RECORDATORIO DE CONTENIDOS ADQUIRIDOS EL CURSO PASADO NECESARIOS PARA ESTE MÓDULO

<table>
<tr><th colspan="3">FICHEROS</th></tr>
<tr><th colspan="3">Tipos de ficheros</th></tr>
<tr><td colspan="2">BINARIOS</td><td>TEXTO</td></tr>
<tr><td colspan="2">Su información se almacena en binario, no se pueden leer directamente necesitamos de un programa editor especial dependiendo del archivo.
El número 12 se almacenará en binario utilizando tantos bytes dependiendo del tipo de dato que tenga (entero, float, etc...).</td><td>Se pueden leer fácilmente con cualquier editor de texto, la información se almacena en forma de caracteres, por ejemplo el número 12 se almacena como un carácter 1 y un carácter 2.
Dentro de esta clase de ficheros están los archivos XML que incluyen en el mismo fichero la definición de la estructura de la información utilizando etiquetas XML.</td></tr>
<tr><td colspan="3">Existen diferentes clases para tratar ficheros binarios (incluirán métodos para leer/escribir bytes/arrays de bytes/un datos de un terminado tipo) y ficheros de texto (incluirán métodos para leer/escribir caracteres/arrays de caracteres/una línea).
En cualquiera de los casos tendremos que conocer la estructura del archivo (saber qué datos hay en el archivo y de qué tipo son, su longitud, etc).
Generalmente los ficheros se leen a través de flujos de datos (streams).
El tratamiento de un fichero es igual en cualquiera de los lenguajes, lo que cambiará será la sintaxis de las instrucciones (las clases a utilizar) pero la mecánica es la misma.</td></tr>
<tr><td>Para tratar un fichero en modo ACCESO SECUENCIAL tenemos que:
Abrir el fichero
Leer del archivo
Mientras no fin de fichero
Tratar lo leído
Leer siguiente
Fin del bucle
Cerrar fichero</td><td>Dependiendo de cómo se detecta el fin de fichero podemos tener esto:
Abrir el fichero
Mientras no fin de fichero
Leer del archivo
Tratar lo leído
Fin del bucle
Cerrar fichero</td><td>Para tratar el fichero en modo ACCESO DIRECTO:
Abrir el fichero
Mientras queramos tratar información
Leer el registro deseado (a veces habrá que posicionarse previamente en él)
Tratar el registro
Fin del bucle
Cerrar fichero</td></tr>
<tr><td colspan="3">Veamos ahora algunas clases que se utilizan.</td></tr>
<tr><td colspan="2">VB .NET</td><td>JAVA</td></tr>
<tr><td colspan="2">◦ FileStream Para crear flujos y leer y escribir …bytes
◦ BinaryReader y BinaryWriter para leer/escribir cualquier tipo primitivo desde/en un archivo/flujo
◦ StreamReader y StreamWriter para leer/escribir caracteres desde/en un archivo/flujo</td><td>◦ InputStream/OutputStream para leer/escribir flujos de bytes.
◦ Reader/ Writer OutputStream para leer/escribir flujos de caracteres (Unicode)
◦ Según la forma de acceso al fichero:
◦ Acceso secuencial
• FileInputStream/FileOutputStream
• FileReader/FileWriter
◦ Acceso directo o aleatorio
• RandomAccessFile</td></tr>
</table>

MANEJO DE FICHEROS

FICHEROS DE TEXTO

LEER LINEA A LINEA DE UN FICHERO DE TEXTO

```
Dim f As StreamReader = New StreamReader("doc.txt")
Dim linea As String
While ((linea = f.ReadLine()) <> Nothing)
    Console.WriteLine(linea)
End While
f.close()
```

```
FileReader f = new FileReader("doc.txt");
BufferedReader l = new BufferedReader(f);
String linea;
 while((linea = l.readLine()) != null)
       System.out.println(linea);
 f.close();
```

ESCRIBIR LINEAS EN UN FICHERO DE TEXTO

```
Dim f As StreamWriter = New StreamWriter("doc.txt")
For i = 0 To 9
       f.WriteLine("Línea número: " & i)
 Next i
f.Close()
```

```
File fw = new FileWriter("doc2.txt");
BufferedWriter f = new BufferedWriter(fw);
for (int i=0; i<10; i++)
      f.write("Línea número: " + x + "\n");
f.close();
```

