Bases de datos (parte 1)

Rodrigo Alexander Castro Campos
Última actualización: 25 de mayo de 2015

Este documento presenta un resumen de la UEA "Bases de datos" de la UAM Azcapotzalco, tal como yo la impartí durante el trimestre 15-P. Este documento está pensado para uso personal, aunque es posible que pueda convertirse en unas notas de curso o que sea tratado como tal.

Durante el curso se ven algunos temas que no están incluidos en el temario oficial, como la codificación de caracteres y el lenguaje PHP. Esto lo hago pues considero que:

- En la práctica, es vital saber al menos un poco de codificación de caracteres. Esto es particularmente cierto al momento de desarrollar sistemas de información y sistemas portables.
- Es necesario aprender a utilizar una base de datos desde un lenguaje de programación para aplicaciones del lado del servidor.

La elección del lenguaje PHP es personal. Por lo menos, parece ser cierto que PHP sigue siendo relevante a pesar del surgimiento de otros lenguajes como Python.

Introducción a la codificación de caracteres

El tema de codificación de caracteres, visto a detalle, es bastante complicado. Y es desafortunado que baste un ejemplo sencillo para mostrar que se debe tener mucho cuidado siempre que se esté trabajando con archivos.

En C++ existen dos tipos de archivos reconocidos por `fopen`: archivos de texto y archivos binarios. Por omisión, un archivo se abre como un archivo de texto pero se puede indicar el modo "`b`" al momento de abrirlo:

```
auto f1 = std::fopen("archivo.txt", "r");   // archivo de texto
auto f2 = std::fopen("archivo.txt", "rb");  // archivo binario
```

Supondremos que el archivo en cuestión tiene el siguiente contenido:

```
hola↵
adios
```

Ahora escribiremos un programa que cuente cuántos caracteres tiene el archivo en ambos modos (usando `fgetc` hasta que se devuelva `EOF`):

```
#include <cstdio>

int main( )
{
   auto f = std::fopen("archivo.txt", "r");  // o "rb"
   auto cuenta = 0;
```

```
    while (std::fgetc(f) != EOF) {
        ++cuenta;
    }

    std::printf("%d", cuenta);
}
```

Al ejecutar el programa anterior obtendremos resultados diversos dependiendo del sistema operativo. En Windows obtendremos una cuenta de 10 caracteres para el modo de texto y 11 caracteres para el modo binario, mientras que en Linux/UNIX obtendremos una cuenta de 10 caracteres en ambos casos.

La razón es sencilla y simultáneamente molesta: la codificación de un fin de línea puede variar entre sistemas operativos. En Windows, un fin de línea se codifica con la pareja de caracteres `\r\n` (retorno de carro y salto de línea) mientras que en versiones viejas de Mac se utiliza el caracter `\r` y en Linux se usa el caracter `\n`. Esto explica por qué algunas veces en Windows es problemático leer archivos de texto que fueron creados en Linux, pues los editores de texto de Windows no reconocerán como fin de línea al caracter `\n` en solitario[1].

Cuando un archivo es abierto en modo texto, las funciones de lectura y escritura en archivos (como `fgetc` y `fputc`) realizan de manera transparente la conversión entre el caracter `\n` (la representación de un fin de línea al estilo UNIX) y la representación utilizada por la plataforma para fines de línea. Dicha conversión no se realiza para archivos binarios. Desafortunadamente, la función de posicionamiento en archivos `fseek` no se comportará bien en archivos de texto: ésta se especifica en términos de bytes y es ajena a la conversión antes mencionada.

Codificación de caracteres, ASCII y ASCII extendido

ASCII es una codificación de caracteres que incluyen el alfabeto inglés y muchos símbolos de puntuación comunes . La codificación ASCII define exactamente 128 caracteres (ver figura 1).

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Figura 1 - Tabla del ASCII

[1] En el editor de texto Notepad++ es posible mostrar todos los caracteres de un archivo en "Ver" / "Mostrar símbolo". También es posible cambiar la codificación de una nueva línea en "Editar" / "Conversión fin de línea".

Como es frecuente en computación, cada caracter tiene asociado un valor entero que es usado para almacenar dicho caracter. La codificación ASCII mapea el conjunto de caracteres en cuestión a un rango entero de 0 a 127 inclusivo. Esto quiere decir que bastan 7 bits para poder almacenar un caracter ASCII.

Por otro lado, un caracter ASCII comúnmente es almacenado en un `char` de 8 bits. Ya que un `char` sin signo puede almacenar valores enteros de 0 a 255 inclusivo, una pregunta natural es cómo se utiliza en la práctica el rango de 128 a 255 que ASCII no contempla ni utiliza. Una codificación de caracteres que es compatible con ASCII y que además asigna caracteres en el rango libre de 128 a 255 se denomina codificación ASCII extendida.

Existen decenas de codificaciones ASCII extendidas y es común que la codificación que se use en una comunidad sea la que más le convenga a la misma. Por ejemplo, la codificación ASCII extendida llamada Latin1 o ISO/IEC 8859-1 incluye en el rango libre a la letra *ñ* y a las vocales acentuadas, lo cual es muy útil para los hispanohablantes (ver figura 2). La codificación llamada ISO/IEC 8859-7, por otro lado, no contiene *ñ* ni vocales acentuadas pero sí caracteres griegos: el caracter 241 en ISO/IEC 8859-1 es la *ñ* mientras que en ISO/IEC 8859-7 es la *ρ* (ver figura 3).

F_	ð 00F0 240	ñ 00F1 241	ò 00F2 242	ó 00F3 243	ô 00F4 244
	_0	_1	_2	_3	_4

Figura 2 - Algunos de los caracteres de ISO/IEC 8859-1 con sus valores numéricos

F_	π 03C0 240	ρ 03C1 241	ς 03C2 242	σ 03C3 243	τ 03C4 244
	_0	_1	_2	_3	_4

Figura 3 - Algunos de los caracteres de ISO/IEC 8859-7 con sus valores numéricos

Aunque ambas codificaciones son compatibles con ASCII (rango de 0 al 127), es claro que usuarios de diferentes partes del mundo tendrán problemas al momento de intercambiar información si usan distintas codificaciones para otros caracteres.

Conjunto universal de caracteres, caracteres y caracteres anchos

El conjunto universal de caracteres ISO/IEC 10646 (UCS por sus siglas en inglés) es un conjunto estándar de caracteres que pretende incluir todos los caracteres (útiles o históricamente interesantes) inventados por el hombre. Cada uno de los caracteres o símbolos definidos en el estándar tiene asociado un nombre y un valor entero denominado *code point*. En 1999, este estándar tenía definidos 49259 símbolos, aunque para 2014 ya se encuentran definidos 113021 caracteres. Los caracteres en el estándar no están asignados en un rango contiguo de enteros (existen subrangos reservados), por lo que el espacio de *code points* actual va del *code point* 0 al 1114111 (10FFFF en hexadecimal) inclusivo. Los primeros 128 caracteres y *code points* son compatibles con la codificación ASCII. Una lista de todos los caracteres actualmente definidos se puede consultar en http://www.fileformat.info/info/unicode/block/index.htm

El tipo `char` de C fue diseñado para almacenar caracteres de un conjunto básico definido por el estándar de C, el cual es un subconjunto del ASCII. Por esta razón, a `char` le bastan 8 bits (aunque claramente son insuficientes para poder almacenar cualquier caracter del UCS). Por otro lado, el tipo `wchar_t` de C y C++ es denominado un tipo de caracter ancho y está pensado para poder representar los caracteres de un conjunto mucho más amplio que ASCII (un conjunto de caracteres extendido) y el cual depende de la plataforma. El tamaño de un `wchar_t` también depende de la plataforma: en Windows éste ocupa 16 bits pues hasta 1999 eran suficientes para almacenar cualquier caracter del UCS; en Linux un `wchar_t` generalmente ocupa 32 bits.

Tanto C como C++ definen adicionalmente los tipos de caracter `char16_t` y `char32_t`, los cuales no tienen signo y tienen un tamaño (no dependiente de la implementación) de 16 y 32 bits respectivamente. Esto abre la posibilidad de usar siempre el tipo `char32_t` para poder almacenar cualquier caracter existente en el UCS. Sin embargo, usar siempre `char32_t` cuadruplicaría el consumo de memoria, uso de ancho de banda o almacenamiento en disco duro en comparación a usar `char`, especialmente si no se utilizan caracteres "extraños" en un documento. Debemos recordar que, a final de cuentas, los caracteres más usados en la cultura occidental se encuentran incluidos en el ASCII.

