

2.5 Inserción y acceso a los datos

Alias: En una base de datos con tipos y objetos, lo más recomendable es utilizar siempre alias para los nombres de las tablas. El alias de una tabla debe ser único en el contexto de la consulta. Los alias sirven para acceder al contenido de la tabla, pero hay que saber utilizarlos adecuadamente en las tablas que almacenan objetos.

El siguiente ejemplo ilustra cómo se deben utilizar.

```
CREATE TYPE persona AS OBJECT (nombre VARCHAR(20));  
CREATE TABLE ptab1 OF persona;  
CREATE TABLE ptab2 (c1 persona);  
CREATE TABLE ptab3 (c1 REF persona);
```

La diferencia entre las dos primeras tablas está en que la primera almacena objetos del tipo persona, mientras que la segunda tabla tiene una columna donde se almacenan valores del tipo persona.

Considerando ahora las siguientes consultas, se ve cómo se puede acceder a estas tablas.

1. SELECT nombre FROM ptab1; Correcto
2. SELECT c1.nombre FROM ptab2; Incorrecto
3. SELECT p.c1.nombre FROM ptab2 p; Correcto
4. SELECT p.c1.nombre FROM ptab3 p; Correcto

5. SELECT p.nombre FROM ptab3 p; Incorrecto

En la primera consulta nombre es considerado como una de las columnas de la tabla ptab1, ya que los atributos de los objetos se consideran columnas de la tabla de objetos. Sin embargo, en la segunda consulta se requiere la utilización de un alias para indicar que nombre es el nombre de un atributo del objeto de tipo persona que se almacena en la columna c1. Para resolver este problema no es posible utilizar los nombres de las tablas directamente: ptab2.c1.nombre es incorrecto. Las consultas 4 y 5 muestran cómo acceder a los atributos de los objetos referenciados desde un atributo de la tabla ptab3.

En conclusión, para facilitar la formulación de consultas y evitar errores se recomienda utilizar alias para acceder a todas las tablas que contengan objetos con o sin identidad, y para acceder a las columnas de las tablas en general.

Inserción de referencias: La inserción de objetos con referencias implica la utilización del operador REF para poder insertar la referencia en el atributo adecuado.

La siguiente sentencia inserta una orden de pedido en la tabla definida en la sección anterior.

```
INSERT INTO ordenes_tab
SELECT 3001, REF(C), '30-MAY-1999', NULL
--se seleccionan los valores de los 4 atributos de la tabla
FROM cliente_tab C WHERE C.clinum= 3;
```

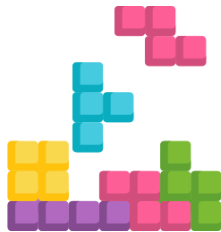
El acceso a un objeto desde una referencia **REF** requiere primero referenciar al objeto. Para realizar esta operación, Oracle proporciona el operador **DEREF**. No obstante, utilizando la notación de punto también se consigue referenciar a un objeto de forma implícita.

Obsérvese el siguiente caso:

```
CREATE TYPE persona_t AS OBJECT (
  nombre VARCHAR2(30),
  jefe REF persona_t ) ;
```

Si **x** es una variable que representa a un objeto de tipo **persona_t**, entonces las dos expresiones siguientes son equivalentes:

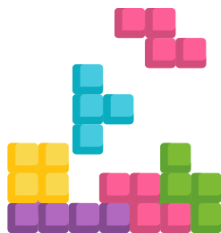
1. x.jefe.nombre
2. y.nombre, y=DEREF(x.jefe)



EJEMPLO PRÁCTICO

Para obtener una referencia a un objeto de una tabla de objetos, se puede aplicar el operador **REF** como muestra el siguiente ejemplo.

```
CREATE TABLE persona_tab OF persona_t;
DECLARE ref_persona REF persona_t;
SELECT REF(pe) INTO ref_persona
FROM persona_tab pe WHERE pe.nombre= 'Juan Pérez Ruíz';
```



EJEMPLO PRÁCTICO

Simétricamente, para recuperar un objeto desde una referencia es necesario usar **DEREF**, como muestra el siguiente ejemplo que visualiza los datos del jefe de la persona indicada.

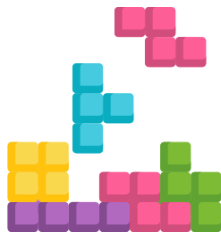
```
SELECT DEREF(pe.jefe)
FROM persona_tab pe WHERE pe.nombre= 'Juan Pérez Ruíz';
```

Llamadas a métodos: Para invocar un método hay que utilizar su nombre y unos paréntesis que encierren sus argumentos de entrada. Si el método no tiene argumentos, se especifican los paréntesis, aunque estén vacíos.

