

Diseño de Bases de Datos Relacionales

Índice

1	Introducción	1
1.1	Para quién es este manual	1
1.2	Qué es una base de datos	1
1.2.1	SQL	2
1.2.2	Usuarios	2
1.2.3	Orientadas a Objetos	3
1.3	Microsoft Access.....	3
1.4	MySQL	3
1.5	Contenido del Manual	4
2	Bases de Datos Relacionales.....	5
2.1	Tablas y Campos.....	5
2.1.1	Tipos de datos y Tamaño	6
2.1.2	Cómo se hace en Access	8
2.1.3	Cómo se hace en MySQL.....	12
2.2	Códigos y Relaciones. Consultas.....	14
2.2.1	Códigos	15
2.2.2	Cómo se hace en Access. Consultas	15
2.2.3	Cómo se hace en MySQL.....	17
3	Diseño de Bases de Datos	18
3.1	Un método de diseño.....	18
3.2	Tablas.....	19
3.3	Campos	19
3.4	Relaciones.....	20
3.4.1	Tipos de relaciones.....	20
3.4.2	Reparando relaciones.....	23
3.4.3	Qué significa una relación.....	23
3.4.4	Creando la biblioteca	24
3.4.5	Árboles de relaciones	27
3.5	Claves	28
3.5.1	Claves principales.....	29
3.5.2	Claves foráneas	30
4	Modelo Entidad/Relación.....	31
4.1	Método de Diseño.....	31
4.1.1	Captura de requisitos.....	31
4.1.2	Esquema conceptual. Modelo E/R	32
4.1.3	Implementación del modelo de datos	32
4.1.4	Implementación de las operaciones	32
4.2	Componentes del Modelo	32
4.3	Entidades.....	33
4.4	Atributos	33
4.5	Relaciones.....	35
4.5.1	Roles	35
4.5.2	Multiplicidad	35
4.5.3	Participación Total	37
4.5.4	Entidades Débiles	37
4.6	El Diagrama E/R	39
4.6.1	Otros Símbolos	39
4.7	Modelo Extendido	40

5 Formas Normales	41
5.1 Definición	41
5.2 Conceptos Básicos	41
5.3 Primera Forma Normal (1NF)	42
5.4 Segunda Forma Normal (2NF)	43
5.5 Tercera Forma Normal (3NF)	44
5.6 Forma Normal de Boyle-Codd (BCFN)	45
5.7 Cuarta Forma Normal (4NF)	46
5.8 Quinta Forma Normal (5NF)	47
6 Access	49
6.1 Tablas	50
6.1.1 <i>Propiedades de los campos</i>	50
6.1.2 <i>Claves</i>	51
6.2 Relaciones	52
6.3 Consultas	53
6.3.1 <i>Criterios</i>	54
6.3.2 <i>Totales</i>	56
6.3.3 <i>Campos calculados</i>	58
6.4 Formularios	60
6.4.1 <i>Crear un formulario con el asistente</i>	60
6.4.2 <i>Diseño y manejo de controles</i>	63
6.4.3 <i>Añadiendo controles</i>	64
6.4.4 <i>Cuadros combinados y cuadros de Lista</i>	65
6.4.5 <i>Subformularios</i>	67
6.4.6 <i>Campos calculados</i>	70
6.5 Informes	72
6.5.1 <i>Crear un informe con el asistente</i>	72
6.5.2 <i>Diseño de un informe</i>	75
7 Introducción a MySQL. Lenguaje SQL	80
7.1 Conceptos Básicos de MySQL	80
7.2 Conceptos Básicos de SQL	81
7.3 Iniciando por primera vez	81
7.3.1 <i>Formato de almacenamiento</i>	85
7.3.2 <i>Elegir una clave</i>	85
7.4 Creación de Bases de Datos, Usuarios y Permisos	86
7.4.1 <i>Create y Drop Database</i>	87
7.5 Creación de Tablas	88
7.5.1 <i>Create, Alter y Drop Table</i>	89
7.6 Operaciones con los datos	91
7.6.1 <i>Insert, Update y Delete</i>	92
7.7 Claves	92
7.8 Relaciones: Claves Foráneas	94
7.9 Consultas. Select	95
7.9.1 <i>Columnas</i>	96
7.9.2 <i>Tablas</i>	98
7.9.3 <i>Where y Having. Condiciones</i>	98
7.9.4 <i>Agrupando datos</i>	100
7.9.5 <i>Ordenar datos</i>	100
7.9.6 <i>Limitar las filas mostradas</i>	101
7.9.7 <i>Consultas a Varias Tablas. Join.</i>	101
7.9.8 <i>Union</i>	104
7.9.9 <i>Subconsultas</i>	104
7.10 Vistas	104

1

Introducción

1.1 Para quién es este manual

Está dirigido a personas sin conocimientos de programación que quieran diseñar y escribir una base de datos, y de hacerlo bien.

Evitaré la profunda, interesante y farragosa base matemática en la que se sustenta el diseño de bases de datos y me centraré en lo práctico; Mi objetivo es diseñar correctamente una base de datos, no aprobar un examen teórico. Aunque por supuesto, habrá teoría...

El huir de las definiciones matemáticas tiene un precio. En diseños complejos puede ser muy útil tener los conceptos exactamente definidos. Pero será raro que te enfrentes a bases de datos de ese tipo. Con lo explicado en este manual podrás diseñar bases de datos de tamaño medio sin problemas.

Supondré que te manejas con un ordenador con cierta soltura. No voy a explicar qué es un fichero, como se pone "Arial 10", ni cómo se maneja el ratón. Pero eso será lo único que necesitas saber.

1.2 Qué es una base de datos

El término "base de datos" no es una definición informática. Sencillamente, es un conjunto de datos: Los archivadores con las tarjetas de cartón de una biblioteca (qué tiempos aquellos), los libros de contabilidad de una empresa, el fichero de texto donde apuntas los discos que tienes, tu agenda de teléfonos.