El estándar de codificación Unicode

El estándar Unicode es un estándar que actualmente va de la mano con el conjunto universal de caracteres: ambos definen los mismos símbolos. Sin embargo, mientras que el conjunto universal de caracteres se ocupa de definir cuáles son los caracteres existentes y asignarles nombre y *code point*, Unicode se encarga de definir diversas maneras de codificar los mismos. Unicode define tres codificaciones importantes: UTF-8, UTF16 y UTF32; éstas usan unidades de código (*code units*) de 8, 16 y 32 bits respectivamente. Hablaremos de esto a continuación

UTF-8

La codificación UTF-8 es por mucho la más popular, y usa *code units* de 8 bits. Tiene la propiedad de ser indistinguible de ASCII cuando un documento contiene únicamente caracteres definidos en éste (caracteres con *code points* en el rango de 0 a 127 inclusivo). Cuando un caracter tiene un *code point* fuera del rango del ASCII, entonces es necesario usar más de un *code unit* para representarlo:

```
// código fuente en UTF-8
const char* s1 = "hola";
std::printf("%s", std::strlen(s1));     // imprime 4

const char* s2 = "niño";
std::printf("%s", std::strlen(s2));     // imprime 5
```

En el ejemplo anterior, la cadena `s2` se codifica internamente como (110, 105, 195, 177, 111, 0) donde `'n'` = 110, `'i'` = 105, `'ñ'` = (195, 177), `'o'` = 111 y el terminador = 0. En general, los caracteres en UTF-8 se codifican como sigue:

Rango	Bytes	Byte 1	Byte 2	Byte 3	Byte 4
0 - 127	1	0XXXXXXX			
128 - 2047	2	110XXXXX	10XXXXXX		
2048 - 65535	3	1110XXXX	10XXXXXX	10XXXXXX	
65536 - 1114111	4	11110XXX	10XXXXXX	10XXXXXX	10XXXXXX

La *ñ* tiene *code point* 241 en UCS y por lo tanto necesita dos bytes para ser codificada en UTF-8: el entero 241 es **11110001** en binario, por lo que ésta queda codificada como 11000**11** 10**110001**.

Inicialmente pareciera un tanto extraña la codificación de un caracter en UTF-8, en particular los bits fijos, pero ésta tiene un propósito: el primer byte incluye implícitamente la información de en cuántos bytes consiste la codificación completa de un caracter; la peculiar codificación de los bytes de cola permiten determinar fácilmente que éstos no son bytes de inicio.

UTF-8 es un ejemplo de lo que se conoce como una codificación de longitud variable: no todos los caracteres ocupan el mismo espacio. Un caracter en UTF-8 puede ocupar entre 1 y 4 *code units* (bytes u octetos en este caso). Esto tiene algunas desafortunadas consecuencias:

```
const char* s = "niño";
std::printf("%s", std::strlen(s));      // imprime 5
     // conceptualmente son 4 caracteres, pero strlen calcula bytes
char c = s[3];
     // el tercer caracter es la 'o', pero no así el tercer byte
```

Es ilegal intentar codificar un caracter con una cantidad de bytes que no corresponde con la especificación dada. Por ejemplo, se podría intentar codificar el símbolo @ (*code point* 64) usando dos bytes y llenando los bits sobrantes con cero, pero esto es un error y las bibliotecas Unicode deben reportarlo como tal.

UTF-16

La codificación UTF-16 es, al igual que la codificación UTF-8, una codificación de longitud variable. Sin embargo, en esta codificación cada *code unit* ocupa 16 bits, por lo que invariablemente utiliza más espacio que UTF-8 cuando todos los caracteres son ASCII:

```
char16_t s[] = u"hola";                 // codificación utf-16
std::printf("%d", sizeof(s));           // imprime 10
// cuatro caracteres más el terminador pero usando dos bytes por
// caracter
```

La codificación de caracteres en UTF-16 sigue una serie de reglas ligeramente más complicadas que la codificación UTF-8. Lo que vale la pena ser mencionado es que un caracter con *code point* en el rango de 0 a 65535 puede codificarse con un único *code unit* (16 bits en UTF-16) y un caracter con un *code point* mayor a 65535 necesita dos *code units* (32 bits en total). Esto quiere decir que algunos caracteres ocupan menos espacio en UTF-16 que en UTF-8: la letra japonesa あ tiene *code point* 12354, por lo que necesita 3 bytes en UTF-8 pero 2 bytes en UTF-16.

UTF-32

La codificación UTF-32 es una codificación de ancho fijo: cada *code unit* ocupa 32 bits por lo que todos los caracteres del UCS pueden codificarse en un solo *code unit*.

UTF-16, UTF-32 y *endianness*

Para codificaciones donde cada *code unit* ocupa más de un byte, es necesario hacer una observación adicional: es usual que UTF-16 se almacene en `char16_t` y que UTF-32 se almacene en `char32_t`, pero no todos los procesadores almacenan los bytes de estas variables en el mismo orden: un procesador *big-endian* almacenará un entero de dos bytes (un `char16_t`) con el valor

2580 como la secuencia de bytes (10, 20) de modo que 10 * 256 + 20 = 2580, mientras que un procesador *little-endian* almacenará el mismo valor como la secuencia (20, 10) (con los bytes en orden inverso, primero el byte menos significativo). La versión *big-endian* de UTF-16 se denota como UTF-16BE mientras que la *little-endian* se denota como UTF-16LE; lo análogo ocurre para UTF-32.

Para que un programa pueda identificar qué *endianness* se está usando en un documento que use UTF16 o UTF32, éste debe comenzar con un caracter-marca llamado BOM (*byte order mark*): en UTF16 la secuencia (254, 255) indica que se está usando una codificación *big-endian* mientras que la secuencia (255, 254) indica el uso de *little-endian*. En UTF32, *big-endian* y *little-endian* tienen un BOM de (0, 0, 254, 255) y (255, 254, 0, 0) respectivamente.

Introducción al lenguaje PHP

El lenguaje PHP es un lenguaje de programación que es:

- Interpretado: el código fuente debe ser entregado a un programa (`php.exe` o similar en Windows) el cual lo procesa y inmediatamente después lo ejecuta, sin emitir un archivo ejecutable intermedio.
- Dinámico: el tipo de una variable puede cambiar a lo largo de la ejecución del programa.
- Débilmente tipado: se permite que una variable de cierto tipo se comporte como si fuera de otro dependiendo del contexto.

El código fuente de un programa en PHP puede estar incrustado en un archivo arbitrario. El intérprete buscará las marcas de inicio `<?php` y fin `?>` e interpretará todo lo que esté dentro de ellas. De todos modos, la extensión usual de un archivo con código PHP es `.php`; el típico primer programa en PHP es

```
<?php
   echo "hola mundo";
?>
```

Al igual que en los lenguajes basados en C, se usa el `;` para especificar el fin de una sentencia (excepto cuando una sentencia termina en llave de cierre). La sentencia `echo` permite enviar datos a la salida; se pueden mandar varios valores a la salida separándolos por comas:

```
echo "hola", 5, "gatito";     // imprime "hola5gatito"
```

Muchas de las secuencias de escape de C (como `\n`, `\r`, `\t`) también existen en PHP.

En PHP, una variable se define mediante el símbolo `$` seguido de un identificador. El tipo de la variable no se especifica, sino que se deduce de su inicializador:

```
$n = 5;              // $n es un int
$f = 3.1416;         // $f es un float
$s = "hola";         // $s es un string (cadena)
$b = true;           // $b es un bool
```

Una variable también puede ser nula si se inicializa con `null`. Como ya se ha mencionado, una variable puede cambiar su tipo:

```
$n = 5;              // $n es un int
$n = "adios";        // $n ahora es un string
```

La longitud en bytes de una cadena se puede obtener con la función `strlen`:

```
$s = "hola";
echo strlen($s);     // imprime 4
$s = "";             // $s ahora tiene la cadena vacía
echo strlen($s);     // imprime 0
```

En PHP no existe un tipo caracter, sino que se usan cadenas de longitud 1. Se puede acceder a los bytes individuales de una cadena y esto devuelve la cadena de longitud 1 correspondiente:

```
$s = "hola";
$c = $s[0];          // $c contiene la cadena "h"
```

Se puede intentar modificar un byte individual de una cadena, pero esto no siempre funciona intuitivamente:

```
$s = "caso";
$s[3] = "a";         // ahora $s vale "casa"
$s[0] = "pwyz";      // ahora $s vale "pasa" (se toma sólo la "p")
$s[1] = "";          // ahora $s vale "p\0sa" (no se borra el
                     // caracter, se reemplaza por un nulo)
```

Dos cadenas pueden concatenarse con el operador `.` de PHP:

```
$s1 = "hola";
$s2 = "gatito";
$s3 = $s1."@".$s2;  // $s3 vale "hola@gatito";
```

Una característica muy útil de PHP es que el valor de una variable se puede incrustar en una cadena de la siguiente manera:

```
$n = 5;
$s = "tengo $n gatitos";      // $s vale "tengo 5 gatitos"
```

Una literal de cadena también se puede escribir con comillas simples. Cuando se usan comillas simples, las variables no se inscrustan y las secuencias de escape no se interpretan (excepto `\'`):

```
$n = 5
$s = 'tengo $n gatitos\n';    // $s vale 'tengo $n gatitos\n'
                              // (no incrusta $n, no se evalúa el
                              // salto de línea)
```

Al igual que C++, PHP tiene comentarios de bloque `/* ... */` y comentarios de línea `//`. También existen comentarios de línea con `#`.

La función `var_dump` manda a la salida una descripción del valor o valores dados como argumentos. Dada la naturaleza dinámica de PHP, ésta es extremadamente útil al momento de hacer depuración:

```
$n = 5;
$s = "hola";

var_dump($n);        // int(5)
var_dump($s);        // string(4) "hola"
```

El tema de arreglos (que también es un tipo de dato proporcionado por el lenguaje) se tocará posteriormente y a profundidad; en PHP este tipo es por mucho el más importante.