Por ejemplo, si **tb** es una tabla con la columna **c** de tipo de objeto **t**, y **t** tiene un método **m** sin argumentos de entrada, la siguiente consulta es correcta.

```
SELECT p.c.m( ) FROM tb p;
```

Inserción en tablas anidadas: Además del constructor del tipo de colección disponible por defecto, la inserción de elementos dentro de una tabla anidada puede hacerse siguiendo estas dos etapas: 1. Crear el objeto con la tabla anidada y dejar vacío el campo que contiene las tuplas anidadas y 2. Comenzar a insertar tuplas en la columna correspondiente de la tupla seleccionada por una subconsulta.



EJEMPLO PRÁCTICO

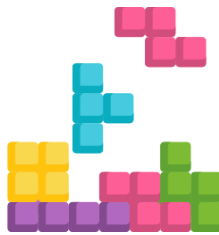
El siguiente ejemplo ilustra la manera de realizar estas operaciones sobre la tabla de órdenes.

```
INSERT INTO ordenes_tab --inserta una orden
SELECT 3001, REF(C),
        SYSDATE,'30-MAY-1999',
        lineas_pedido_t(),
        NULL
FROM cliente_tab C
WHERE C.clinum= 3 ;
```

```
INSERT INTO THE ( --selecciona el atributo pedido de la
orden
SELECT P.pedido
FROM ordenes_tab P
WHERE P.ordnum = 3001
)
VALUES (linea_t(30, NULL, 18, 30));
--inserta una línea de pedido anidada
```

Para ello, se tiene que utilizar la palabra clave THE con la siguiente sintaxis: **INSERT INTO THE** (subconsulta) (tuplas a insertar). Esta técnica es especialmente útil si dentro de una tabla anidada se guardan referencias a otros objetos.

Para poner condiciones a las tuplas de una tabla anidada, se pueden utilizar cursores dentro de un **SELECT** o desde un programa PL/SQL.



EJEMPLO PRÁCTICO

Véase aquí un ejemplo de acceso con cursores. Utilizando el ejemplo de la sección 1.1.4.2, se va a recuperar el número de las ordenes, sus fechas de pedido y las líneas de pedido que se refieran al ítem 'CH4P3'.

```
SELECT ord.ordnum, ord.fechpedido,  
CURSOR (SELECT * FROM TABLE(ord.pedido) lp WHERE lp.item= 'CH4P3')  
FROM ordenes_tab ord;
```

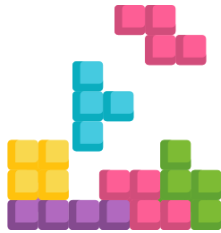
La cláusula THE también sirve para seleccionar las tuplas de una tabla anidada. La sintaxis es como sigue: **SELECT** ... **FROM THE** (*subconsulta*) **WHERE** ...



EJEMPLO PRÁCTICO

Por ejemplo, para seleccionar las primeras dos líneas de pedido de la orden **8778** se hace:

```
SELECT lp FROM THE  
(SELECT ord.pedido FROM ordenes_tab ord WHERE  
ord.ordnum= 8778) lp  
WHERE lp.linum<3;
```



EJEMPLO PRÁCTICO

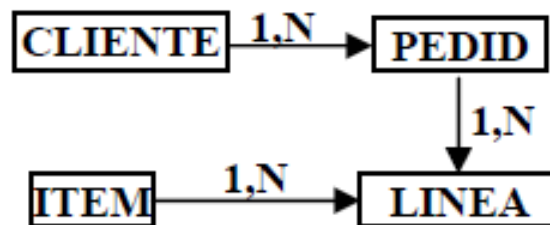
Se parte de una base de datos para gestionar los pedidos de los clientes, y se ve cómo Oracle proporciona una solución relacional y otra objeto-relacional:

CLIENTE(clinum, clinomb, calle, ciudad, prov, codpos, tel1, tel2, tel3)

PEDIDO (ordnum, clinum, fechped, fechentrega, callent, ciuent, provent, codpent)

ITEM(numitem, precio, tasas)

LINEA(linum, ordnum, numitem, cantidad, descuento)



