

# RECORDATORIO DE CONTENIDOS ADQUIRIDOS EL CURSO PASADO NECESARIOS PARA ESTE MÓDULO

FICHEROS		
Tipos de ficheros		
BINARIOS		TEXTO
<p>Su información se almacena en binario, no se pueden leer directamente necesitamos de un programa editor especial dependiendo del archivo.</p> <p>El número 12 se almacenará en binario utilizando tantos bytes dependiendo del tipo de dato que tenga (entero, float, etc...).</p>		<p>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.</p> <p>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.</p>
<p>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).</p> <p>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 ).</p> <p>Generalmente los ficheros se leen a través de flujos de datos (streams).</p> <p>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.</p>		
<p>Para tratar un fichero en modo ACCESO SECUENCIAL tenemos que:</p> <p>Abrir el fichero</p> <p>Leer del archivo</p> <p>Mientras no fin de fichero</p> <p>    Tratar lo leído</p> <p>    Leer siguiente</p> <p>Fin del bucle</p> <p>Cerrar fichero</p>	<p>Dependiendo de cómo se detecta el fin de fichero podemos tener esto:</p> <p>Abrir el fichero</p> <p>Mientras no fin de fichero</p> <p>    Leer del archivo</p> <p>    Tratar lo leído</p> <p>Fin del bucle</p> <p>Cerrar fichero</p>	<p>Para tratar el fichero en modo ACCESO DIRECTO:</p> <p>Abrir el fichero</p> <p>Mientras queramos tratar información</p> <p>    Leer el registro deseado (a veces habrá que posicionarse previamente en él)</p> <p>    Tratar el registro</p> <p>Fin del bucle</p> <p>Cerrar fichero</p>
Veamos ahora algunas clases que se utilizan.		
VB .NET		JAVA
<ul style="list-style-type: none"> <li>◦ FileStream Para crear flujos y leer y escribir ...bytes</li> <li>◦ BinaryReader y BinaryWriter para leer/escribir cualquier tipo primitivo desde/en un archivo/flujo</li> <li>◦ StreamReader y StreamWriter para leer/escribir caracteres desde/en un archivo/flujo</li> </ul>		<ul style="list-style-type: none"> <li>◦ InputStream/OutputStream para leer/escribir flujos de bytes.</li> <li>◦ Reader/ Writer OutputStream para leer/escribir flujos de caracteres (Unicode)</li> <li>◦ Según la forma de acceso al fichero: <ul style="list-style-type: none"> <li>◦ Acceso secuencial <ul style="list-style-type: none"> <li>• FileInputStream/FileOutputStream</li> <li>• FileReader/Writer</li> </ul> </li> <li>◦ Acceso directo o aleatorio <ul style="list-style-type: none"> <li>• RandomAccessFile</li> </ul> </li> </ul> </li> </ul>

MANEJO DE FICHEROS	
FICHEROS DE TEXTO	
LEER LINEA A LINEA DE UN FICHERO DE TEXTO	
<pre>Dim f As StreamReader = New StreamReader("doc.txt") Dim linea As String While ((linea = f.ReadLine()) &lt;&gt; Nothing)     Console.WriteLine(linea) End While f.close()</pre>	<pre>FileReader f = new FileReader("doc.txt"); BufferedReader l = new BufferedReader(f); String linea; while((linea = l.readLine()) != null)     System.out.println(linea); f.close();</pre>
ESCRIBIR LINEAS EN UN FICHERO DE TEXTO	
<pre>Dim f As StreamWriter = New StreamWriter("doc.txt") For i = 0 To 9     f.WriteLine("Línea número: " &amp; i) Next i f.Close()</pre>	<pre>FileWriter fw = new FileWriter("doc2.txt"); BufferedWriter f = new BufferedWriter(fw); for (int i=0; i&lt;10; i++)     f.write("Línea número: " + x + "\r\n"); f.close(); ¡Ojo! Para el salto de línea lo tienes que incluir en la línea que escribes (" + "\r\n"), o bien después de f.write(); añadir f.newLine();</pre>
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).	
<pre>Dim f As StreamReader = New StreamReader("doc.txt") Dim i As Integer While (i = f.Read()) &lt;&gt; -1     Console.WriteLine(Chr(i)) EndWhile f.close()</pre>	<pre>FileReader f = new FileReader("doc.txt"); Int i; while((i = f.read()) != -1)     System.out.println((char) i); f.close();</pre>
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("nombredelfichero", 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:

Primer registro	Segundo registro	Tercero registro
0	29	30
59	60	89
30 bytes	30 bytes	30 bytes

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:

$$\text{Posición} = \text{Longitud de registro} * (X - 1)$$

Si el registro tiene 30 bytes de longitud, el registro número dos empezará en el byte 30.  $\text{Posición} = 30 * (2 - 1)$

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.