Introducción al entorno de PHP

Para los ejemplos siguientes, supondremos que EasyPHP DevServer se encuentra instalado correctamente. Existen dos formas de ejecutar un programa en PHP: si el código fuente está en `archivo.php` entonces éste se puede ejecutar desde la consola usando la instrucción `php archivo.php` (ver figura 4):

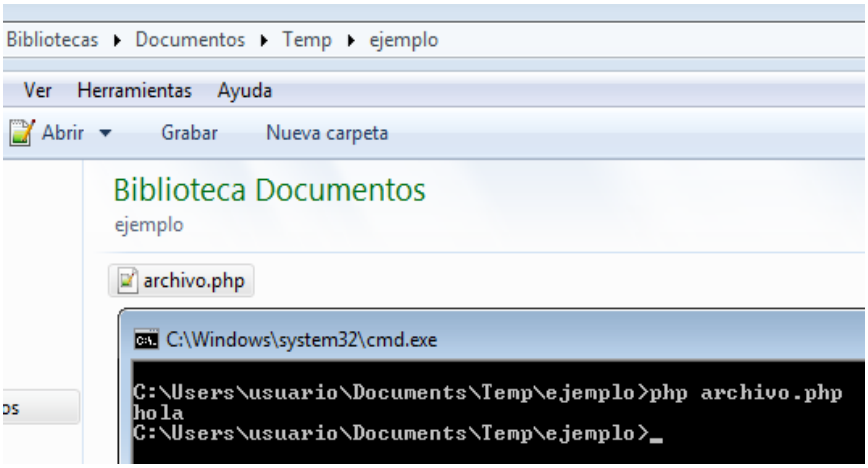


Figura 4 – Ejecutando un programa PHP desde la consola

La otra manera de ejecutar un programa en PHP es usando un servidor web que esté en ejecución. Suponiendo que `archivo.php` se encuentra en la raíz de los documentos web de un servidor local, entonces podemos dirigirnos a la dirección `http://localhost/archivo.php` (ver figura 5):

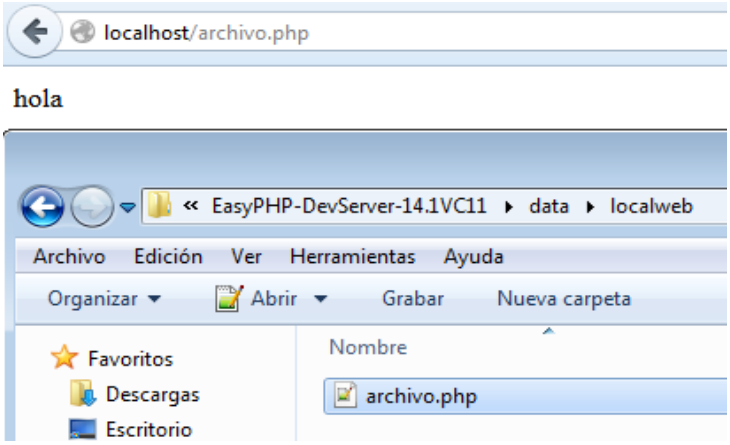
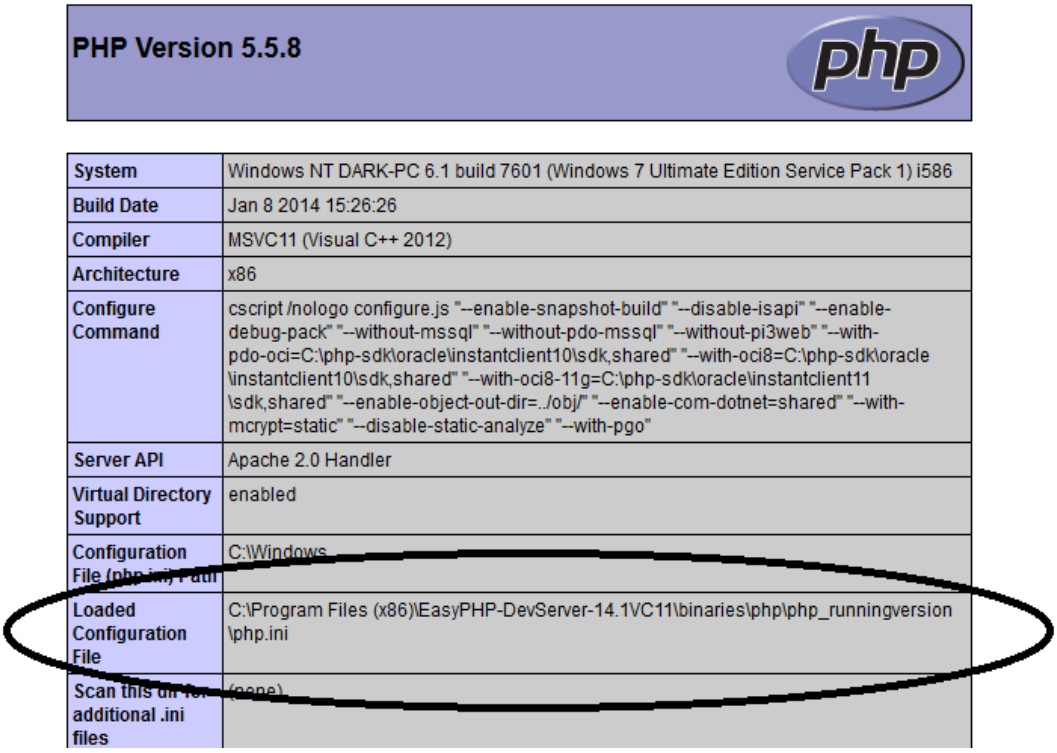


Figura 5 – Ejecutando un programa PHP desde el servidor web
En EasyPHP, la raíz de los documentos web está en la subcarpeta /data/localweb de la instalación

Una manera de examinar la configuración completa de PHP es usando la función `phpinfo` (ver figura 6):

```
<?php
   phpinfo( );
?>
```



PHP Version 5.5.8

System	Windows NT DARK-PC 6.1 build 7601 (Windows 7 Ultimate Edition Service Pack 1) i586
Build Date	Jan 8 2014 15:26:26
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\Program Files (x86)\EasyPHP-DevServer-14.1VC11\binaries\php\php_runningversion\php.ini
Scan this dir for additional .ini files	(none)

Figura 6 – Información desplegada por `phpinfo` en un servidor web

Es de particular interés la versión del lenguaje, la ruta al archivo de configuración php.ini y la codificación por omisión que usa el intérprete. A no ser de que la opción `zend.multibyte` esté habilitada (por omisión no lo está), la codificación de las literales de cadena en un programa PHP es la usada por el archivo fuente. A su vez, la codificación anunciada por el intérprete corriendo en un servidor web es la codificación por omisión del sistema (generalmente ISO 8859-1).

Lo anterior puede crear problemas fácilmente: supongamos que un archivo fuente PHP está codificado en UTF-8 y envía una cadena a la salida, pero que la codificación por defecto del sistema es ISO 8859-1; el servidor web anunciará que la codificación de la salida desplegada es también ISO 8859-1 pero la cadena no se mostrará correctamente (ver figuras 7 y 8):

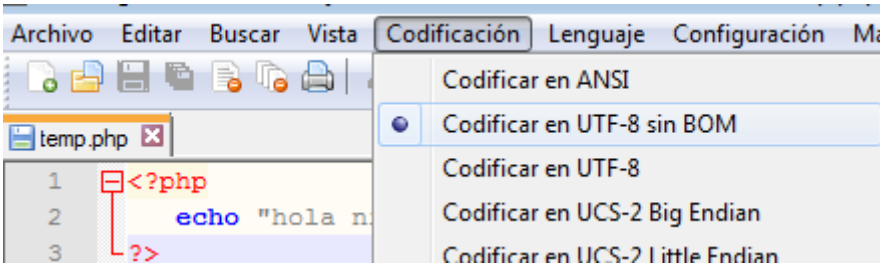


Figura 7 - Código fuente PHP en UTF-8 (imprime "hola niño")

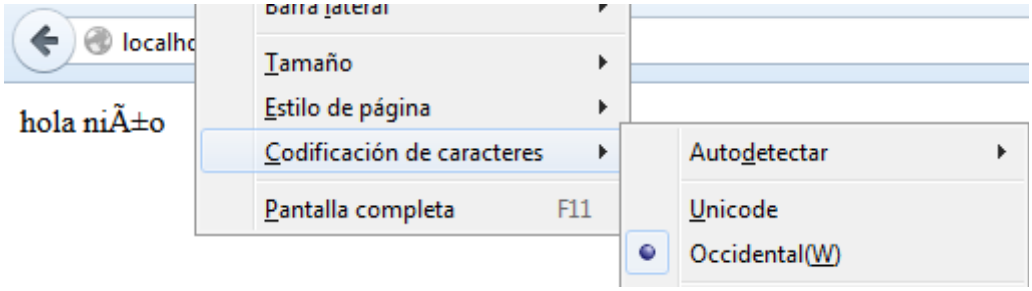


Figura 8 - El explorador Firefox utilizando la codificación ISO 8859-1 para mostrar la salida

Cabe mencionar que cuando se está corriendo PHP del lado de un servidor web, un salto de línea ordinario `\n` no se desplegará como tal, pues el explorador creerá que el servidor está emitiendo código html (`\n` no es un salto de línea en html). Para que se despliegue un salto de línea en el explorador, se deberá usar la etiqueta `<br>` de html (enviada como cadena a la salida):

```
echo "hola<br>";
```