Modelo lógico relacional.

```

CREATE TABLE pedido (
  ordnum NUMBER,
  clinum NUMBER REFERENCES cliente,
  fechpedido DATE,
  fechaentrega DATE,
  callent VARCHAR2(200),
  ciuent VARCHAR2(200),
  provent CHAR(2),
  codpent VARCHAR2(20),
  PRIMARY KEY (ordnum));
  
```

```

CREATE TABLE ítem (
  numitem NUMBER PRIMARY KEY,
  precio NUMBER,
  tasas NUMBER);
  
```

```

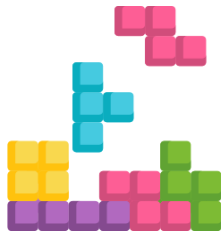
CREATE TABLE línea (
  PEDID
  ITEM LINEA
  1,N
  1,N
  1,N
  CLIENTE
  
```

```

linum NUMBER,
ordnum NUMBER REFERENCES pedido,
numitem NUMBER REFERENCES ítem,
cantidad NUMBER,
descuento NUMBER,
PRIMARY KEY (ordnum, linum));

```

Modelo lógico de una base de datos orientada a objetos: Primero se va a utilizar un lenguaje de definición de bases de datos orientado a objetos para definir el esquema de la base de datos que después se creará en Oracle.



EJEMPLO PRÁCTICO

Ejemplo de modelo lógico:

```

define type Lista_Tel_T: type list(string);
define type Direccion_T: type tuple [ calle:string,
ciudad:string,
prov:string,
codpos:string];

define class Cliente_T: type tuple [ clinum: integer,
clinomb:string,
direccion:Direccion_T,
lista_tel: Lista_Tel_T];

define class Item_T: type tuple [ itemnum:integer,
precio:real,
tasas:real];

define type Linea_T: type tuple [linum:integer,
item:Item_T,
cantidad:integer,
descuento:real];

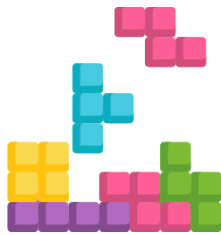
define type Linea_Pedido_T: type set(Linea_T);

define class Pedido_T: type tuple [ ordnum:integer,
cliente:Cliente_T,
fechpedido:date,
fechentrega:date,
pedido:Lineas_Pedido_T
direcentrega:Direccion_T];

```

Implementación objeto relacional con Oracle:

Creación de tipos: Se va a indicar cómo definir todos los tipos anteriores en Oracle.



EJEMPLO PRÁCTICO

Definición de tipos en Oracle:

```
CREATE TYPE lista_tel_t AS VARRAY(10) OF
VARCHAR2(20);
```

```
CREATE TYPE direccion_t AS OBJECT (
calle VARCHAR2(200),
ciudad VARCHAR2(200),
prov CHAR(2),
codpos VARCHAR2(20));
```

```
CREATE TYPE cliente_t AS OBJECT (
clinum NUMBER,
clinomb VARCHAR2(200),
direccion direccion_t,
lista_tel lista_tel_t) ;
```

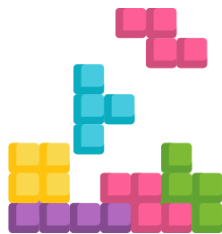
```
CREATE TYPE item_t AS OBJECT (
itemnum NUMBER,
precio NUMBER,
tasas NUMBER);
```

```
CREATE TYPE linea_t AS OBJECT (
linum NUMBER,
ítem REF item_t,
cantidad NUMBER,
descuento NUMBER);
```

```
CREATE TYPE lineas_pedido_t AS TABLE OF linea_t;
```

```
CREATE TYPE pedido_t AS OBJECT (
ordnum NUMBER,
cliente REF cliente_t,
fechpedido DATE,
fechentrega DATE,
pedido lineas_pedido_t,
direcentrega direccion_t);
```

Creación de tablas objeto: Ahora se va a crear las tablas donde almacenar los objetos de la aplicación.



EJEMPLO PRÁCTICO

Creación de tablas:

```
CREATE TABLE cliente_tab OF cliente_t  
(clinum PRIMARY KEY);
```

```
CREATE TABLE item_tab OF item_t  
(itemnum PRIMARY KEY);
```

```
CREATE TABLE pedido_tab OF pedido_t (  
PRIMARY KEY (ordnum),  
SCOPE FOR (cliente) IS cliente_tab)  
NESTED TABLE pedido STORE AS pedido_tab;  
ALTER TABLE pedido_tab  
ADD (SCOPE FOR (item) IS item_tab);
```

Esta última declaración sirve para restringir el dominio de los objetos referenciados desde **ítem** a aquellos que se almacenan en la tabla **item_tab**.