¡Ojo! Para el salto de línea lo tienes que incluir en la línea que escribes (+"\n"), o bien después de f.write(); añadir f.newline();

LEER CARACTERES DE UN FICHERO DE TEXTO

El método Read() sin parámetros lee un carácter y devuelve el entero asociado, si queremos obtener el carácter correspondiente lo tenemos que transformar mediante char() (En VB) o mediante un CAST (en Java).

```
Dim f As StreamReader = New StreamReader("doc.txt")
Dim i As Integer
While (i = f.Read()) <> -1
      Console.WriteLine(Chr(i))
EndWhile
f.close()
```

```
FileReader f = new FileReader("doc.txt");
Int i;
 while((i = f.read()) != -1)
       System.out.println((char) i);
 f.close();
```

ESCRIBIR CARACTERES EN UN FICHERO DE TEXTO

Utilizando la clase StreamWriter y su método .write()

Utilizando la clase FileWriter y su método .write()

FICHEROS BINARIOS

Después de ver las clases orientadas a flujos de caracteres veamos las orientadas a flujos de bytes especializadas en tratar ficheros binarios. EN un fichero binario los valores almacenados ya no son caracteres son bytes que unidos a otros bytes forman un valor de un determinado tipo, por lo que ahora nos tendremos que fijar muy bien en los tipos de datos de los valores que se escriben o recuperan.

- La clase FileStream se utiliza para definir secuencias para leer y escribir BYTES de/a archivos. Dispone del método Seek para colocar la posición actual en una determina posición del fichero.
- Las clases BinaryReader y BinaryWriter están especializadas en tratar ficheros binarios. Los flujos creados con los objetos BinaryReader y BinaryWriter son flujos de bytes como los objetos FileStream, pero tienen métodos que simplifican el trabajo ya que permiten leer/escribir datos primitivos.

- Las clases que se utilizan son FileInputStream y FileOutputStream. Los métodos que utilizan son read() y write() para leer/escribir bytes. El método read() devuelve -1 si se ha llegado al final del fichero.
- Las clases DataInputStream y DataOutputStream proporcionan métodos para lectura y escritura de datos primitivos.

- El constructor de BinaryReader abre un flujo de entrada desde otro flujo existente.
- Ejemplo

```
fs = New FileStream("nombredefichero", FileMode.Open,
FileAccess.Read)
Dim r As New BinaryReader(fs)
```

```
File f = new File("nombredelfichero");
FileInputStream fi = new FileInputStream(f);
DataInputStream d = new DataInputStream(fi);
```

Los métodos asociados a estas clases son muy similares por lo que solo los nombraremos una vez y puedes consultar la ayuda de cada clase para ver los métodos específicos según el lenguaje.

Algunos métodos del objeto BinaryReader y DataInputStream:

- **ReadByte** Devuelve un valor de tipo byte
- **ReadChar** Devuelve un valor de tipo Char
- **ReadDecimal** Devuelve un valor de tipo Decimal
- **ReadDouble**Devuelve un valor de tipo Double
- ……

En vez de leer bytes y luego hacer la conversión, los métodos anteriores se encargan de leer la cantidad de bytes necesaria para leer un dato del tipo indicado.

- Para la escritura tanto el BinaryReader como el DataInputStream utilizan el método write() y según el tipo de dato del valor indicado como parámetro, escribirá ese valor en el formato correspondiente.

ACCESO DIRECTO

El acceso directo consiste en colocar el puntero de lectura en una determinada posición del archivo para leer a partir de esa posición.

Cuando utilizamos este método para acceder a posiciones concretas del fichero normalmente es porque tenemos la información del fichero organizada en registros y lo que buscamos es posicionarnos en un registro determinado.

Para ello nuestros registros deben tener una longitud fija para que podamos saber dónde empieza cada registro, y además los registros los identificaremos, los "nombraremos" por un número que corresponderá a su número de orden dentro del fichero.