Arreglos en PHP

Un arreglo en PHP es un arreglo asociativo *clave* => *valor* en donde la clave puede ser un valor numérico o una cadena. Un arreglo en PHP también es dinámico, es decir, se le pueden agregar o quitar elementos en cualquier momento. Un arreglo se declara con la siguiente sintaxis:

```
$arr = [ ];          // $arr es un arreglo vacío
$arr = array( );     // sintaxis alternativa
```

Se puede agregar un elemento a un arreglo indicando la clave o posición del mismo y el valor a guardar. La función `count` devuelve el número de elementos del arreglo:

```
$arr = [ ];               // inicialmente vacío
$arr[5] = 12345;          // se guarda "hola" en la clave 5
$arr["idioma"] = "ES";    // se guarda "ES" en la clave "idioma"
echo count($arr);         // imprime 2
```

Un arreglo en PHP no necesita tener elementos en índices enteros consecutivos. Incluso se pueden usar índices negativos:

```
$arr = [ ];
$arr[8] = "hola";
$arr[-5] = "adios";
```

Se puede usar `unset` para eliminar una entrada de un arreglo:

```
$arr = [ ];               // inicialmente vacío
$arr[3] = "hola";         // se guarda "hola" en la clave 3 de $arr
unset($arr[3]);           // el arreglo vuelve a estar vacío
```

De manera similar, `isset` sirve para determinar la existencia de una variable y también la existencia de un elemento de un arreglo. Esto es particularmente útil pues las claves existentes en un arreglo de PHP no son tan predecibles como las de un arreglo de C (todas enteras en un rango):

```
$b = isset($var);   // $var no ha sido definida, $b es false
$var = 5;           // $var acaba de ser definida
$b = isset($var);   // $var no ha sido definida, $b es true

$arr = [ ];
$b = isset($arr[0]);     // no existe la clave 0, $b es false
$arr[0] = "hola";        // se guarda "hola" en la clave 0 de $arr
$b = isset($arr[0]);     // ya existe la clave 0, $b es true
```

Desafortunadamente, `isset` también devuelve `false` si la variable o elemento del arreglo existe pero es nulo:

```
$var = null;
$b = isset($var);        // $var existe pero es nula, $b es false

$arr = [ ];
$arr[0] = null;
$b = isset($arr[0]);     // la clave 0 existe pero el valor es
                         // nulo, $b es false
```

En el caso de variables, no hay una manera de diferenciar entre una variable que no existe y una variable que sí existe pero es nula. En el caso de arreglos, se puede usar la función `array_key_exists`.

Un arreglo también se puede definir especificando sus elementos iniciales:

```
$arr = [ 8, 55, -3 ];    // tres elementos en posiciones enteras
                         // consecutivas empezando desde 0
echo $arr[0];       // imprime 8
echo $arr[1];       // imprime 55
echo $arr[2];       // imprime -3
```

También se pueden especificar explícitamente las claves a usar con la notación *clave => valor* como se muestra a continuación:

```
$arr = [ "nombre" => "pablo", -6 => 55, "edad" => 10 ];
echo $arr["nombre"];          // imprime "pablo"
echo $arr[-6];                // imprime 55
echo $arr["edad"];            // imprime 10
```

Se puede insertar un nuevo elemento en el arreglo sin tener que especificar explícitamente la clave. El nuevo elemento se insertará en el siguiente índice entero disponible:

```
$arr = [ ];         // inicialmente vacío
$arr[] = "hola";    // se guarda "hola" en la clave 0
$arr[] = "adios;    // se guarda "adios" en la clave 1
```

Sentencias de control básicas en PHP

En PHP existen las sentencias de control `if`, `else`, `while`, `for`, `do` y `switch`; estos se comportan, en términos generales, igual que en C (un `switch` en PHP admite expresiones no

constantes y expresiones no enteras en los `cases`, los cuales se prueban secuencialmente). En PHP también existen los operadores de asignación compuesta como `+=` y `*=`, los operadores de incremento y decremento `++`, `--` y los operadores relacionales `<`, `<=`, `>`, `>=`, `==`, `!=`.

```
$arr = [ ];
$temp = "";

for ($i = 0; $i < 4; ++$i) {
   $arr["nombre$i"] = $temp;
   $temp .= "z";
}

switch (count($arr)) {
case rand( ):
   echo "volado";
   break;
case 4:
   echo "bien";
   break;
case "hola":
default:
   echo "imposible",
}
```

En PHP también existen los operadores `===` y `!==`, los cuales además comparan los tipos de las variables en cuestión:

```
$a = 5;
$b = "5";

if ($a == $b) {     // la condición será cierta
   echo "iguales ignorando tipos";
}

if ($a === $b) {    // la condición será falsa
    echo "iguales tomando en cuenta tipos";
}
```

Una variable o entrada de un arreglo puede estar o no definida dependiendo de la ejecución del flujo de control de un programa:

```
$arr = [ ];

if (rand( ) == 5) {
   $arr["mensaje"] = "hola";
}

$b = isset($arr["mensaje"]);       // el valor de $b depende del if
```

Probablemente la sentencia de control más importante de PHP es la sentencia `foreach`: ésta itera sobre los valores (o sobre las claves y los valores) de un arreglo:

```
$arr = [ "nombre" => "pablo", "apellido" => "perez" ];

// imprime "pablo perez "
foreach ($arr as $valor) {
   echo "$valor ";
}

// imprime "nombre apellido "
foreach ($arr as $clave => $valor) {
   echo "$clave ";
}
```

La sentencia `foreach` itera sobre una copia de los valores y las claves de un arreglo. Si se necesitan modificar los valores, el operador `&` puede ser usado:

```
$arr = [ 10, 20, 30 ];

foreach ($arr as $valor) {
   $valor += 10;
}

var_dump($arr);          // $arr sigue siendo [ 10, 20, 30 ];

foreach ($arr as &$valor) {
   $valor += 10;
}

var_dump($arr);          // $arr ahora es [ 20, 30, 40 ];
```

Durante el recorrido de un arreglo con `foreach`, es posible usar `unset` para eliminar entradas del mismo, pero se necesitará contar con la clave:

```
$arr = [ 5, 8, 13 ];

foreach ($arr as $clave => $valor) {
   if ($valor % 2 == 0) {          // quitar los valores
      unset($arr[$clave]);         // pares de $arr
   }
}
```

Constantes en PHP

Una constante se define con la función `define`, la cual toma el nombre de la constante a definir (como cadena) y su valor. Las constantes se nombran sin el símbolo de `$`:

```
define("PI", 3.1416);
define("UNIVERSIDAD", "UAM");
echo PI, " ", UNIVERSIDAD;         // imprime "3.1416 UAM"
```

Una constante no se puede inscrustar en una literal de cadena. Tampoco se pueden declarar constantes que sean arreglos:

```
define("PI", 3.1416);          // OK
define("ILEGAL", [ 5, 7 ]);    // error, una constante no puede ser
                               // un arreglo

echo "el valor de pi es PI";   // imprime "el valor de pi es PI"
                               // la constante no se incrusta
```

Funciones en PHP

Una función en PHP se define con la palabra function. No es necesario indicar el tipo de retorno ni el tipo de los parámetros de la función:

```
function saluda($nombre)
{
   echo "hola $nombre\n";
}

function suma($a, $b)
{
   return $a + $b;
}

saluda("pablo");          // imprime "hola pablo"
saluda("pedro");          // imprime "hola pedro"
echo suma(5, 7);          // imprime 12
```

Se pueden recibir parámetros por referencia decorando con el símbolo `&` el parámetro de la función:

```
function incrementa(&$n)
{
   ++$n;
}

$n = 5;
incrementa($n);
echo $n;                  // imprime 6
```

Una función puede declararse esencialmente en cualquier parte, incluso a la mitad de un programa:

```
<?php
   echo "ya comenzamos...\n";

   $a = 1;
   $b = 2;

   function suma($a, $b)
   {
      return $a + $b;
   }

   echo $a + $b;
?>
```

Una función también puede declararse dentro de un bloque que se ejecuta condicionalmente, pero esto puede volverse confuso rápidamente:

```
$n = rand( );                  // valor aleatorio para $n

for ($i = 0; $i < $n; ++$i) {
   function interna( )         // función declarada en un ciclo:
   {                           // si $n > 1 entonces se producirá
      echo "hola";            // un error de redefinición de
   }                           // función en la segunda iteración
}

interna( );          // si $n <= 0, se producirá un error de
                     // "función no definida"
```

Es buena costumbre declarar todas las funciones de un programa al inicio del mismo (y fuera de código condicional). La función `function_exists` ayuda a determinar la existencia o no de una función en caso de que se requiera hacer algo raro o complicado.