Sin embargo, un ordenador resulta muy útil cuando hablamos de datos, ya que permite hacer cosas con ellos que de otro modo nos resultarían imposibles o muy costosas:

- Nos permite modificar los datos existentes fácilmente. Acuérdate de las veces que has cambiado un número telefónico de la agenda
- Podemos ordenarlos como queramos. ¿Qué sucede cuando has llenado la letra "J" de la agenda y te presentan a Javier?
- Permite hacer estadísticas y clasificaciones automáticamente. ¿A cuantas personas conozco? ¿A quienes puedo pedirles los apuntes de lengua? (bueno, vale, con una agenda no tiene mucho sentido, pero si se tratase de una lista de clientes y proveedores...)
- Y lo más importante, nos permite RELACIONARLOS. Imagínate que tienes que organizar una cena de clase y llamar por teléfono a todos los demás. Posees una "base de datos" con los teléfonos (la guía telefónica) y otra con los alumnos (la lista de clase), pero la única forma

que tienes de relacionar la información es... buscar en la guía uno por uno. Un ordenador haría exactamente lo mismo, pero lo haría automáticamente.

Por todo esto, cuando hablemos de una base de datos entenderemos bases de datos informáticas. Y concretamente de un tipo especial de base de datos: **Bases de Datos Relacionales**. Existen diferentes tipos de bases de datos, pero sin duda el modelo más utilizado es el Relacional. Prácticamente todas las bases de datos usadas en la actualidad son relaciones: Access, Oracle, MySQL, Postgress, SQL Server... Los otros modelos existentes no eran peores: de hecho, eran más rápidos, pero complicaban mucho el diseño y modificación de una base de datos.

Ya que vamos a trabajar con bases de datos informáticas, un poco de cultura general:

- El motor de la base de datos o **Sistema Gestor de Base de Datos** (SGBD o DBMS en inglés) es el conjunto de programas que se encargan de manejar los datos. El usuario “pedirá” algo (ordenar los datos, borrarlos, seleccionar cierto conjunto) y será el motor quien lo haga más o menos rápido.
- **La base de datos**. Se entiende por base de datos el fichero donde están físicamente almacenados los datos, aunque es un término ambiguo que puede significar muchas cosas.
- Toda base de datos tendrá unas **herramientas de diseño**, que ayudarán al programador a crear o manejar la base de datos

En Access no veremos la diferencia entre el motor de la base de datos y las herramientas de diseño. Todo estará dentro de “Access”, pues al iniciar el ejecutable lo tendremos todo disponible. Sin embargo nuestros datos y diseños se guardarán aparte, en los diferentes “.mdb” que creemos.

Con bases de datos más profesionales, como MySQL, esto no es así; Por un lado estará el motor de la base de datos, por otro los programas de diseño (que suelen pagarse aparte) y por otro la enorme base de datos, con todos los datos de todos los usuarios. En estos sistemas no hay “ficheros independientes” para cada pequeña aplicación. Por supuesto la seguridad (permisos, claves, etc.) es muy superior a la que proporciona Access.

1.2.1 SQL

SQL (Structured Query Language) es un lenguaje muy simple y más o menos estándar para la creación y operación de bases de datos relacionales.

En las bases de datos más profesionales, dirigidas a informáticos, es habitual usar el lenguaje SQL para diseñar y administrar la base de datos. Mientras que en Access todo puede hacerse con el ratón, en estos sistemas se suele trabajar con una consola de comandos en la que se escriben las instrucciones. Están pensadas para ser usadas desde un programa (hecho en Java, por ejemplo). La consola es simplemente para la creación del diseño y pequeñas pruebas.

1.2.2 Usuarios

En una base de datos tradicional se distinguen tres tipos de usuarios o roles:

- **Administrador**. La persona encargada de gestionar la base de datos. En Access no es necesario, pero en sistemas más complejos hay mucha gente usando muchas bases de datos que compiten por los recursos del ordenador. El administrador se encarga de crear o revocar permisos, usuarios, bases de datos, administrar el espacio, instalar actualizaciones, hacer copias, etc.
- **Diseñador**. La persona encargada de crear los diseños de bases de datos en el espacio dado por el administrador.

¹ Informática significa literalmente INFORmación autoMÁTICA.

- **Usuario final.** La persona que va a usar la base de datos. Puede ser alguien sin preparación informática o un programador que va a escribir una aplicación de base de datos.

A menudo los roles se solapan, sobre todo en empresas o diseños pequeños.

1.2.3 Orientadas a Objetos

Las bases de datos orientadas a objetos son el futuro (casi el presente). Se parecen a las relacionales, pero incluyen las características de la programación orientada a objetos en su diseño y funcionamiento.

Si no vas a programar en algún lenguaje orientado a objetos, no es algo que de momento deba preocuparte. De lo contrario echa un vistazo a tecnologías como Java y EJB3. Y aprende primero a diseñar objetos.

1.3 Microsoft Access

Usaremos Microsoft Access para implementar lo que vayamos aprendiendo. ¿Por qué Access? Es barato², fácil de manejar y pueden escribirse bases de datos complejas sin necesidad de escribir ni una línea de programación. Está diseñado para usuarios, no para informáticos, y plasmar un buen diseño resulta sencillo.

También tiene sus desventajas: Comparado con gestores de bases de datos profesionales (Oracle, por ejemplo) es MUY lento. Es muy sencillo escribir aplicaciones estándares, pero si te alejas de la norma, estás obligado a programar casi todo lo que hagas. Y de vez en cuando, se comporta como un auténtico producto Microsoft: Se cuelga.

¿Seguro que las ventajas superan los inconvenientes? Depende del uso. Es ideal para bases de datos pequeñas o medias: Una biblioteca de un colegio, los datos de contabilidad de un negocio, las ventas de una empresa, tu colección de discos, las entradas y salidas de un almacén. Para bases de datos mayores, como el padrón de Vitoria, los resultados serían malos (lentos), aunque todo depende del uso que se vaya a dar a la información. Es muy difícil dar cifras absolutas, pero por experiencia y pensando en una movilidad de los datos “normal”, Access funciona bien con bases de datos de 30.000 registros. Por supuesto, el límite teórico es muy superior. Y repito, todo depende de cómo se vayan a usar esos datos.