Por ejemplo en la imagen se ven tres registros del fichero:

Para leer un determinado registro se calcula a partir del inicio del archivo, cual es el primer byte que deseamos leer, si estamos trabajando con una estructura de datos que es de longitud fija, el registro x empezará en:

Posición = Longitud de registro * (X – 1)

Si el registro tiene 30 bytes de longitud, el registro número dos empezará en el byte 30. Posición = 30 * (2 – 1)

Para manejar este tipo de acceso necesitaremos poder posicionarnos en una determinada posición del archivo y saber en qué posición se encuentra el puntero de lectura/escritura.

Para ello tendremos método similares en los dos lenguajes:

- Para posicionarse: seek()
- Para saber en qué posición se encuentra el puntero de lectura/escritura: current() en VB y getFilePointer() en Java.

SERIALIZACIÓN

La serialización es el proceso de convertir el estado de un objeto a un formato que se pueda almacenar o transportar. El complemento de la serialización es la deserialización, que convierte una secuencia en un objeto.

Juntos, estos procesos permiten almacenar y transferir fácilmente datos con estructuras más o menos complejas entre diferentes sistemas.La serialización puede ser:

- Binaria: objeto ---> conjunto bytes
- XML: objeto ---> documento XML

VB.NET

La clase central de la serialización XML es **XmlSerializer** y sus métodos más importantes son **Serialize** y **Deserialize**.

- Se pueden serializar utilizando XmlSerializer:
  - Propiedades públicas de lectura y escritura.
  - Campos de clases públicas.
  - Clases que implementan **ICollection** o **IEnumerable**
  - Objetos **XmlElement**
  - Objetos **XmlNode**
  - Objetos **DataSet**

JAVA

- No todos los objetos se pueden serializar, para que un objeto sea serializable basta con que la clase a la que pertenezca, o una superclase de ésta, implemente la interfaz Serializable del paquete java.io.
- Para serialización XML se utiliza la librería XStream.

Para la serialización XML:

Por defecto, cada objeto se guardará como una etiqueta cuyo nombre es el nombre de la clase del objeto y cada atributo se convertirá en una etiqueta interna, el nombre del atributo pasará a ser el nombre de la etiqueta y el valor almacenado en la propiedad del objeto será el texto incluido en la etiqueta.

Si la clase contiene una lista, se creará también una etiqueta con el nombre de la lista.

Se puede alterar el comportamiento por defecto de la serialización como cambiar los nombres de las etiquetas que se generarán.

Tanto si es serialización XML como binaria, el proceso a seguir es muy sencillo:

Una vez tenemos los objetos a serializar.

Creamos el "serializador".

Podemos si queremos alterar los parámetros de serialización para alterar el comportamiento por defecto.

Aplicamos el método de serializar y enviamos el resultado al flujo de salida que nos permita guardarlo en el archivo correspondiente, o donde queramos.

Veamos un ejemplo:

- Por ejemplo, tenemos estas clases:

```
Public Class Producto
Private cNombre As String
Private cDescripcion As String
....

Public Class ListaProductos Implements ICollection
Private aProductos As ArrayList = New ArrayList()
......

(Para acortar no se ha incluido la definición  completa de la clase)
```

- Por ejemplo, tenemos estas clases:

```
public class Producto  implements Serializable {
        private String cNombre;
        private String cDescripcion;
        …

public class ListaProductos{
  private List<Producto> lista = new ArrayList<Producto>();
  .....

    (Para acortar no se ha incluido la definición  completa de la clase)
```

```
--Importamos lo necesario:
Imports System.Xml.Serialization
-- Crear el objeto a serializar (una lista de productos) y rellenarlo:
-- Las clases ListaProductos y Producto ya las tenemos definidas
Dim lista As ListaProductos = New ListaProductos()
For i = 0 To 5
        Dim p As New Producto
        p.Nombre = "Producto" & i
        p.Descripcion = "Descripcion producto " & i
        lista.Add(p)
Next i

-- Ya tenemos el objeto completo, ahora lo vamos a serializar

-- Crear un XmlSerializar indicando el tipo de objeto a serializar
Dim flujox As XmlSerializer = New XmlSerializer(GetType(ListaProductos))

-- Definir un método de transporte
Dim escritor As TextWriter = New StreamWriter("Productos.xml")
-- Serializar el objeto indicando método de transporte y objeto.
  flujox.Serialize(escritor, lista)
```

```
//Importamos lo necesario:
import com.thoughtworks.xstream.XStream;
// Crear el objeto a serializar (una lista de productos) y rellenarlo:
// Las clases ListaProductos y Producto ya las tenemos definidas
ListaProductos lista = new ListaProductos();
for (int i=0;i<6;i++) {
    Producto p = new Producto() ;
    p.Nombre = "Producto" + i
    p.Descripcion = "Descripcion producto " + i
    lista.add(p);
}

// Ya tenemos el objeto completo, ahora lo vamos a serializar

// Crear un XStream
     XStream flujox = new XStream();

// Utilizamos un FileOutputStream como método de transporte y
directamente serializamos
     flujox.toXML(lista,new FileOutputStream("Personas.xml"));
```