EJEMPLO PRÁCTICO

Ejemplo de inserción de objetos:

```
REM inserción en la tabla  
ITEM_TAB*****  
INSERT INTO item_tab VALUES(1004, 6750.00, 2);  
INSERT INTO item_tab VALUES(1011, 4500.23, 2);  
INSERT INTO item_tab VALUES(1534, 2234.00, 2);  
INSERT INTO item_tab VALUES(1535, 3456.23, 2);  
INSERT INTO item_tab VALUES(2004, 33750.00, 3);  
INSERT INTO item_tab VALUES(3011, 43500.23, 4);  
INSERT INTO item_tab VALUES(4534, 5034.00, 6);  
INSERT INTO item_tab VALUES(5535, 34456.23, 5);  
REM inserción en la tabla  
CLIENTE_TAB*****
```


Nótese cómo en estas definiciones se utilizan los constructores del tipo de objeto `direccion_t` y el tipo de colección `lista_tel_t`.

```
INSERT INTO cliente_tab
VALUES (
  1, 'Lola Caro',
  direccion_t('12 Calle Lisboa', 'Nules', 'CS', '12678'),
  lista_tel_t('415-555-1212'));
```

```
INSERT INTO cliente_tab
VALUES (
  2, 'Jorge Luz',
  direccion_t('323 Calle Sol', 'Valencia', 'V', '08820'),
  lista_tel_t('609-555-1212','201-555-1212'));
```

```
INSERT INTO cliente_tab
VALUES (
  3, 'Juan Pérez',
  direccion_t('12 Calle Colon', 'Castellón', 'ES', '12001'),
  lista_tel_t('964-555-1212', '609-543-1212',
  '201-775-1212','964-445-1212'));
```

```
INSERT INTO cliente_tab
VALUES (
  4, 'Ana Gil',
  direccion_t('5 Calle Sueca', 'Burriana', 'ES', '12345'),
  lista_tel_t());
```

REM inserción en la tabla PEDIDO_TAB*****

Nótese cómo en estas definiciones se utiliza el operador REF para obtener una referencia a un objeto de `cliente_tab` y almacenarlo en la columna de otro objeto de `pedido_tab`. La palabra clave THE se utiliza para designar la columna de las tuplas que cumplen la condición del WHERE, donde se deben realizar la inserción. Las tuplas que se insertan son las designadas por el segundo SELECT, y el objeto de la orden debe existir antes de comenzar a insertar líneas de pedido.

REM Pedidos del cliente 1*****

```
INSERT INTO pedido_tab
SELECT 1001, REF(C),
  SYSDATE,'10-MAY-1999',
  linea_pedido_t(),
NULL
FROM cliente_tab C
WHERE C.clinum= 1;
```

```
INSERT INTO THE (  
  SELECT P.pedido  
  FROM pedido_tab P  
  WHERE P.ordnum = 1001  
)  
  SELECT 01, REF(S), 12, 0  
  FROM item_tab S  
  WHERE S.itemnum = 1534;
```

```
INSERT INTO THE (  
  SELECT P.pedido  
  FROM ordenes_tab P  
  WHERE P.ordnum = 1001  
)  
  SELECT 02, REF(S), 10, 10  
  FROM item_tab S  
  WHERE S.itemnum = 1535;
```

REM Pedidos del cliente 2*****

```
INSERT INTO pedido_tab  
  SELECT 2001, REF(C),  
  SYSDATE,'20-MAY-1999',  
  lineas_pedido_t(),  
  direccion_t('55 Madison Ave','Madison','WI','53715')  
  FROM cliente_tab C  
  WHERE C.clinum= 2;
```

```
INSERT INTO THE (  
  SELECT P.pedido  
  FROM pedido_tab P  
  WHERE P.ordnum = 2001  
)  
  SELECT 10, REF(S), 1, 0  
  FROM item_tab S  
  WHERE S.itemnum = 1004;
```

```
INSERT INTO THE (  
  SELECT P.pedido  
  FROM pedido_tab P  
  WHERE P.ordnum= 2001  
)  
  VALUES( linea_t(11, NULL, 2, 1) ) ;
```