Una variable que almacene una cadena puede ser usada como función: el intérprete de PHP buscará una función que se llame igual que el contenido de la cadena:

```
function saluda( )
{
   echo "hola\n";
}

$s = "saluda";       // $s contiene la cadena "saluda"
$s( );               // se invoca la función saluda

$s = "f";
$s( );               // error, la función f no se ha declarado
```

Una función de PHP devuelve `null` si no se devuelve algún otro valor. Esto ocurre al llegar a la llave de cierre o al ejecutar un `return` vacío:

```
function tonta( )
{
   return;           // no se devuelve un valor
}

$v = inutil( );
var_dump($v);        // $v es null
```

Se pueden especificar valores por defecto para los parámetros de una función:

```
function imprime($letrero = "sin mensaje")   // valor por defecto
{
   echo "$letrero\n";
}

imprime( );              // imprime "sin mensaje"
imprime("hola");         // imprime "hola"
```

Además, una función puede recibir más argumentos que los que necesita:

```
function f( )       // sin parámetros
{
   return;
}

f("a", 5, "c");     // enviando tres argumentos, OK en PHP
```

Los argumentos usados en la invocación de una función (obligatorios y extras) pueden ser obtenidos en un arreglo con la función `func_get_args`:

```
function f( )       // sin parámetros
{
   $arr = func_get_args( );

   foreach ($arr as $valor) {
      echo "$valor ";
   }
}

f("a", 5, "c");     // imprime "a 5 c "
```

Clases y objetos en PHP

Una clase en PHP se declara con la palabra `class` y un objeto se crea con la palabra `new`. Un `struct` de C que tenga dos miembros (x, y) se declararía en PHP de la siguiente manera:

```
class punto {
   public $x;
   public $y;
}

$p = new punto( );
$p->x = 5;
$p->y = 7;
```

Debido a que el operador `.` se usa para concatenar cadenas, en PHP se usa el operador `->` para acceder a los miembros de una clase.

Una función declarada dentro de una clase es llamada una función miembro. Una función miembro se invoca usando un objeto de la clase en cuestión y el operador `->`.

```
class ejemplo {
   function saluda( )
   {
      echo "hola";
   }
}

$e = new ejemplo( );
```

```
$e->saluda( );           // imprime "hola"; $e es el objeto usado
                         // para invocar a la función
```

Una función miembro puede acceder a los miembros del objeto usado para invocarla. Dentro de una función miembro, la variable `$this` denota el objeto que invocó a la función:

```
class par {
   public $x;
   public $y;

   function suma( )
     // $this denota al objeto que invocó a la función
   {
      return $this->x + $this->y;
   }
}

$p = new par( );
$p->x = 5;
$p->y = 7;

echo $p->suma( );   // el invocador es $p
                    // imprime 12
```

Cuando una clase declare una función miembro llamada `__construct` entonces la clase la usa como función constructura. El constructor será la función que se ejecute al momento de usar `new`:

```
class par {
   public $x;
   public $y;

   function __construct($p1, $p2)
   {
      $this->x = $p1;    // inicializa los miembros con
      $this->y = $p2;    // los parámetros recibidos
   }
}

$p = new par( );      // error, el constructor tiene dos parámetros
$p = new par(5, 7);      // OK
echo $p->x + $p->y;      // imprime 12
```

Cuando un miembro de una clase es declarado como `private` en lugar de `public`, sólo las funciones miembro pueden acceder a ellos:

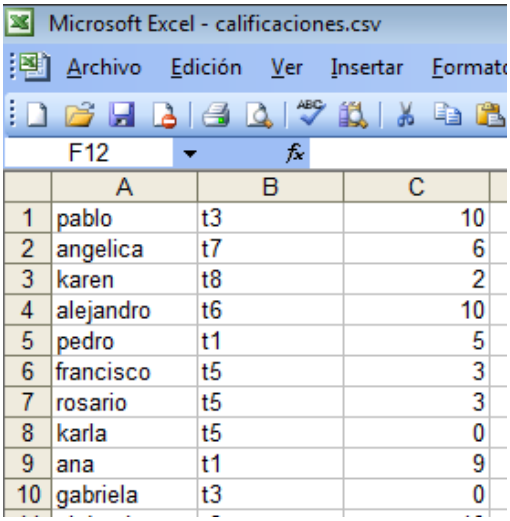
```
class par {
   private $x;      // $x ahora es privado
   private $y;      // $y ahora es privado

   function __construct($p1, $p2)
   {
      $this->x = $p1;         // OK, el constructor es función
      $this->y = $p2;         // miembro
```

```
    }

    function suma( )
    {
       return $this->x + $this->y;    // OK, suma es función miembro
    }
}

$p = new par(5, 7);
echo $p->suma( );          // imprime 12
echo $p->x + $p->y;        // error, no se puede acceder a miembros
                           // privados directamente
```

Manejo de archivos en PHP y modelo jerárquico de bases de datos

PHP cuenta con las mismas funciones de C para manejo de archivos como `fopen`, `fread` y `fwrite`. Sin embargo, nosotros veremos el proceso de serialización de datos ayudado por varias utilidades disponibles en PHP. Trabajaremos en base a los siguientes archivos:

- *calificaciones.csv* es una hoja de cálculo; en el formato *csv* cada fila de la hoja corresponde con una línea del archivo y las columnas están separadas por comas. La hoja de cálculo con la que trabajaremos es una tabla *alumno – evaluación – calificación*. Ver figura 9.



	A	B	C
1	pablo	t3	10
2	angelica	t7	6
3	karen	t8	2
4	alejandro	t6	10
5	pedro	t1	5
6	francisco	t5	3
7	rosario	t5	3
8	karla	t5	0
9	ana	t1	9
10	gabriela	t3	0

Figura 9 – La hoja de cálculo con los datos *alumno – evaluación – calificación*

- *calificaciones.dat* es un archivo creado (de alguna manera) después de procesar la hoja de cálculo descrita anteriormente. Este archivo fue creado únicamente con el propósito de poder encontrar rápidamente la calificación asociada a alguna pareja (*alumno – evaluación*).
- *calificaciones_consultar.php* es un programa que muestra cómo encontrar rápidamente la calificación de una pareja dada (*alumno – evaluación*) usando *calificaciones.dat*.

Mostraremos cómo crear el archivo *calificaciones.dat* dado el archivo *calificaciones.csv* y explicaremos cómo es que funciona el programa *calificaciones_consultar.php*.

En PHP, un archivo puede leerse completo en memoria (y almacenándose en una cadena) usando la función `file_get_contents`:

```
$archivo = file_get_contents('calificaciones.dat');
```

La función `explode` de PHP toma dos cadenas y parte la segunda en todos los lugares donde encuentre la primera, devolviendo el arreglo con los pedazos resultantes. Por ejemplo, podemos partir un archivo en base a saltos de línea de la siguiente forma:

```
$archivo = file_get_contents('calificaciones.dat');
$lineas = explode("\n", $archivo);
```

Desafortunadamente, si la codificación de salto de línea del sistema operativo no es exactamente el caracter `\n`, podemos tener algunos inconvenientes (en Windows, el caracter \r seguirá apareciendo). La constante `PHP_EOL` contiene la cadena usada para representar el fin de línea en un sistema operativo dado:

```
$archivo = file_get_contents('calificaciones.dat');
$lineas = explode(PHP_EOL, $archivo);
```

Ya teniendo el arreglo con las líneas del archivo, podemos partir cada línea en base a comas:

```
foreach ($lineas as $actual) {
   $partes = explode(',', $actual);
}
```

Lo que haremos es crear un arreglo multidimensional que, para la pareja de claves (*alumno – evaluación*), guarde la calificación. Según el archivo *csv*, la primera columna corresponde al alumno, la segunda a la evaluación y la tercera a la calificación:

```
$archivo = file_get_contents('calificaciones.dat');
$lineas = explode(PHP_EOL, $archivo);
$datos = [ ];

foreach ($lineas as $actual) {
   $partes = explode(',', $actual);
   $datos[$partes[0]][$partes[1]] = $partes[2];
}
```

Finalmente procederemos a guardar el arreglo `$datos` en un archivo. En C o C++, nosotros tendríamos que decidir y programar cómo guardarlo. La función `serialize` de PHP facilita las cosas: esta función toma un valor y devuelve una cadena que contiene una representación del valor dado (una representación que PHP construye y puede entender). Posteriormente usaremos la función `file_put_contents`, la cual crea un archivo (y si ya existe lo limpia), guarda una cadena en él y cierra el archivo:

```
$representacion = serialize($datos);
file_put_contents('calificaciones.dat', $representacion);
```

La variable puede ser recreada si dicha representación en cadena es pasada a la función unserialize, que es precisamente lo primero que se hace en calificaciones_consultar.php:

```
<?php
     $datos = unserialize(file_get_contents('calificaciones.dat'));
```

```
    echo "calificacion de karen en t2: ";
    echo $datos['karen']['t2'], "\n";

    echo "calificacion de arturo en t3: ";
    echo $datos['arturo']['t3'], "\n";

    echo "calificacion de roberto en t1: ";
    echo $datos['roberto']['t1'], "\n";

    echo "calificacion de alejandra en t5: ";
    echo $datos['alejandra']['t5'], "\n";
?>
```

Para este ejemplo, no todos los alumnos tienen asignadas calificaciones para todas las tareas, por lo que se debe tener cuidado en acceder a algo que sí exista en $datos. Usando `isset` se puede solucionar este inconveniente. Por otro lado, la combinación de `file_get_contents` y `explode` con saltos de línea puede realizarse en un sólo paso usando la función `file` de PHP.

El ejemplo anterior expone un base de datos que usa el modelo jerárquico. Para este modelo, una base de datos es un árbol y se navega en la base de datos navegando sobre el árbol; podemos hacer una analogía con `structs` de C que a su vez guardan `structs` (ver figura 10):

```
struct punto {
   int x;
   int y;
};

struct recta {
   punto p1;
   punto p2;
};

recta r;
```

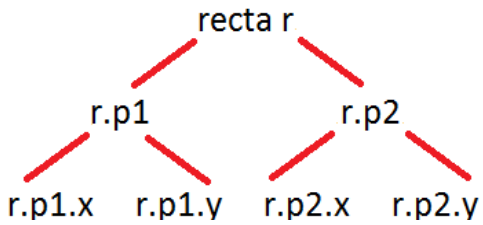


Figura 10 – Los structs anidados forman un árbol con respecto a cómo se acceden a los datos

Es muy fácil usar una base de datos que use el modelo jerárquico, pero este modelo tiene desventajas: si la base de datos se ve como un árbol entonces una entidad del árbol (un nodo) no puede hacer referencia a un nodo que no sea un hijo inmediato (un miembro).