El aspecto de las pantallas será diferente en Access 97 ó 2003, pero las diferencias entre uno y otro son mínimas. Todo lo explicado en este manual es totalmente aplicable para ambos.

1.4 MySQL

Y también usaremos MySQL. Es una base de datos bajo licencia GNU GPL y propiedad de Oracle³. ¿Por qué MySQL? Es de las más usadas en Internet, gratuita y bastante rápida. No vas a diseñar ninguna base de datos para la que MySQL se quede pequeña.

Y sobre todo, está pensada para programadores. Es un buen ejemplo de cómo se maneja una base de datos “a la antigua usanza”: Consola de comandos, SQL, permisos, usuarios...

Sólo veremos lo imprescindible. Si quieres saber más, Internet esta lleno de información sobre MySQL. Te recomiendo la página oficial <http://dev.mysql.com/doc>, donde encontrarás manuales en inglés y castellano.

² Puede costar unos 200 € (precios 2008). Con Oracle habríamos de varios miles de euros... por cada ordenador.

³ Es decir, es **gratis** a no ser que quieras vender modificaciones de MySQL. Nunca lo vas a hacer.

Por supuesto, también posee un montón de herramientas gráficas (gratuitas y de pago) para hacer el trabajo más sencillo. No llega al nivel de Access, pero pueden simplificar mucho la vida al administrador de la base de datos.

1.5 Contenido del Manual

Está dividido en dos partes, diseño de una base de datos e implementación de la misma.

El capítulo 2, “Bases de Datos Relacionales” y 3, “Diseño de Bases de Datos” explican lo necesario para crear diseños de bases de datos pequeños o medios, con un poco de práctica. Explican desde los rudimentos básicos hasta los conceptos de relaciones o claves primarias.

El capítulo 4, “Modelo Entidad/Relación” formaliza los conocimientos adquiridos. Se trata de darle nombre a parte de lo que se ha visto en los capítulos sobre diseño. Útil si se desean consultar libros más avanzados.

El capítulo 5, “Formas Normales” amplía los conocimientos sobre diseño de bases de datos. Es necesario aplicar las formas normales (o al menos tenerlas en cuenta) si queremos crear base de datos más complejas.

La última parte del manual enseña cómo implementar un diseño en una base de datos real. En el capítulo 6, “Access” aprenderemos a usar esta base de datos, incluidas las utilidades que proporciona para crear pantallas y listados. En el capítulo 7, “Introducción a MySQL. Lenguaje SQL” estudiaremos la base de datos MySQL y el lenguaje de consultas SQL.

2 Bases de Datos Relacionales

El objetivo de este capítulo es familiarizarnos con el manejo de Access y con los conceptos fundamentales de una base de datos relacional, así como echar un vistazo a MySQL, una base de datos dirigida a programadores. En capítulos posteriores todos estos temas se explicarán con mayor profundidad.

2.1 Tablas y Campos

La idea básica de una base de datos relacional es muy sencilla. Los datos se organizan en **tablas**, con filas y columnas. Supongamos una relación de libros y autores⁴. Queremos saber el escritor del libro, el título, la editorial, el número de páginas (somos muy meticulosos) y el tema. Eso implica una tabla con cinco columnas, una por cada concepto que queremos almacenar. El número de filas depende del número de libros que tengamos:

Escritor	Título	Editorial	Nº de Pag.	Tema
Cervantes	El Quijote	Alfaguara	1200	Literatura
Neuromante	William Gibson	Minotauro	316	Literatura – CF
Paul Auster	El palacio de la Luna	Anagrama	308	Literatura
Toti Martinez	La Calle de la Judería	Abra	490	Literatura – HIST
Paul Auster	Leviatán	Anagrama	266	Literatura
Günter Grass	El tambor de Hojalata	Alfaguara	655	Literatura
P. Auster	La Música del Azar	Anagrama	251	Literatura
Frederik Pohl	Pórtico	Ed. B	363	Literatura – CF
Agustín Froute	Java 2	RaMa	650	Informática
Petzold – Paul Yao	Programación en Windows 95	Mc Graw Hill	1100	Informática
Varios	Historia Universal	Anaya	900	Historia
Kim S. Robinson	Marte Rojo	Minotauro	650	Literatura -CF
Kim S. Robinson	Marte Verde	Minotauro	692	Literatura -CF
Kim S. Robinson	Marte Azul	Minotauro	724	Literatura -CF

⁴ Más adelante se verá que igual podría haber sido clientes y pedidos, proveedores y materias primas o discos y músicos. En el fondo todos los datos presentan la misma estructura

La tabla por cierto, está horriblemente diseñada. Ya la mejoraremos.

Como cualquier otra tabla, se compone de filas, a las que a partir de ahora llamaremos **registros o tuplas** y de columnas, que en diseño de base de datos se denominan **campos**. Una tabla puede tener las filas o registros que se desee, pudiéndose añadir, borrar o modificar cuando convenga.

Con los campos (las columnas), una base de datos es mucho más estricta. Aunque se puede añadir o quitar campos en cualquier momento, no es recomendable hacerlo; más adelante comprobarás que eso da un montón de trabajo, ya que te obligará a modificar en cascada pantallas, listados, consultas e incluso campos de otras tablas.

2.1.1 Tipos de datos y Tamaño

¿Qué se puede escribir en un campo? ¿Cuánto cabe? Del mismo modo que en una hoja de papel dejas más o menos espacio en función del tamaño que **prevés** que necesitarás, en una tabla de una base de datos tienes que indicarle al ordenador cuanto espacio necesitarás para guardar los datos.

El ordenador almacena de forma totalmente distinta los textos y los números. Lo primero que tienes que hacer es decidir si el valor a guardar son letras o números. Lo segundo, el tamaño **máximo** de los mismos. En general, todos los campos tendrán un tamaño fijo, independientemente de lo que contengan. Si decides que vas a reservar treinta letras (bytes) para guardar el título del libro, todos los títulos ocuparán treinta letras, incluso si por un descuido dejas vacío el campo. Evidentemente, el tamaño adecuado será el del libro con el título más largo. No te obsesiones con el tamaño, ya no es importante. Los ordenadores actuales tienen discos muy grandes, y a no ser que vayas a hacer el censo de tu ciudad, dará exactamente igual si reservas cincuenta bytes o cien.