- En cualquiera de los casos, el archivo xml obtenido tendrá este aspecto:

```
<ficheros.xml.ListaPersonas>
  <lista>
    <ficheros.xml.Persona>
        <nombre>Ana</nombre>
        <edad>14</edad>
    </ficheros.xml.Persona>
    <ficheros.xml.Persona>
        <nombre>Luis</nombre>
        <edad>15</edad>
    </ficheros.xml.Persona>
    <ficheros.xml.Persona>
        <nombre>Alicia</nombre>
        <edad>13</edad>
    </ficheros.xml.Persona>
    <ficheros.xml.Persona>
      <nombre>Pedro</nombre>
      <edad>15</edad>
    </ficheros.xml.Persona>
    <ficheros.xml.Persona>
      <nombre>Manuel</nombre>
      <edad>16</edad>
    </ficheros.xml.Persona>
    <ficheros.xml.Persona>
```

Para la operación inversa de deserialización, cargar en un objeto el contenido de un archivo XML:

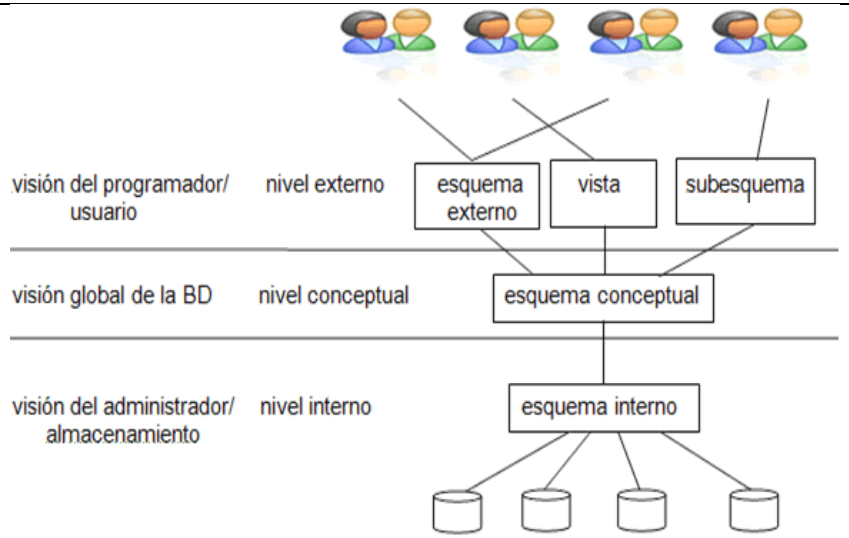
```
-- Crear el objeto que va a recibir el resultado de la deserialización:
Dim lista As ListaProductos = New ListaProductos()
-- Crear un XmlSerializar indicando el tipo de objeto a deserializar
Dim x As XmlSerializer = New XmlSerializer(GetType(ListaProductos))
-- Definir un método de transporte
Dim fs As FileStream = New FileStream("Productos.xml", FileMode.Open)
-- Deserializar el objeto indicando método de transporte y asignando el
resultado al objeto.
    lista = x.Deserialize(fs)
```

```
// Crear el objeto que va a recibir el resultado de la deserialización:
ListaProductos lista = new ListaProductos();
// Definimos un XStream
XStream x = new XStream();
// Definir un InputStream para obtner los datos del fichero
FileInputStream fs = new FileInputStream("Productos.xml"));
// Deserializar el contenido del fichero al objeto
ListaProductos lista = (ListaProductos) x.fromXML(fs);
// indicamos a qué tipo de objeto pasamos el flujoXML (ListaPersonas)
```