Retomando la analogía con C, esta restricción puede esquivarse fácilmente usando apuntadores (ver figura 11):

```
struct autor;
struct libro;
```

```
struct autor {    // un autor puede escribir a lo mucho dos libros
   char nombre[20];
   libro* obra1;
   libro* obra2;
};

struct libro {      // un libro puede tener a lo mucho dos autores
   char titulo[20];
   autor* autor1;
   autor* autor2;
};

autor a1 = { "hugo", NULL, NULL };
autor a2 = { "paco", NULL, NULL };
libro l1 = { "principito", NULL, NULL };
libro l2 = { "harry", NULL, NULL };

a1.obra1 = &l1;     l1.autor1 = &a1;
// "hugo" escribió "principito"

a1.obra2 = &l2;     l2.autor1 = &a1;
// "hugo" escribió "harry"

a2.obra1 = &l2;     l2.autor2 = &a2;
// "paco" escribió "harry"

printf("%s", a1.obra2->autor2->nombre);
// el nombre del segundo autor de la segunda obra de "hugo"
```

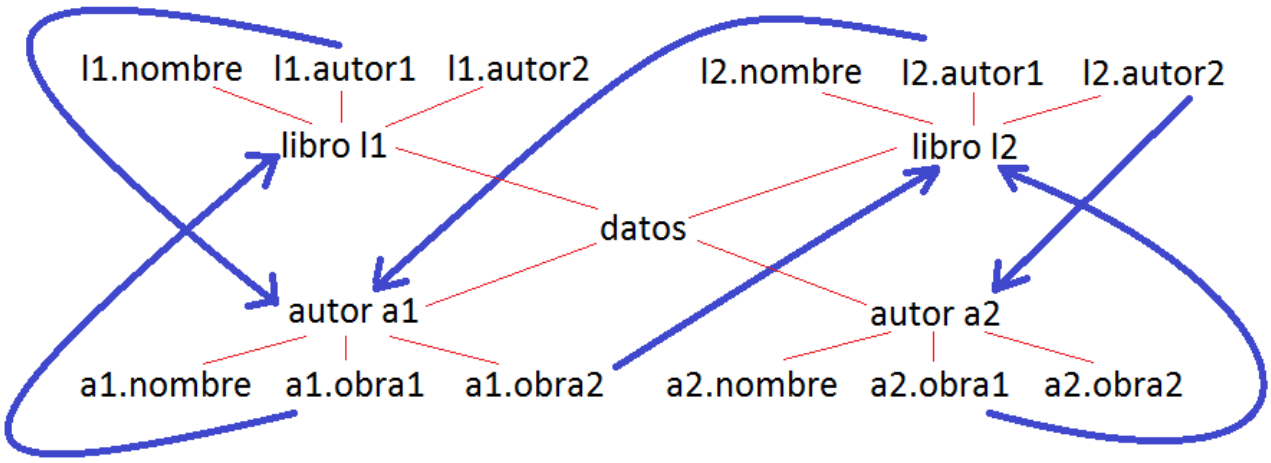


Figura 11 – en rojo las relaciones de acceso directo desde una supuesta raíz llamada *datos*; en azul las relaciones de acceso mediante apuntador

Este modelo es llamado el modelo de red, y su base de datos deja de tener la estructura de un árbol para volverse una gráfica. El problema principal de este modelo no es su poder de representación de la información, sino su complejidad: encontrar un dato en una base de datos que siga el modelo de red típicamente requiere especificar paso a paso cómo navegar de dicha red. El modelo relacional (del cual se hablará posteriormente) es una simplificación de este modelo.

Desventajas en el uso de archivos y gestores de bases de datos

En general es posible abrir un mismo archivo desde varios descriptores simultáneamente. Incluso es posible abrir dicho archivo desde todos los descriptores en modo escritura. A continuación se presenta un programa en C++ que muestra lo antes mencionado (ver figura 12):

```
#include <fstream>

int main( )
{
   auto f1 = std::ofstream("archivo.txt");
   auto f2 = std::ofstream("archivo.txt");

   f1 << "hola" << std::flush;
   f2 << "adios" << std::flush;
   f1 << "hola" << std::flush;
}
```

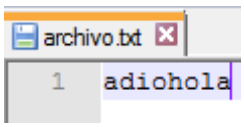


Figura 12 – El contenido final del archivo

El ejemplo anterior hace evidente que una base de datos que esté contenida en un archivo y que sea manejado sin tomar las debidas precauciones enfrentará graves problemas en sistemas concurrentes: si dos usuarios acceden al sistema simultáneamente y dos hilos o procesos distintos modifican el archivo, es altamente probable que eventualmente la base de datos se corrompa.

Peor aún, supóngase una base de datos que brinda servicio a un único usuario pero que tiene información sumamente importante. Ahora supóngase que en la implementación del programa que maneja la base de datos aparecen las siguientes líneas:

```
auto f = std::fopen("archivo.txt", "w");

if (f == NULL) {
   std::printf("ERROR AL ABRIR EL ARCHIVO");
}
else {
   auto s = mensaje_largo( );

   std::fwrite(s, sizeof(char), std::strlen(s), f);
   std::fclose(f);
}
```

Aún en ese pequeño fragmento de código, muchas cosas pueden pasar:

- La función `fwrite` puede fallar si el disco duro tiene espacio insuficiente.
- Si el suministro eléctrico tiene una falla durante la escritura, el archivo puede quedar en un estado inconsistente (por el momento, diremos que un estado inconsistente será aquél que nosotros no esperamos: en este caso podría ser un archivo que no está vacío pero que tampoco contiene todo lo que escribimos en él).

- La función `fclose` puede fallar si `fwrite` no escribió inmediatamente en disco todo lo solicitado, sino que dejó una parte en un búfer de salida (es decir, aún en memoria principal). Es tarea de `fclose` terminar de escribir lo que haya permanecido en el búfer y es posible que, al momento de intentar vaciar el búfer, `fclose` se percate de que el disco ya no tiene espacio.
- Las funciones `fflush` y `fclose` sólo envían el búfer de salida de un `FILE*` al sistema operativo para que éste se encargue de enviarla al disco. El sistema operativo puede decidir no enviar la escritura inmediatamente. La función `fsync` de UNIX/Linux y `FlushFileBuffers` de Windows esperan a que la escritura se haya realizado en disco.

Debemos recordar que muchos de los servicios en la actualidad son completamente dependientes de la tecnología (por ejemplo, la bases de datos de un banco). También debemos recordar que el acceso a disco es aproximadamente un millón de veces más lento que a memoria principal, que los discos o el suministro eléctrico pueden fallar, que existen aplicaciones que deben dar un servicio eficiente a cientos de usuarios simultáneamente y que a pesar de todo es inaceptable perder datos, que éstos se actualicen sólo parcialmente o se corrompan.

Ante estos problemas, rápidamente surgió la idea de implementar un programa cuyo único propósito sea crear bases de datos, garantizando la consistencia y durabilidad de los datos de la misma: es inviable que un programador en solitario intente garantizar estas propiedades partiendo de cero para cada programa que éste implemente.

Un programa se denomina *manejador de bases de datos* si su objetivo es proporcionarle al usuario (al programador) un servicio eficaz de base de datos que éste pueda usar en una aplicación. Actualmente existen muchos manejadores de bases de datos como Oracle, MySQL, MariaDB y FoundationDB. Sin embargo, éstos son el resultado de años de experiencia en la implementación de manejadores de bases de datos: una vez que se acepta la idea de depender de un programa externo para utilizar una base de datos, la pregunta natural que surge es ¿cómo será la comunicación que el usuario tenga con el manejador?, ¿existe alguna interfaz de programación (API) o alguna sintaxis especial que el usuario deba conocer para enviar comandos que el manejador entienda y ejecute?

Como ya se ha comentado, el modelo de red para bases de datos es un modelo poderoso. Desafortunadamente, acceder a la información de una base de datos que use este modelo puede ser tedioso. Retomemos brevemente el ejemplo de (*libros*, *autores*) y supongamos que ahora también contamos con la fecha de publicación de un libro: ¿cómo podríamos calcular cuál fue el autor que más libros publicó en el año 2005? Para obtener este dato, básicamente tendríamos que programar paso a paso cómo navegar sobre la base de datos (que es una gráfica) para recopilar la información necesaria y posteriormente integrarla de modo que podamos calcular el resultado.

Como ya se ha mencionado, el modelo relacional de bases de datos es una simplificación del modelo de red (incluso se podría considerar una simplificación del modelo jerárquico). Este modelo es tan sencillo que los comandos enviados a un manejador de bases de datos relacionales suelen parecer más una descripción de lo que se quiere obtener que una descripción de las tareas que la base de datos debe realizar. Por supuesto que lo sencillo del modelo relacional (y lo sencillo de la comunicación con un manejador de bases de datos de este tipo) tiene tanto ventajas como desventajas, y no pasaremos por alto a estas últimas. A continuación presentaremos este modelo.