2.1.1.1 Tipos de datos de Access

Los principales tipos de datos soportados por Access son:

Texto	Para escribir textos: "El Quijote", "Fahrenheit 451" ó "900 10 00 10". El tamaño máximo es de 255 letras (bytes).
Memo	Textos de longitud variable . No puedes hacer búsquedas ni ordenar por su contenido (ya verás que eso es muy importante), por lo que su uso es muy limitado: Notas sobre el cliente, opiniones sobre el cliente, etc. Hasta 64K.
Numérico	Valores numéricos.
Fecha/Hora	Para almacenar fechas y horas. Es un número; concretamente el número de días que se han pasado desde el 1/01/1900. Un cuatro sería "cuatro de enero de 1900". Por supuesto, Access se encarga de que aparezca con el aspecto adecuado. La hora son los decimales: Un 4,5 sería "cuatro de enero de 1900 a las doce de la mañana" (0,5 es la mitad del día). Que sea un número es muy útil, pues permite comparar fechas, restarlas...
Autonumérico	Un numérico sin decimales que el usuario no puede cambiar ni escribir. Access lo va escribiendo automáticamente a medida que añadimos líneas a la tabla. Muy útil para códigos (ya veremos qué es eso).
Moneda	Un numérico con decimales especial para valores monetarios: Access hace que aparezca con el aspecto adecuado (pone "pts.", "€", "\$" automáticamente, etc.)
Sí/No	Un numérico especial para indicar sí o no. El sí es -1 y el no es 0. Por supuesto, Access mostrará un "Sí" o un "No", y es lo que tendrás que escribir en el campo.
Objeto OLE	Cualquier cosa que puedas "copiar y pegar": Una foto, un documento de Word, un pedazo de una hoja de Excel...
Hipervínculo	Direcciones Web.

Para algunos campos será necesario indicar el **tamaño**:

Texto	El número máximo de letras (de bytes) que ocuparán los datos. Recuerda, <u>todos</u> ocuparán ese número de bytes.
Numérico	<p>La idea es la misma: Cuántos bytes necesitas reservar. El problema es que lo pide de una manera un poco más extraña. En vez de decir 2 ó 4 bytes, debes escoger entre “entero” o “entero largo” (cosas de informáticos):</p> <p>Byte: ocupa un 1 byte. Valores entre 0 y 255. Sin decimales.</p> <p>Entero: 2 bytes. Valores entre –32.768 y 32.767. Sin decimales</p> <p>Entero largo: 4 bytes. Valores entre –2.147.483.648 y 2.147.483.647. Sin decimales.</p> <p>Simple: 4 bytes. Almacena números entre –3,402823E38 y –1,401298E–45 para valores negativos, y entre 1,401298E–45 y 3,402823E38 para valores positivos. Permite decimales, pero solo recuerda siete. Eso es mucho menos de lo que parece. Si te hacen falta cuatro o cinco decimales, y vas a operar con ellos, NO uses este tamaño, pues los errores por falta de decimales se propagan...</p> <p>Doble: 8 bytes. Almacena números entre –1,79769313486231E308 y –4,94065645841247E–324 para valores negativos, y entre 1,79769313486231E308 y 4,94065645841247E–324 para valores positivos. Hasta 15 decimales.</p> <p>Id. De réplica. 16 bytes. Sin decimales. Si tienes que numerar un montón de cosas (todas las direcciones IP de Internet) es útil. No vas a usarlo.</p>
Resto de tipos	Como son para usos concretos, Access ya les ha asignado un tamaño: Moneda ocupa ocho, bytes, Autonumérico cuatro bytes, etc.

2.1.1.2 Tipos de datos de MySQL

El formato y los tipos de datos de MySQL están basados en el estándar SQL. Como ya expliqué en la introducción, vamos a usarlo “a la antigua usanza”: Con la consola de comandos y escribiendo instrucciones de SQL. No hay pantallas y propiedades que se actualizan con el ratón. Cuando creemos los campos de una tabla, hay que indicar el tipo y el tamaño a la vez.

Hay decenas de tipos de datos, muchos de ellos equivalentes entre sí (cosas de la compatibilidad), así como abreviaturas para las expresiones más usadas. Sólo veremos los más usados.

En general, la sintaxis es “**TIPO(Tamaño)**”. Si el tipo es numérico el tamaño sólo indica cuántos dígitos se mostrarán en la consola cuando hagamos consultas (por lo general, nos da igual). Con ciertos tipos numéricos también se puede indicar el número de decimales que se verán.

Los tipos de datos **no numéricos** más usados:

CHAR (n)	Textos de longitud fija “n”. Máximo 255
VARCHAR(n)	Textos de longitud variable; “n” es el límite. Máximo 64K. Siempre ocupan la longitud del texto más dos bytes para el tamaño usado.
BINARY(n)	Como CHAR, pero almacena datos binarios (cualquier cosa) en vez de sólo caracteres ASCII.
VARBINARY(n)	Como VARCHAR, pero almacena datos binarios.

Hay muchos más, pero básicamente son alias o casos especiales de éstos tipos de datos. También admiten modificadores: Por ejemplo, BINARY es realmente un alias de “CHAR BINARY”. Mira el manual de referencia (está en castellano en la página de MySQL) para más información.

Los tipos **numéricos** más habituales:

BIT(M)	Un Bit. Con M se puede indicar cuántos bits van a usarse por cada valor.
TINYINT	Un byte. No admite decimales.
BOOL, BOOLEAN	Sinónimo de TINYINT(1). Se supone que "0" es falso y el resto "verdadero"
SMALLINT	Dos bytes. No admite decimales.
MEDIUMINT	Tres bytes. No admite decimales.
INT, INTEGER	Cuatro bytes. No admite decimales.
BIGINT	Ocho bytes sin decimales.
DOUBLE	Admite decimales. Equivalente al Double de Access.
DEC(M,D), DECIMAL(M,D), NUMERIC(M,D)	Número con decimales. "M" y "D" no son presentación, sino tamaño: Número total de dígitos (máximo 64) y número de decimales (máximo 30).