<table>
<tr><th colspan="2">BASES DE DATOS RELACIONALES</th></tr>
<tr><td>Los datos se organizan en tablas formadas por filas y columnas
・ Fila /Registro
・ Columna/Campo</td><td> 
</td></tr>
<tr><td>・ Cada tabla tiene un nombre único
・ Puede contener 0, una o más filas
・ Las filas en principio están desordenadas
・ En principio no pueden haber filas duplicadas
・ La lista de los atributos que forman la definición de una tabla se denomina esquema de la tabla.
・ Los valores concretos de los datos que están almacenados en la tabla se llaman ocurrencias.
・ Todos los valores de una columna tienen el mismo tipo de datos, y éstos están extraídos de un conjunto de valores legales llamado dominio de la columna.

・ Existe un valor especial, el valor nulo NULL. NULL representa la ausencia de valor. NO es lo mismo que el valor cero 0 o la cadena vacía o espacios en blanco.
・ El valor NULL funciona de forma diferente a los demás valores.</td><td>・ Toda tabla debe tener una clave primaria (principal) PRIMARY KEY.
・ La clave primaria es un campo o combinación de campos que permite identificar de forma unívoca cada fila de la tabla.
・ No admite nulos ni valores duplicados.
・ En una tabla no pueden haber dos claves primarias pero sí una clave compuesta.
・ En una tabla se pueden definir claves alternativas.
・ Una clave alternativa es un campo o combinación de campos que permite identificar de forma unívoca cada fila de la tabla. Y que se utilizará para tal fin en algunas ocasiones.
・ No admite nulos ni valores duplicados.
・ En una tabla se pueden definir claves ajenas (foráneas/externas) FOREIGN KEY.
・ Una clave ajena es un campo o combinación de campos que señala una fila de otra tabla.
・ El valor que contiene identifica el registro en la otra tabla (tabla referenciada).
・ Admite nulos y valores duplicados.</td></tr>
<tr><td colspan="2">・ El SGBD vela por la integridad de los datos, para ello incluye varias reglas de integridad que se comprobarán de forma automática sin necesidad de la intervención externa de los usuarios o de los programas de aplicación.
・ La integridad de la base de datos consiste en que no existan datos erróneos.
・ Integridad de claves. “Toda tabla debe tener una clave primaria que permite identificar unívocamente los registros que contiene, por lo tanto no puede contener el valor nulo ni valores duplicados”.
・ Integridad referencial. “En una clave ajena no puede haber un valor no nulo que no exista en la tabla de referencia”.
A nivel de control sobre los datos, el SGBD debe de proporcionar herramientas para poder definir restricciones de dominio que se comprobarán de forma automática, y reglas de negocio, reglas específicas sobre los datos, en este tipo de reglas entran las reglas de validación y reglas definidas a nivel superior mediante triggers, por ejemplo.</td></tr>
</table>

Un SGBD relacional sigue la arquitectura de tres niveles en la que tenemos en el nivel externo las **vistas** en nivel conceptual el esquema conceptual con la definición de todas las tablas, columnas que las componen y relaciones entre ellas, en el nivel interno tenemos el esquema interno/físico (la definición física de la base de datos).



Para poder manejar la información almacenada en la base de datos disponemos de un lenguaje que cumple las reglas de Codd, el lenguaje SQL que veremos en próximos temas.

- Este lenguaje está basado en operaciones de álgebra relacional.
- SQL (*STRUCTURED QUERY LANGUAGE*), Lenguaje Estructurado de Interrogación/Consulta es el lenguaje utilizado para definir, controlar y acceder a los datos almacenados en una base de datos relacional.