Introducción al modelo relacional y al cálculo relacional

Antes de estudiar el modelo relacional con respecto a su uso en los manejadores de bases de datos, lo estudiaremos desde un punto de vista ligeramente teórico y de implementación. Debemos hacerlo

pues es parte del programa oficial de la UEA (cálculo relacional y construcción de bases de datos relacionales), pero también porque este conocimiento nos ayudará a tener una idea relativamente cercana de lo que en verdad se ejecutará dentro de un manejador de bases de datos relacionales.

En la teoría relacional de bases de datos, una *relación* está compuesta por dos partes: el *encabezado* de la relación y el *cuerpo* de la relación. El encabezado es un conjunto de *N atributos* ordenados de la forma (*nombre*: *tipo*) mientras que el cuerpo de la relación es un conjunto de *N*-tuplas donde el *i*-ésimo elemento de una tupla concuerda con el *i*-ésimo tipo del encabezado. Estas definiciones no deben parecernos extrañas pues las hemos estado manejando todo el tiempo:

```
// una relación escrita en C++; no es la notación que usaremos
struct persona {         // encabezado de la relación persona
   std::string nombre;
   std::string apellido;
   int edad;
} p[] = {                // cuerpo de la relación persona
   { "juan", "perez", 10 },
   { "ana", "sanchez", 20 },
   { "pedro", "garcia", 15 }
};

struct coordenada {      // encabezado de la relación coordenada
   int x;
   int y;
} c[] = {                // cuerpo de la relación coordenada
   { 7, -2 },
   { 50, 50 },
   { -16, 32 }
};
```

Lo anterior se puede representar de una manera más amigable:

persona		
nombre: cadena	apellido: cadena	edad: entero
juan	perez	10
ana	sanchez	20
pedro	garcia	15

coordenada	
x: entero	y: entero
7	-2
50	50
-16	32

Al nombre de una relación junto con su encabezado muchas veces se le denomina esquema. Los esquemas de las dos relaciones anteriores los denotaremos como `persona(nombre: cadena, apellido: cadena, edad: entero)` y `coordenada(x: entero, y: entero)` respectivamente.

Usaremos la notación matemática usual para representar las tuplas de una relación: en el ejemplo anterior, el cuerpo de la relación `persona` tiene las tuplas `(juan, perez, 10)`, `(ana, sanchez, 20)` y `(pedro, garcia, 15)`.

Como ya se había mencionado, el modelo relacional es una simplificación de los modelos anteriormente vistos (el jerárquico y el de red). Esto es debido a que se tiene la siguiente restricción:

El tipo de un atributo debe ser un tipo básico (valores numéricos y cadenas).
En particular, el tipo de un atributo no puede ser otra relación.

El equivalente en C sería prohibir `structs` dentro de `structs`, arreglos en general dentro de `structs` y apuntadores. Esta restricción nos obligará a modelar nuestro conjunto de datos de una manera diferente (y a veces incómoda). Nuestro siguiente objetivo será entender cómo nos ayudará esta restricción a comunicarnos fácilmente con un manejador de bases de datos relacionales y qué implicaciones en rendimiento existen. Para lograrlo, implementaremos la lectura y las operaciones básicas entre relaciones.

Implementación de lectura de relaciones

Asumiremos que las relaciones con las que trabajaremos están en archivos en formato *csv*. Además, supondremos que el nombre de la relación es el nombre del archivo, que todos los atributos serán leídos como cadenas (las conversiones implícitas de PHP nos serán útiles en caso de necesitar usar las cadenas como si fueran números) y que los nombres de los atributos están en la primera fila del archivo. Por ejemplo, la relación `persona` se vería como indica la figura 13:

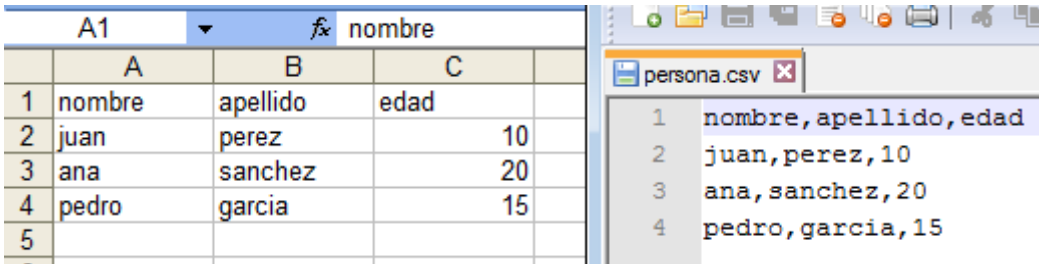


Figura 13 – el contenido de *persona.csv*

El archivo anterior puede ser procesado desde PHP como sigue: leer en memoria el archivo y guardar en un arreglo cada una de sus líneas; en la primera línea está el encabezado de la relación mientras que el resto de las líneas contienen las tuplas del cuerpo de la relación (cada línea debe ser partida por comas para obtener cada atributo o valor por separado).

En la teoría relacional es un error que haya atributos con nombres repetidos. Además, el cuerpo de una relación es un conjunto de tuplas (por lo que tuplas iguales colapsan a una sola). Ambos aspectos son tratados de una manera un poco más flexible en la práctica, pero por el momento seguiremos lo que la teoría indica (ver figuras 14 y 15):

	A	B	C
1	nombre	nombre	edad
2	juan	perez	10
3	ana	sanchez	20
4	pedro	garcia	15

Figura 14 – Una relación inválida (atributos con nombres repetidos)

	A	B	C
1	nombre	apellido	edad
2	juan	perez	10
3	juan	perez	10
4	pedro	garcia	15

Figura 15 – Una relación cuyo cuerpo tiene en realidad sólo dos tuplas (una tupla está repetida)

Proyección, selección y renombramiento en relaciones

Dada una relación R, la operación de *proyección* sobre R devuelve una nueva relación que contiene sólo algunas columnas de R (debemos recordar a diferencia de un *esquema*, una *relación* incluye tanto el encabezado como el conjunto de tuplas del cuerpo). La notación que usaremos será $\pi_{A1,A2,...,AN}(R)$ donde A_i es el nombre de un atributo que se desea conservar en la proyección. Por ejemplo, la relación resultante de $\pi_{nombre,edad}(persona)$ es la siguiente:

$\pi_{nombre,edad}(persona)$	
nombre: cadena	edad: entero
juan	10
ana	20
pedro	15

Si a la relación anterior le aplicamos la proyección π_{nombre} obtendríamos:

$\pi_{nombre}(\pi_{nombre,edad}(persona))$
nombre: cadena
juan
ana
pedro

La notación $\pi_{nombre}(\pi_{nombre,edad}(persona))$ puede confundir a algunos estudiantes. El álgebra relacional permite introducir nombres simbólicos a resultados intermedios, por ejemplo:

```
R1 := π_nombre,edad(persona)        // primer paso
R2 := π_nombre(R2)                  // segundo paso
```

En este caso, R2 es igual a $\pi_{nombre}(\pi_{nombre,edad}(persona))$. Claro está, podemos obtener lo mismo si hubieramos proyectado únicamente el atributo nombre desde el inicio (sin relaciones intermedias):

```
R3 := π_nombre(persona)             // equivalente a R2
```

$\pi_{nombre}(persona)$
nombre: cadena
juan
ana
pedro

Dada una relación R, la operación de *renombramiento* sobre R devuelve una nueva relación que contiene algunos atributos de R renombrados. La notación que usaremos será $\rho_{A/B}(R)$ donde A es el nombre nuevo del atributo B. Por ejemplo, la relación resultante de $\rho_{antigüedad/edad}(persona)$ es la siguiente:

$\rho_{antiguedad/edad}$ **(persona)**		
nombre: cadena	apellido: cadena	antigüedad: entero
juan	perez	10
ana	sanchez	20
pedro	garcia	15

Dada una relación R, la operación de *selección generalizada* sobre R devuelve una nueva relación que contiene sólo las tuplas de R que cumplan una condición. La notación que usaremos será $\sigma_C(R)$ donde C es una condición sobre los valores de una tupla individual. Por ejemplo, la relación resultante de $\sigma_{edad>12}(persona)$ es la siguiente:

$\sigma_{edad>12}$ **(persona)**		
nombre: cadena	apellido: cadena	edad: entero
ana	sanchez	20
pedro	garcia	15

Si quisieramos calcular los nombres de las personas con edad > 12, podríamos hacer lo siguiente:

$\pi_{nombre}(\sigma_{edad>12}$ **(persona))**
nombre: cadena
ana
pedro

Las operaciones de proyección y renombramiento no deberían suponer problemas al implementarlas en PHP. La operación de selección generalizada puede ser implementada enviando como parámetro el nombre de una función (o usar una función anónima) que implemente un predicado sobre una tupla (es decir, que devuelva el valor de verdad de la condición):

```
function edad_mayor12($tupla)
{
   return $tupla["edad"] > 12;
}

$p = lee_relacion("persona.csv");
$s = selecciona($p, "edad_mayor12");
```

Por supuesto, es labor de la función selecciona utilizar el predicado dado correctamente.

Operaciones de conjuntos

En teoría relacional también existe el equivalente de las operaciones de conjuntos como la unión, la intersección y la diferencia. Nosotros definiremos cada una de éstas como operaciones entre dos

relaciones y tendremos como restricción que ambas relaciones tengan exactamente los mismos atributos.