Un modificador muy habitual es UNSIGNED: Para el caso de TINYINT, por ejemplo, "TINYINT UNSIGNED" hace que el rango de valores posibles vaya de 0 a 255 en vez de -127 a 126. El modificador ZEROFILL hace que cuando se muestren los números en la consola se rellenen a ceros hasta el tamaño "M"...

Los tipos para **fecha y hora** son internamente numéricos, pero tienen unas cuantas características propias:

DATE	Una fecha. MySQL muestra los datos en formato "yyyy-mm-dd".
DATETIME	Fecha y hora. El formato en el que se muestran los datos es "yyyy-mm-dd hh:mm:ss".
TIME	Sólo la hora. El formato es "hh:mm:ss".
TIMESTAMP	El forma es el mismo que DATETIME, pero funciona de manera muy distinta. El sistema escribe automáticamente la fecha y la hora de la última operación realizada. Muy útil para saber cuándo se insertó la fila o se actualizó por última vez. Si se el asigna el valor NULL, se escribe la hora del sistema.

Las fechas suelen dar problemas cuando se programa con ellas. Asegúrate de que los tipos de datos empleados en tu programa y en la base de datos son compatibles o que los conviertes adecuadamente.

Hay muchas combinaciones posibles para escribir la fecha en MySQL. Por lo general, hay que respetar el formato de cada tipo de datos, aunque se admiten variaciones. Por ejemplo, para escribir en un campo DATE se admite "YYYY-MM-DD", "YYYY/MM/DD", "YYYYMMDD", "NOW()", "CURRENT_DATE", etc. Si un valor es incorrecto el campo se rellena con cero.

2.1.2 Cómo se hace en Access

Vamos a aplicar todos estos conceptos a Access. Sólo vamos a ver lo fundamental. En el capítulo 6, "Access" ampliaremos todo lo explicado.

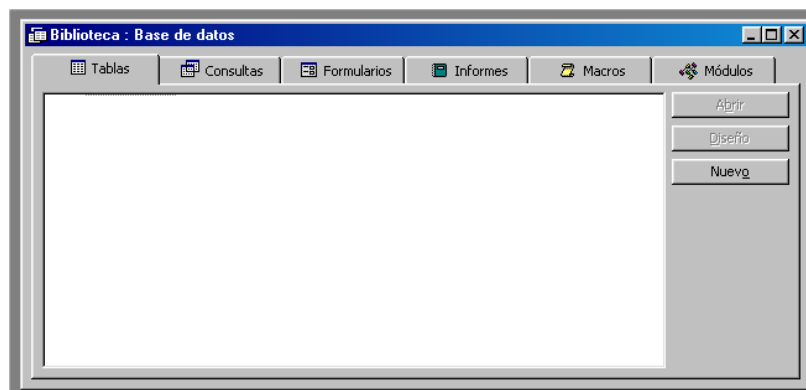
Crear una tabla en Access es sencillo: Como es la primera tabla que vamos a crear, la base de datos no existirá, por lo que lo primero que tenemos que hacer es **crear la base de datos**: En Word o Excel lo que se suele hacer es crear un documento en blanco y guardarlo al final. Con una base de datos no es

así. En cuantoelijamos “Base de datos en blanco” lo primero que preguntará Access es el nombre de la base de datos:



¿Por qué pregunta el nombre primero? Todas las tablas y diseños que hagamos se guardarán en ese fichero (por ejemplo “Biblioteca.mdb”). Y también los datos que escribamos dentro de las tablas. Pues bien, los datos se guardarán automáticamente. Nunca preguntará si queremos guardar el nuevo libro que hayamos introducido. Por eso necesita saber el fichero donde almacenará los registros.

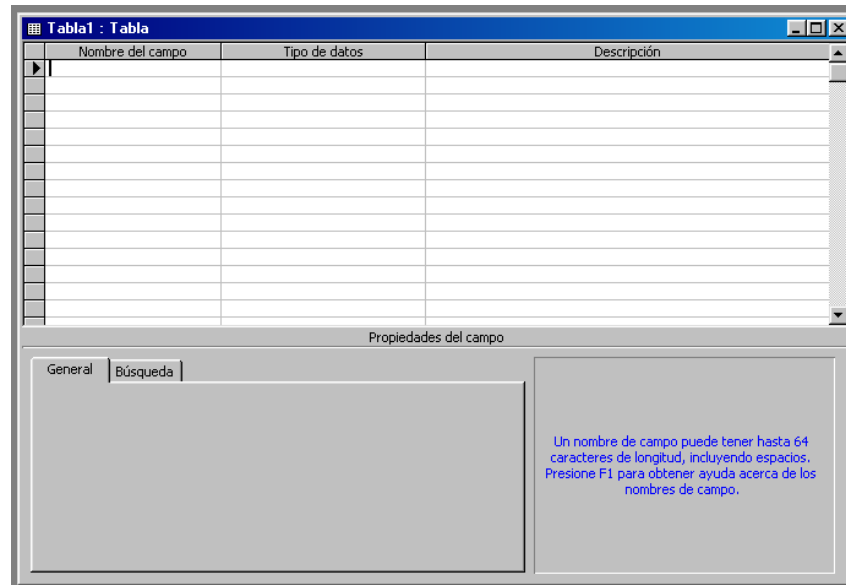
La pantalla que aparecerá será la siguiente:



Access tiene diferentes elementos: Tablas, consultas, etc. Más adelante veremos para qué vale cada uno. En Access 2003 las diferentes lengüetas aparecen a la derecha en vez de arriba.