REM Pedidos del cliente 3*****

```
INSERT INTO pedido_tab
  SELECT 3001, REF(C),
  SYSDATE, '30-MAY-1999',
  lineas_pedido_t(),
  NULL
FROM cliente_tab C
WHERE C.clinum= 3;
```

```
INSERT INTO THE (
  SELECT P.pedido
FROM pedido_tab P
WHERE P.ordnum = 3001
)
SELECT 30, REF(S), 18, 30
FROM items_tab S
WHERE S.itemnum = 3011;
```

```
INSERT INTO THE (
  SELECT P.pedido
FROM pedido_tab P
WHERE P.ordnum = 3001
)
SELECT 32, REF(S), 10, 100
FROM item_tab S
WHERE S.itemnum = 1535;
```

```
INSERT INTO pedido_tab
  SELECT 3002, REF(C),
  SYSDATE, '15-JUN-1999',
  linea_pedido_t(),
  NULL
FROM cliente_tab C
WHERE C.clinum= 3;
```

```
INSERT INTO THE (
  SELECT P.pedido
FROM pedido_tab P
WHERE P.ordnum = 3002
)
SELECT 34, REF(S), 200, 10
FROM item_tab S
WHERE S.itemnum = 4534;
```

```

REM Pedidos del cliente 4*****
INSERT INTO pedido_tab
    SELECT 4001, REF(C),
    SYSDATE, '12-MAY-1999',
    linea_pedido_t(),
    direccion_t('34 Nave Oeste', 'Nules', 'CS', '12876')
FROM cliente_tab C
WHERE C.clinum = 4;

INSERT INTO THE (
    SELECT P.pedido
FROM pedido_tab P
WHERE P.ordnum = 4001
)
SELECT 41, REF(S), 10, 10
FROM item_tab S
WHERE S.itemnum = 2004;

INSERT INTO THE (
    SELECT P.pedido
FROM pedido_tab P
WHERE P.ordnum = 4001
)
SELECT 42, REF(S), 32, 22
FROM item_tab S
WHERE S.itemnum = 5535;

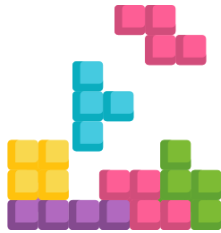
```

Definición de métodos para los tipos: El siguiente método calcula la suma de los valores de las líneas de pedido correspondientes a la orden de pedido sobre la que se ejecuta.

La palabra clave SELF permite referirse al objeto sobre el que se ejecuta el método.

La palabra clave COUNT sirve para contar el número de elementos de una tabla o de un array. Junto con la instrucción LOOP permite iterar sobre los elementos de una colección, en nuestro caso las líneas de pedido de una orden.

El SELECT es necesario porque Oracle no permite utilizar Deref directamente en el código PL/SQL.



EJEMPLO PRÁCTICO

Definido en definición de métodos para los tipos:

```
CREATE TYPE pedido_t AS OBJECT (
    ordnum NUMBER,
    cliente REF cliente_t,
    fechpedido DATE,
    fechentrega DATE,
    pedido linea_pedido_t,
    direcentrega direccion_t,
    MEMBER FUNCTION
        coste_total RETURN NUMBER,
    PRAGMA RESTRICT_REFERENCES(coste_total, WNDS,
        WNPS));
```

```
CREATE TYPE BODY pedido_t AS
    MEMBER FUNCTION coste_total RETURN NUMBER IS
        i INTEGER;
        ítem item_t;
        línea linea_t;
        total NUMBER:=0;

    BEGIN
        FOR i IN 1..SELF.pedido.COUNT LOOP
            línea:=SELF.pedido(i);
            SELECT Deref(línea.item) INTO ítem FROM DUAL;
            total:=total + línea.cantidad * ítem.precio;
        END LOOP;
        RETURN total;
    END;

END;
```

A continuación, se muestran ejemplos de consultas a BDOR.

Consultar la definición de la tabla de clientes:

```
SQL> describe cliente_tab;  
Name Null? Type  
CLINUM  
CLINOMB  
DIRECCION  
LISTA_TEL  
NOT NULL  
NUMBER  
VARCHAR2(200)  
DIRECCION_T  
LISTA_TEL_T
```

Insertar en la tabla de clientes a un nuevo cliente con todos sus datos:

```
SQL> insert into cliente_tab  
2 values(5, 'John smith',  
3 direccion_t('67 rue de percebe', 'Gijón', 'AS',  
'73477'), lista_tel_t('7477921749', '83797597827'));  
1 row created.
```