La operación de *unión* toma dos relaciones y devuelve una nueva relación cuyo cuerpo contiene la unión de las tuplas de ambas relaciones:

persona1		
nombre: cadena	apellido: cadena	edad: entero
juan	perez	10
ana	sanchez	20
pedro	garcia	15

persona2		
nombre: cadena	apellido: cadena	edad: entero
carlos	rodriguez	12
olivia	gonzalez	23
pedro	garcia	15

persona1 ∪ persona2		
nombre: cadena	apellido: cadena	edad: entero
juan	perez	10
ana	sanchez	20
pedro	garcia	15
carlos	rodriguez	12
olivia	gonzalez	23

La unión anterior sólo tiene cinco tuplas porque hay una en común entre ambas relaciones (y en la unión ésta sólo debe aparecer una vez).

La operación de *intersección* toma dos relaciones y devuelve una nueva relación cuyo cuerpo contiene la intersección de las tuplas de ambas relaciones:

persona1 ∩ persona2		
nombre: cadena	apellido: cadena	edad: entero
pedro	garcia	15

La operación de *diferencia* toma dos relaciones y devuelve una nueva relación cuyo cuerpo contiene las tuplas que están en la primera relación pero no en la segunda:

persona1 - persona2		
nombre: cadena	apellido: cadena	edad: entero
juan	perez	10
ana	sanchez	20

Producto cartesiano de relaciones (producto cruz)

Dadas dos relaciones R_1, R_2 de m y n atributos respectvamente, el producto cartesiano $R_1 \times R_2$ produce una nueva relación que contiene todos los atributos de R_1 y R_2; el cuerpo de esta relación es el conjunto de todas las parejas (p_1, p_2) donde p_1 es tupla de R_1 y p_2 es tupla de R_2. Por ejemplo:

mascota	
especie: cadena	nombre: cadena
perro	fido
gato	neko

comida	
platillo: cadena	origen: cadena
mole	mexico
kimchi	corea

mascota × comida			
especie: cadena	nombre: cadena	platillo: cadena	origen: cadena
perro	fido	mole	mexico
perro	fido	kimchi	corea
gato	neko	mole	mexico
gato	neko	kimchi	corea

Debemos recordar que la teoría relacional pide que una relación no tenga nombres de atributos duplicados. Si dos relaciones tienen un nombre de atributo en común, entonces habrá problemas al momento de calcular su producto cruz (existirían dos atributos con el mismo nombre, uno que viene de la primera relación y el otro que viene de la segunda relación). Debemos usar renombramiento antes de realizar el producto cruz cuando anticipamos que pueden surgir estos problemas.

Esta operación pareciera ser un tanto rara, y lo es. Sin embargo, en la teoría relacional ésta es una de las operaciones primitivas o fundamentales de la teoría relacional (las demás son la proyección, la selección, el renombramiento, la unión y la diferencia; la intersección puede definirse en términos de uniones y diferencias). Muchas de las operaciones que veremos posteriormente (más intuitivas y útiles) se pueden implementar en términos de las operaciones primitivas.

Para ejemplificar la utilidad del producto cruz, supongamos que queremos calcular el máximo de una relación de valores enteros:

números
valor: entero
7
8
23
1

Si nos limitamos a las operaciones primitivas de la teoría relacional (es decir, sin usar ciclos `for` o cosas por el estilo) parece difícil poder calcular dicho máximo. Éste se puede calcular de la siguiente manera: primero calcularemos el producto cruz de la relación `números` consigo misma (debemos usar renombramiento para nombres de atributos duplicados):

$\rho_{valor1/valor}$(números) × $\rho_{valor2/valor}$(números)	
valor1: entero	valor2: entero
7	7
7	8
7	23
7	1
8	7
8	8
8	23
8	1
23	7
23	8
23	23
23	1
1	7
1	8
1	23
1	1

Posteriormente usaremos la operación de selección bajo la condición `valor1 < valor2`:

$\sigma_{valor1<valor2}$($\rho_{valor1/valor}$(números) × $\rho_{valor2/valor}$(números))	
valor1: entero	valor2: entero
7	8
7	23
8	23
1	7
1	8
1	23

Ya que el producto cruz contenía todas las posibles parejas de números de la relación original, la selección realizada contiene las parejas en las que el primer valor es menor que el segundo valor.

La clave es darnos cuenta que el máximo valor de la relación original jamás será menor que otro. Ya que en la selección anterior la columna `valor1` contiene los valores que fueron menores que algún otro, realizaremos una proyección sobre dicho atributo:

$\pi_{valor1}(\sigma_{valor1<valor2}(\rho_{valor1/valor}(números) \times \rho_{valor2/valor}(números)))$
valor1: entero
7
8
1

Al final una diferencia de conjuntos permite extraer al único valor que nunca fue menor que otro, es decir, el valor máximo:

$números - \rho_{valor/valor1}(\pi_{valor1}(\sigma_{valor1<valor2}(\rho_{valor1/valor}(números) \times \rho_{valor2/valor}(números))))$
valor: entero
23

La introducción de nombre simbólicos para las relaciones intermedias facilita la comprensión. Desafortunadamente, es usual tener que escribir expresiones muy largas al momento de ejecutar consultas en una base de datos relacional.

Junturas

Supongamos que tenemos dos relaciones que tienen atributos en común, por ejemplo:

empresa	
nombre: cadena	marca: cadena
microsoft	lumia
apple	iphone

teléfono	
marca: cadena	software: cadena
lumia	windows
iphone	ios

Ahora deseamos conocer qué marca y qué software maneja cada empresa. Lo anterior lo podemos calcular con las operaciones primitivas de cálculo relacional:

$\rho_{marca/marca1}(\pi_{nombre,marca1,software}(\sigma_{marca1=marca2}(\rho_{marca1/marca}(empresa) \times \rho_{marca2/marca}(teléfono))))$		
nombre: cadena	marca: cadena	software: cadena
microsoft	lumia	windows
apple	iphone	ios

A pesar de que la operación que acabamos de hacer es intuitiva, la expresión resultante en cálculo relacional es bastante larga. Dicho lo anterior, intentaremos definir dicha operación intuitiva olvidándonos temporalmente de los detalles engorrosos de su implementación.

La operación de *juntura natural* es una operación entre dos relaciones que tienen atributos en común: el resultado de esta operación es casi el producto cruz de ambas relaciones, excepto que para cada pareja de tuplas a incluir en el resultado, los valores de los atributos en común deben coincidir exactamente (y como coinciden exactamente entonces cada atributo en común sólo se reportará una vez).

Para mostrar que dicha definición coincide con lo que hicimos anteriormente, mostraremos un producto cruz inválido pero que usaremos como referencia:

empresa × teléfono (*)			
nombre: cadena	marca[1]: cadena	marca[2]: cadena	software: cadena
microsoft	lumia	lumia	windows
microsoft	lumia	iphone	ios
apple	iphone	lumia	windows
apple	iphone	iphone	ios

Ahora eliminaremos las tuplas en donde el valor de los atributos en común (en este caso, sólo el atributo `marca`) no coinciden:

...			
nombre: cadena	marca[1]: cadena	marca[2]: cadena	software: cadena
microsoft	lumia	lumia	windows
apple	iphone	iphone	ios

Ya que los valores de las tuplas que permanecieron coinciden en los atributos en común, podemos reportar sólo uno de ellos:

...		
nombre: cadena	marca: cadena	software: cadena
microsoft	lumia	windows
apple	iphone	ios

Esta última relación es idéntica a la que habíamos calculado anteriormente. La notación de juntura natural entre dos relaciones es $R_1 \bowtie R_2$.

La θ-juntura de dos relaciones, denotada como $R_1 \bowtie_\theta R_2$, es simplemente un producto cruz seguido de una selección que usa el predicado θ para comparar elementos de una tupla resultante del producto cruz.

Cuando el predicado θ de una θ-juntura es una comparación de igualdad, la juntura se denomina equi-juntura. La única diferencia entre la equi-juntura y la juntura natural es que la equi-juntura no necesita que los elementos a comparar tengan el mismo nombre de atributo en las relaciones R_1 y R_2 usadas en el producto cruz.

Ejemplos tipo examen

De Wikipedia "en.wikipedia.org/wiki/Relational_algebra"

The division is a binary operation that is written as R ÷ S. The result consists of the restrictions of tuples in R to the attribute names unique to R, i.e., in the header of R but not in the header of S, for which it holds that all their combinations with tuples in S are present in R. For an example see the tables Completed, DBProject and their division:

Completed

Student	Task
Fred	Database1
Fred	Database2
Fred	Compiler1
Eugene	Database1
Eugene	Compiler1
Sarah	Database1
Sarah	Database2

DBProject

Task
Database1
Database2

Completed ÷ ***DBProject***

Student
Fred
Sarah

Del libro "Database Systems – The complete book" (pág. 89 del PDF – pág 52 del documento)

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

a) What PC models have a speed of at least 3.00?

b) Which manufacturers make laptops with a hard disk of at least 100GB?

c) Find the model number and price of all products (of any type) made by manufacturer B.

d) Find the model numbers of all color laser printers.

e) Find those manufacturers that sell Laptops, but not PC's.

! f) Find those hard-disk sizes that occur in two or more PC's.

```
maker:
   { A, B, C, D, E, F, G, H }
model: entero
type (product):
   { pc, laptop, printer }
speed: real
ram: entero
hd: entero
price: entero
screen: real
color: booleano
type (printer):
   { ink-jet, laser }
```