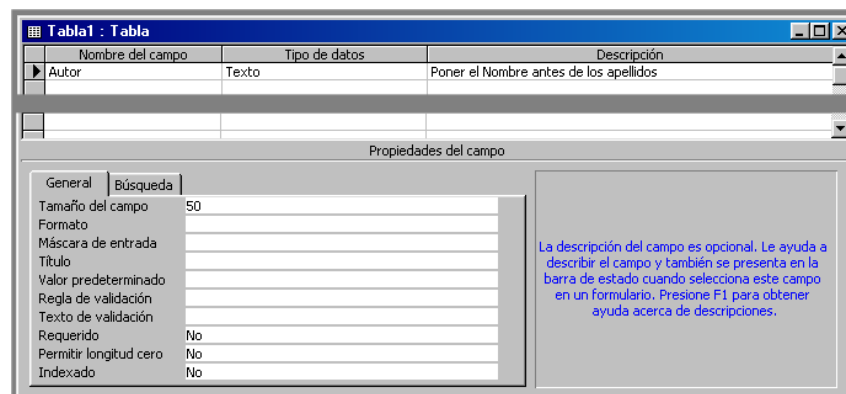
Para crear la tabla pulsaremos en el botón “Nuevo”, usaremos el botón derecho, elegiremos la opción del menú... Como en todo Windows, siempre hay diferentes maneras de hacer lo mismo. Access nos propondrá diferentes modos de crear la tabla. Para tablas y consultas **siempre** elegiremos “**Diseño**”, es decir, crear la tabla (o consulta) desde cero. Las herramientas para crear tablas y consultas sólo son útiles si se desconoce totalmente el funcionamiento de una base de datos.

Una vez que hemos escogido crear la tabla diseñándola desde cero, Aparecerá el siguiente cuadro de diálogo:



Nombre del campo	En nombre de la columna de la tabla, el nombre del campo. En nuestro caso los diferentes campos serán "Escritor", "Título", "Editorial", etc
Tipo de datos	Qué vamos a escribir (texto o números) y cuanto van a ocupar. El tipo de datos.
Descripción	Anotaciones (para nosotros, los diseñadores de la base de datos) que nos permitan recordar en un futuro detalles sobre ese campo.

Si nos ponemos a escribir los nombres y tipos de los campos⁵ la parte inferior de la pantalla cambia. Aparecen as **Propiedades** de ese campo:



Más adelante las veremos con detalle. De momento sólo usaremos "**Tamaño del campo**". Es ahí donde indicamos el tamaño máximo (y mínimo, todas las celdas ocupan lo mismo) que reservaremos para el Autor.

Siguiendo con el ejemplo, el aspecto final de la tabla sería:

⁵ Para escribir el tipo de datos, no lo escribas; escógelos de la lista. Y haz pruebas con el ratón: Pulsa al final de la celda, no en el medio, cliquee varias veces seguidas... recuerda que Windows tiene un montón de trucos para hacer más cómodas las tareas.

Nombre del campo	Tipo de datos	Descripción
Escritor	Texto	
Editorial	Texto	
Título	Texto	
Nº Pag	Numérico	
Tema	Texto	

Propiedades del campo

General | Búsqueda

Tamaño del campo: 50

Formato:

Máscara de entrada:

Título:

Valor predeterminado:

Regla de validación:

Texto de validación:

Requerido: No

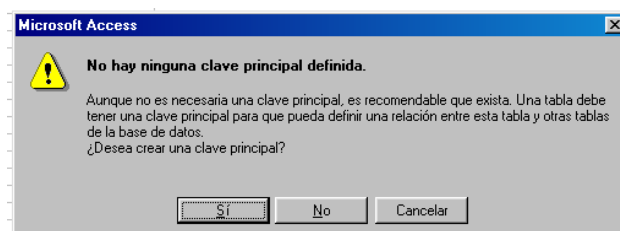
Permitir longitud cero: No

Indexado: No

El máximo número de caracteres que puede introducir en el campo. La longitud máxima que puede establecer es de 255. Presione F1 para obtener ayuda acerca de tamaño de campo.

Sólo queda guardar el diseño. Recuerda, Access guarda automáticamente los datos, no los diseños que hagas. Para guardarla puedes cerrarla (te preguntará el nombre) o guardar y cerrar. Yo la voy a llamar "Autores y Libros". Puedes usar espacios, guiones, acentos, etc. en los nombres. Los únicos caracteres prohibidos son el punto y el asterisco.

Cuando cierres la tabla aparecerá un cuadro de diálogo muy extraño:



De momento pulsa el "No". Las claves son muy importantes, pero las explicaré cuando profundicemos en el diseño de bases de datos.

2.1.2.1 Operaciones con los datos

La tabla ya está guardada, y aparecerá en la lista de tablas. Si queremos escribir datos en la tabla basta con hacer doble clic sobre ella, pulsando "Intro", el botón Abrir... como siempre, diferentes maneras de hacer lo mismo:

Escritor	Editorial	Título	Nº Pag	Tema
Cervantes	El Quijote	Alfaguara	1200	Literatura
Neuromante	William Gibson	Minotauro	316	Literatura - CF
Paul Auster	El palacio de la Luna	Anagrama	308	Literatura
Toti Martinez	La Calle de la Judería	Abra	490	Literatura - HIST
Paul Auster	Leviatán	Anagrama	266	Literatura
Günther Grass	El tambor de Hojalata	Alfaguara	655	Literatura
P. Auster	La Música del Azar	Anagrama	251	Literatura
Frederik Pohl	Pórtico	Ed. B	363	Literatura - CF
Agustín Froute	Java 2	RaMa	650	Informática
Petzold - Paul Yao	Programación en Windows 95	Mc Graw Hill	1100	Informática
Varios	Historia Universal	Anaya	900	Historia
Kim S. Robinson	Marte Rojo	Minotauro	650	Literatura -CF
Kim S. Robinson	Marte Verde	Minotauro	692	Literatura -CF
Kim S. Robinson	Marte Azul	Minotauro	724	Literatura -CF
*			0	

Registro: 1 de 14

El dato quedará automáticamente escrito al **cambiar de fila**. Una base de datos sólo **guarda filas completas**; nunca trozos de filas. Si sólo escribes un campo y pasas de fila escribirá todos los campos de dicha fila. Recuerda que un dato siempre ocupa lo mismo, aunque no tenga nada escrito. La última fila siempre parece vacía. Realmente esa fila no existe. Es el modo que tiene Access de permitirte añadir nuevas filas.