Consultar y modificar el nombre del cliente número 2:

```
SQL> select clinomb from cliente_tab where clinum=2;  
CLINOMB  
-----  
Jorge Luz  
SQL> update cliente_tab  
2 set clinomb='Pepe Puig' where clinum=5;  
1 row updated.
```

Consultar y modificar la dirección del cliente número 2:

```
SQL> select direccion from cliente_tab where clinum=2;  
DIRECCION(CALLE, CIUDAD, PROV, CODPOS)  
-----  
DIRECCION_T('Calle Sol', 'Valencia', 'VA', '08820')  
SQL> update cliente_tab  
2 set direccion=direccion_t('Calle Luna', 'Castello',
```

```
'CS',68734')
3 where clinum=2;
1 row updated.
```

Consultar todos los datos del cliente número 1 y añadir un nuevo teléfono a su lista de teléfonos:

```
SQL> select * from cliente_tab where clinum=1;
CLINUM
-----
CLINOMB
-----
DIRECCION(CALLE, CIUDAD, PROV, CODPOS)
-----
LISTA_TEL
-----
1
Lola Caro
DIRECCION_T('Calle Luna', 'Castellón', 'CS', '64827')
LISTA_TEL_T('415-555-1212')
También se podría haber consultado así:
SQL> select value(C) from cliente_tab C where C.clinum=1;
VALUE(C)(CLINUM, CLINOMB, DIRECCION(CALLE, CIUDAD,
PROV,
CODPOS),LISTA_TEL)
-----
CLIENTES_T(1, 'Lola Caro', DIRECCION_T( 'Calle Luna',
'Castellón', 'CS','64827'), LISTA_TEL_T('415-555-
1212'))
SQL> update cliente_tab
2 set lista_tel=lista_tel_t('415-555-1212',
'6348635872')
3 where clinum=1;
1 row updated
```

Visualizar el nombre del cliente que ha realizado la orden número 1001:

```
SQL> select o.cliente.clinomb from pedido_tab o where
o.ordnum=1001;
CLIENTE.CLINOMB
-----
Lola Caro
```

Visualizar todos los detalles del cliente que ha realizado la orden número 1001:

```
SQL> select Deref(o.cliente) from pedido_tab o where
o.ordnum=1001;
Deref(O.CLIENTE)(CLINUM, CLINOMB, DIRECCION(CALLE,
CIUDAD, PROV,
CODPOS),LISTA_TEL
-----
CLIENTE_T(1, 'Lola Caro', DIRECCION_T('Calle Luna',
'Castellón', 'CS','64827'),
LISTA_TEL_T('415-555-1212', '6348635872'))
De la siguiente manera se puede obtener la referencia al
objeto que, obviamente, es
ininteligible:
SQL> select o.cliente from pedido_tab o where
o.ordnum=1001;
CLIENTE
-----
00002EA5F6693E4A73F8E003960286473F83EA5F6693E3E7
3F8E0039680286473F8
```

Visualizar el número de todos los ítems que se han pedido en la orden número 3001:

```
SQL> select cursor(select p.item.itemnum from
Table(o.pedido) p)
2 from pedido_tab o where o.ordnum=3001;
CURSOR(SELECTP.ITEM.
-----
CURSOR STATEMENT : 1
CURSOR STATEMENT : 1
ITEM.ITEMNUM
-----
3011
1535
```


Seleccionar el número de orden y el coste total de las órdenes hechas por el cliente número:

```
SQL> select o.ordnum, o.coste_total() from pedido_tab o
2 where o.cliente.clinum=3;
ORDNUM O.COSTE_TOTAL()
-----
3001 817566.44
3002 1006800
```

Borrado de objetos, tablas y tipos de usuario:

```
DELETE FROM pedido_tab;
DROP TABLE pedido _tab;
DELETE FROM cliente_tab;
DROP TABLE cliente_tab;
DELETE FROM item_tab;
DROP TABLE item_tab;
DROP TYPE pedido _t;
DROP TYPE linea_pedido_t;
DROP TYPE linea_t;
DROP TYPE item_t;
DROP TYPE cliente_t;
DROP TYPE lista_tel_t;
DROP TYPE direccion_t;
```

RESUMEN FINAL

Los SGBDOR son una combinación entre los sistemas relacionales y las bases de datos orientadas a objetos. En este sentido, presentan tanto las ventajas como los inconvenientes de ambos. Normalmente, gestionan tipos de datos mucho más complejos de los habituales con menor esfuerzo, además crean funciones mediante tipos de datos que el propio usuario define (característica de la programación orientada a objetos), proporcionando flexibilidad y seguridad.