- Como ejemplos de sistemas gestores de bases de datos que utilizan SQL podemos citar DB2, SQL Server, Oracle, MySql, Sybase, PostgreSQL, Foxpro, Access…

El SQL es un lenguaje universal que se emplea en cualquier sistema gestor de bases de datos relacional. Tiene un estándar definido, a partir del cual cada sistema gestor ha desarrollado su versión propia.
Las instrucciones SQL se clasifican según su propósito en tres grupos:

- El DDL (*DATA DESCRIPTION LANGUAGE*) Lenguaje de Descripción de Datos.
- El DML (*DATA MANIPULATION LANGUAGE*) Lenguaje de Manipulación de Datos.
- El DCL (*DATA CONTROL LANGUAGE*) Lenguaje de Control de Datos.

Fabricantes

IdFab	Nombre	Direccion
Fab1	Muebles La Madera	Null
Fab2	Maderas Asociados	Avda. del Sol, nave3
Fab5	Industrias Madereras	Pol. Ind 3 fuentes, nave 23

Piezas

Codigo	Denominacion	Precio	Fabricante	Codigo_segun_fab
1	Taburete 3 patas	25	Fab1	T123-34
2	Mesa ovalada	1200	Fab1	M234-87
3	Taburete 3 patas	78	Fab2	T2S-0P

BASES DE DATOS OBJETO-RELACIONALES

Las bases de datos objeto-relacionales son una extensión de las bases de datos relacionales a las que se les ha añadido conceptos del modelo orientado a objetos.

Permiten datos complejos, se pueden almacenar objetos en las tablas, podemos tener tablas de objetos, tablas anidadas e incluso tener implementado el concepto de herencia.

El lenguaje se ha adaptado también a partir del SQL-1999/SQL-99/SQL3

Ejemplo: Oracle

Se pueden crear tipos nuevos:

```
CREATE OR REPLACE TYPE TADDRESS AS OBJECT
(
STREET VARCHAR(30),
ZIPCOD NUMBER(5),
CITY VARCHAR(30)
);
```

Y usarlos luego

```
CREATE OR REPLACE TYPE TPERSON AS OBJECT
(
ID NUMBER,
NAME VARCHAR(40),
ADDRESS TADDRESS,
BIRTH_DATE DATE
);
```

El propio SQL nos permite definir variables y usarlas:

```
DECLARE
ADDR TADDRESS:= TADDRESS(NULL,NULL,NULL);
PERSON TPERSON:=(NULL,NULL,NULL,NULL);
BEGIN
ADDR.STREET:= 'C/ DEL MAR, 1';
ADDR.ZIPCOD:=46001;
ADDR.CITY:='VALENCIA';
PERSON.ID:=1;
PERSON.NAME:='ANA';
PERSON.ADDRESS:=DIR;
PERSON.BIRTH_DATE:='01/01/1980';
END;
```

Podemos tener objetos con métodos:

```
CREATE OR REPLACE TYPE TADDRESS AS OBJECT
(
STREET VARCHAR(30),
ZIPCOD NUMBER(5),
CITY VARCHAR(30),
MEMBER PROCEDURE SET_STREET(C VARCHAR2(25)),
MEMBER FUNCTION GET_STREET RETURN VARCHAR(25)
);
```

Y definir esos métodos:

```
CREATE OR REPLACE TYPE BODY TADDRESS AS
--
MEMBER PROCEDURE SET_STREET(C VARCHAR2(25)) IS
BEGIN
STREET:= UPPER(C);
END;
--
MEMBER FUNCTION GET_STREET RETURN VARCHAR(25) IS
BEGIN
RETURN STREET;
END;
);
```

Podemos definir constructores:

```
CREATE OR REPLACE TYPE TADDRESS AS OBJECT (
STREET …
MEMBER FUNCTION GET_STREET RETURN VARCHAR2(25),
CONSTRUCTOR FUNCTION TADDRESS(C, CP, L) RETURN SELF AS RESULT);
--
CREATE OR REPLACE TYPE BODY TADDRESS AS
MEMBER ….
```

Podemos definir tablas de objetos:

```
CREATE TABLE Students OF TPERSON (
ID PRIMARY KEY
);
```