## BASES DE DATOS RELACIONALES

Los datos se organizan en tablas formadas por filas y columnas

- Fila /Registro
- Columna/Campo

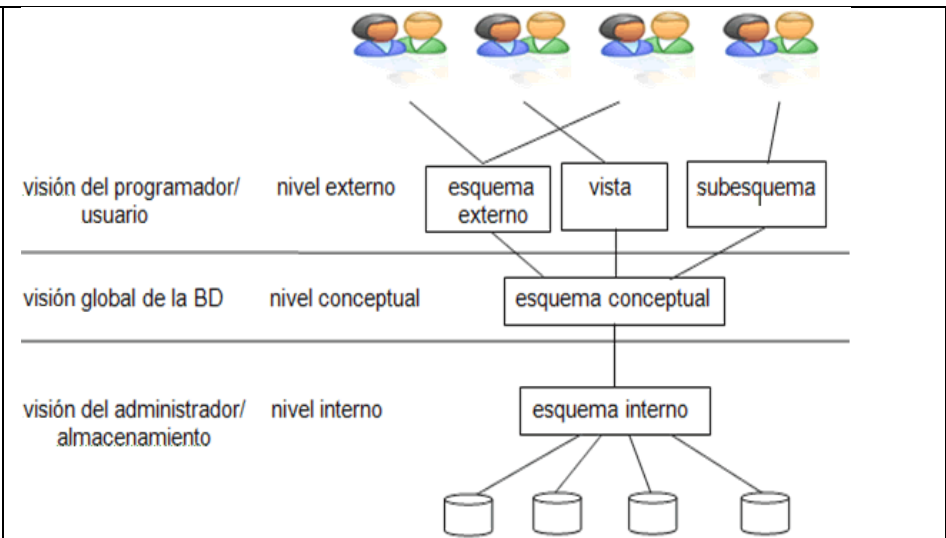
IdFab	Nombre	Direccion
Fab1	Muebles La Madera	Null
Fab2	Maderas Asociados	Avda. del Sol, nave3
Fab5	Industrias Madereras	Pol. Ind 3 fuentes, nave 23

- 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.

- 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.

- 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.

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).



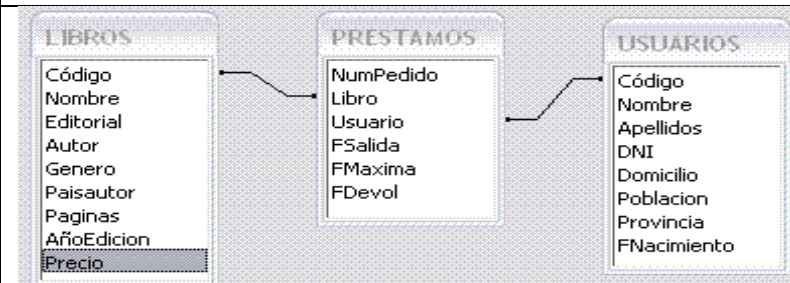
Para poder manejar la información almacenada en la base de datos disponemos de un lenguaje que cumple las reglas de Codd, el lenguaje SQL que veremos en próximos temas.

- 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	
<p>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.</p> <p>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.</p> <p>El lenguaje se ha adaptado también a partir del SQL-1999/SQL-99/SQL3</p> <p>Ejemplo: Oracle</p>	
<p>Se pueden crear tipos nuevos:</p> <pre>CREATE OR REPLACE TYPE <b>TADDRESS</b> AS OBJECT ( STREET VARCHAR(30), ZIPCOD NUMBER(5), CITY VARCHAR(30) );</pre> <p>Y usarlos luego</p> <pre>CREATE OR REPLACE TYPE TPERSON AS OBJECT ( ID NUMBER, NAME VARCHAR(40), ADDRESS <b>TADDRESS</b>, BIRTH_DATE DATE );</pre>	<p>El propio SQL nos permite definir variables y usarlas:</p> <pre>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;</pre>
<p>Podemos tener objetos con métodos:</p> <pre>CREATE OR REPLACE TYPE TADDRESS AS OBJECT ( STREET VARCHAR(30), ZIPCOD NUMBER(5), CITY VARCHAR(30), <b>MEMBER PROCEDURE SET_STREET(C VARCHAR2(25)),</b> <b>MEMBER FUNCTION GET_STREET RETURN VARCHAR(25)</b> );</pre>	<p>Y definir esos métodos:</p> <pre>CREATE OR REPLACE <b>TYPE BODY</b> 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; );</pre>
<p>Podemos definir constructores:</p> <pre>CREATE OR REPLACE TYPE TADDRESS AS OBJECT ( STREET ... MEMBER FUNCTION GET_STREET RETURN VARCHAR2(25), <b>CONSTRUCTOR FUNCTION ADDRESS(C, CP, L) RETURN SELF AS RESULT</b>); -- CREATE OR REPLACE TYPE BODY TADDRESS AS MEMBER ....</pre>	<p>Podemos definir tablas de objetos:</p> <pre>CREATE TABLE Students OF TPERSON ( ID PRIMARY KEY );</pre> <p>Si sacamos el contenido de la tabla, las propiedades de los objetos son las columnas de la tabla:</p> <pre>DESC Students; NAME Null Type -----</pre>

<b>CONSTRUCTOR FUNCTION TIPO_DIR(C VARCHAR2(25), CP NUMBER, L VARCHAR2(30)) RETURN SELF AS RESULT</b> <b>BEGIN</b> <b>SELF.STREET:= C;</b> <b>SELF.ZIPCOD:= CP;</b> <b>SELF.CITY:= L;</b> <b>END;</b>	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.