Del mismo modo sólo puedes **borrar filas completas**. Se hace pulsando en el cuadrado que hay a la izquierda de la misma (la fila queda seleccionada) y pulsado la tecla suprimir.

No puedes borrar sólo un campo. Puedes **modificar** su valor, añadiéndole más letras o quitándoselas todas; Vuelvo a recordar que el campo siempre ocupa lo mismo, independientemente de su contenido.

Por último, si un campo ha quedado vacío (has borrado todo su contenido) ¿Qué valor contiene? Evidentemente contiene el valor **ninguno en absoluto**. Como eso es muy largo de decir, los informáticos dicen **Nulo** (los informáticos ingleses dicen Null). Es un valor muy usado; Por ejemplo, si una persona no tiene número de teléfono, ¿Qué número escribes en el campo "Num Tel"? ¿Un "0" para indicar que no tiene teléfono? Lo más adecuado es dejarlo en blanco, es decir, asignarle un valor Nulo. Las bases de datos son capaces de trabajar con ese valor especial. No confundir "cero" (el saldo del campo "cuenta corriente" puede ser 1000 €, 720 €, 0€ ó -200 €) con nulo (No tiene cuenta corriente)

Para acabar, un consejo. Todos los elementos que creamos en Access tendrán esa doble vertiente de "Diseño" y "Datos". La forma más cómoda de pasar de uno a otro es el botón de la esquina superior izquierda de la pantalla. Cambiará de dibujo en función de dónde nos encontremos, y nos permitirá pasar:

de diseño a datos 

de datos a diseño 

2.1.3 Cómo se hace en MySQL

El proceso es más complicado que con Access. No vamos a usar herramientas gráficas, por lo que habrá que escribir todo el código. Además tenemos obligatoriamente que emplear claves y usuarios. Y la primera vez que accedamos a MySQL hay que realizar una serie de tareas.

Por ese motivo vamos a ver el proceso por encima. En el capítulo 7, "Introducción a MySQL" se explican todos los pasos y conceptos con más detalle. Léete los primeros puntos de ese capítulo para aprender cómo se hace con MySQL. El objetivo de esta sección es que te hagas una idea de cómo funciona una base de datos pensada para informáticos. Con lo explicado en este apartado no podrás implementar la base de datos.

Supongamos que ya hemos configurado MySQL. Al contrario que en Access, MySQL guarda todos los datos en un único almacén que gestiona ella misma. Obliga a entrar con el consabido usuario y clave para comprobar si tienes o no derecho a acceder a los datos. Vamos a entrar como el administrador para crear la base de datos "pruebas" y un usuario llamado "maria" para que la gestione:

```
C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 31
Server version: 5.1.31-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database pruebas;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on pruebas.* to maria identified by 'clavemaria';
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> exit
Bye

C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql -u maria -p pruebas
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 32
Server version: 5.1.31-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> _
```

Bien, todo hecho. Hemos salido de MySQL y hemos vuelto a entrar como maria, directamente a la base de datos “pruebas”. Ahora creamos una tabla parecida a la que hicimos en Access. La forma de hacerlo es usar la sentencia de SQL CREATE:

```
mysql> CREATE TABLE AUTORES_Y_LIBROS (
-> ESCRITOR VARCHAR(40),
-> EDITORIAL VARCHAR(40),
-> TITULO VARCHAR(40),
-> PAGINAS SMALLINT,
-> TEMA VARCHAR(100));
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+-----+
| Tables_in_pruebas |
+-----+
| autores_y_libros   |
+-----+
1 row in set (0.00 sec)
```

Como lo estamos escribiendo en la línea de comandos⁶, trabajar con los datos no es tan sencillo. Todo se hace con sentencias de SQL. En el siguiente apartado veremos ejemplos de las más habituales.

2.1.3.1 Operaciones con los datos

Vamos a escribir un par de líneas en la tabla:

```
mysql> INSERT INTO AUTORES_Y_LIBROS (ESCRITOR,EDITORIAL,TITULO,PAGINAS,TEMA)
-> VALUES("William Gibson","Minotauro","Neuromante",316,"Literatura - CF");
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> INSERT INTO AUTORES_Y_LIBROS (ESCRITOR,EDITORIAL,TITULO,PAGINAS,TEMA)
-> VALUES("Paul Auster","Anagrama","El Palacio de la Luna",308,"Literatura");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM AUTORES_Y_LIBROS;
```

ESCRITOR	EDITORIAL	TITULO	PAGINAS	TEMA
William Gibson	Minotauro	Neuromante	316	Literatura - CF
Paul Auster	Anagrama	El Palacio de la Luna	308	Literatura

```
2 rows in set (0.00 sec)
```

INSERT añade nuevas filas a la tabla. SELECT sirve para ver el contenido de la misma. Léete el capítulo dedicado a MySQL para ver con más detalle éstas y las demás instrucciones de SQL.

⁶ Existen herramientas gráficas, pero no voy a explicarlas: Ya has visto cómo se hace en Access.