Si sacamos el contenido de la tabla, las propiedades de los objetos son las columnas de la tabla:

```
DESC Students;
NAME Null Type
----------------- -------- ------------
```

CONSTRUCTOR FUNCTION TIPO_DIR(C VARCHAR2(25), CP NUMBER, L VARCHAR2(30)) RETURN SELF AS RESULT
BEGIN
SELF.STREET:= C;
SELF.ZIPCOD:= CP;
SELF.CITY:= L;
END;
	ID NOT NULL NUMBER
	NAME VARCHAR2(40)
	ADDRESS TADDRESS
	BIRTH_DATE DATE

El SQL ahora puede manejar objetos: INSERT INTO Students VALUES (1,'ANA GARCIA', TADDRESS('C/ LA PAZ, 1', 46001,'VALENCIA'),'01/01/1980'); O bien: INSERT INTO Students (ID, NAME, ADDRESS, BIRTH_DATE) VALUES (1,'ANA GARCIA', TADDRESS('C/ LA PAZ, 1', 46001,'VALENCIA'),'01/01/1980');	Podemos definir otros tipos complejos: CREATE TYPE TTELEF AS VARRAY(5) OF VARCHAR2(9); CREATE TABLE AGENDA(NAME VARCHAR2,TELEFON TTELEF); INSERT INTO AGENDA VALUES('JUAN', TTELEF('123456789', '456789123') SELECT TELEF FROM AGENDA ; UPDATE AGENDA SET TELEFON=TTELEF('789456123');
Podemos definir tablas anidadas (una tabla incrustada en una celda de otra tabla). Primero definimos una tipo para esa tabla anidada: CREATE TYPE TNESTEDTABLE AS TABLE OF TADDRESS; Luego creamos la tabla contenedora: CREATE TABLE CLIENTS (ID NUMBER , NAME VARCHAR2(25), ADDRESS TNESTEDTABLE) NESTED TABLE ADDRESS STORE AS ADDRESSES; El tipo previamente definido se utiliza para definir la columna contenedora. STORE AS para indicar el nombre físico de la tabla anidada.	Para manejarlas: INSERT INTO CLIENTS VALUES (1,'GARCIA', TADDRESS('C/MAR,1', 46001, 'VALENCIA'), TADDRESS('C/SOL, 12', 46002,'VALENCIA'), TADDRESS('C/AIRE, 12', 46006,'VALENCIA')) SELECT ID, NAME, CURSOR(SELECT TT.STREET FROM TABLE(T.ADDRESS) TT) FROM CLIENTS T; SELECT STREET FROM THE (SELECT T.ADDRESS FROM CLIENTS T WHERE ID=1) WHERE CITY='VALENCIA'
Los objetos creados tienen siempre un OID, una referencia, podemos acceder a esa referencia: CREATE TYPE TEMPLOYEE AS OBJECT (NAME VARCHAR2(30), BOSS REF TEMPLOYEE); CREATE TABLE EMPLOYEES OF TEMPLOYEE; INSERT INTO EMPLOYEES VALUES (TEMPLOYEE('GARCIA', NULL); INSERT INTO EMPLOYEES SELECT TEMPLOYEE('ALBA', REF(E)) FROM EMPLOYEES E WHERE NAME='GARCIA'; Inserta la empleada ALBA cuya jefa es GARCIA. SELECT NAME, DEREF(P.BOSS) FROM EMPLOYEES P Devuelve el nombre de cada emplead@ y los datos de su jef@. SELECT REF(E) FROM EMPLOYEES E WHERE NAME='GARCIA' Devuelve la referencia del emplead@ cuyo nombre es GARCIA REF hace referencia a un objeto. BOSS REF TEMPLOYEE: Indica que en la columna BOSS tenemos la referencia (OID) de un empleado.	Se maneja el concepto de herencia a la hora de definir tipos de datos: Podemos crear un tipo basado en otro ya creado: CREATE TYPE TPERSON AS OBJECT (…) NOT FINAL; CREATE TYPE TSTUDENT UNDER TPERSON (… Con sus atributos propios y sus métodos propios En este caso TSTUDENT está basado en TPERSON, por lo que hereda las propiedades y métodos de TPERSON y además puede tener los suyos propios.