Para modificar y borrar datos:

```
mysql> UPDATE AUTORES_Y_LIBROS SET PAGINAS=400 WHERE TITULO="El Palacio de la Luna";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM AUTORES_Y_LIBROS;
+-----+-----+-----+-----+-----+
| ESCRITOR | EDITORIAL | TITULO | PAGINAS | TEMA |
+-----+-----+-----+-----+-----+
| William Gibson | Minotauro | Neuromante | 316 | Literatura - CF |
| Paul Auster | Anagrama | El Palacio de la Luna | 400 | Literatura |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> DELETE FROM AUTORES_Y_LIBROS WHERE EDITORIAL="Minotauro";
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM AUTORES_Y_LIBROS;
+-----+-----+-----+-----+-----+
| ESCRITOR | EDITORIAL | TITULO | PAGINAS | TEMA |
+-----+-----+-----+-----+-----+
| Paul Auster | Anagrama | El Palacio de la Luna | 400 | Literatura |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Como se verá más adelante, casi todas las instrucciones de SQL de manipulación de datos admiten la cláusula WHERE.

2.2 Códigos y Relaciones. Consultas

La tabla que hemos creado está mal diseñada. En el capítulo sobre diseño de bases de datos aprenderemos a hacerlo bien, pero a simple vista se observa claramente un fallo. Estamos repitiendo demasiada información. El nombre del autor, por ejemplo, lo repetimos constantemente. Y menos mal que es sólo el nombre. Si añadiéramos más campos, como fecha de nacimiento, nacionalidad, etc. el problema sería aun mayor. La solución es sencilla; crear más de una tabla. Voy a rehacer la base de datos y crear las tabla de “Autores” y la tabla de “Libros”⁷.

Título	Editorial	Nº Pag	Tema
El Quijotes	Alfaguara	1200	Literatura
Neuromante	Minotauro	316	Literatura - CF
El palacio de la Luna	Anagrama	308	Literatura
La Calle de la Judería	Abra	490	Literatura - HIST
Leviatán	Anagrama	266	Literatura
El tambor de Hojalata	Alfaguara	655	Literatura
La Música del Azar	Anagrama	251	Literatura
Pórtico	Ed. B	363	Literatura - CF
Java 2	RaMa	650	Informática
Programación en Windows 95	Mc Graw Hill	1100	Informática
Historia Universal	Anaya	900	Historia
Marte Rojo	Minotauro	650	Literatura -CF
Marte Verde	Minotauro	692	Literatura -CF
Marte Azul	Minotauro	724	Literatura -CF
*		0	

Escritor
Cervantes
William Gibson
Paul Auster
Toti Martinez
Günter Grass
Frederik Pohl
Agustín Froute
Petzold - Paul Yao
Varios
Kim S. Robinson
*

De paso, al borrar los escritores repetidos, me he asegurado que el escritor siempre se llame igual (fíjate en Paul Auster en los ejemplos anteriores).

⁷ Si está siguiendo el ejemplo, sé un vago. Copia y pega la tabla de autores y libros, las renombras y en diseño borras los campos que sobren. Y en autores, las filas repetidas.

2.2.1 Códigos

El problema ahora es saber qué autor ha escrito qué libro. Las bases de datos relacionales lo resuelven **añadiendo un campo en las dos tablas**. Tanto a la tabla de libros como a la tabla de Escritores les añadiré el campo “Código de Escritor”. Lo que voy a hacer, sencillamente, es inventarme un número diferente para cada escritor. ¿Por qué un número? Por que las bases de datos trabajan más rápidamente con números que con letras. ¿Y por que un número diferente para cada uno? Porque ese número servirá para identificarle. Será el “DNI” del escritor en nuestra base de datos. Los informáticos denominamos a este tipo de valores **códigos**:

Código de Escritor	Escritor
1	Cervantes
2	William Gibson
3	Paul Auster
4	Toti Martinez
5	Günter Grass
6	Frederik Pohl
7	Agustín Froute
8	Petzold - Paul Yao
9	Varios
10	Kim S. Robinson

Si el valor de ese DNI nos da igual siempre que sea único, el tipo de datos Autonumérico es ideal para ello.

El paso siguiente es hacer lo mismo con la tabla de Libros. Por supuesto el contenido del campo de código no podrá ser cualquiera: Escribiremos el código correspondiente al autor de ese libro:

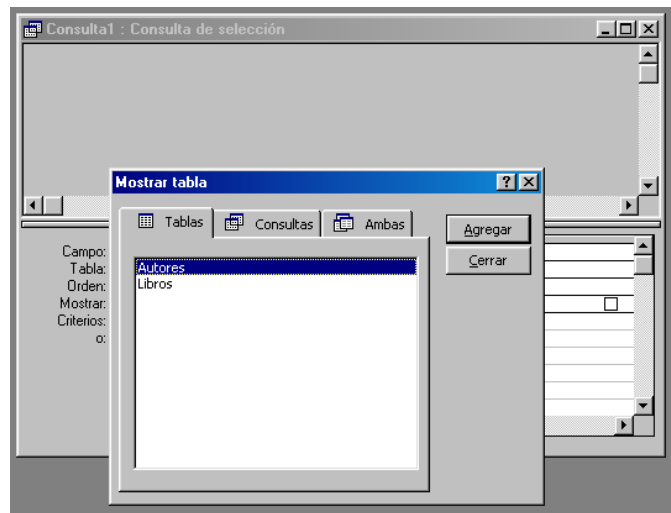
Título	Editorial	Nº Pag	Tema	Código de Escritor
El Quijotes	Alfaguara	1200	Literatura	1
Neuromante	Minotauro	316	Literatura - CF	2
El palacio de la Luna	Anagrama	308	Literatura	3
La Calle de la Judería	Abra	490	Literatura - HIST	4
Leviatán	Anagrama	266	Literatura	3
El tambor de Hojalata	Alfaguara	655	Literatura	5
La Música del Azar	Anagrama	251	Literatura	3
Pórtico	Ed. B	363	Literatura - CF	6
Java 2	RaMa	650	Informática	7
Programación en Windows 95	Mc Graw Hill	1100	Informática	8
Historia Universal	Anaya	900	Historia	9
Marte Rojo	Minotauro	650	Literatura -CF	10
Marte Verde	Minotauro	692	Literatura -CF	10
Marte Azul	Minotauro	724	Literatura -CF	10
		0		0


Ya hay una especie de **relación** entre las dos tablas que hemos creado. Tienen una información en común, “repetida”, el código de escritor. Si añadimos nuevos campos a la tabla de Escritores, no influirá en la tabla Libros. “Cervantes” seguirá siendo el escritor de Código “1” independientemente de que añadamos los campos Nacionalidad o Fecha de Nacimiento a la tabla de Escritores. Del mismo modo, si modificamos el nombre y escribimos “Miguel de Cervantes” o “P. Auster” la tabla libros se queda como estaba.

2.2.2 Cómo se hace en Access. Consultas

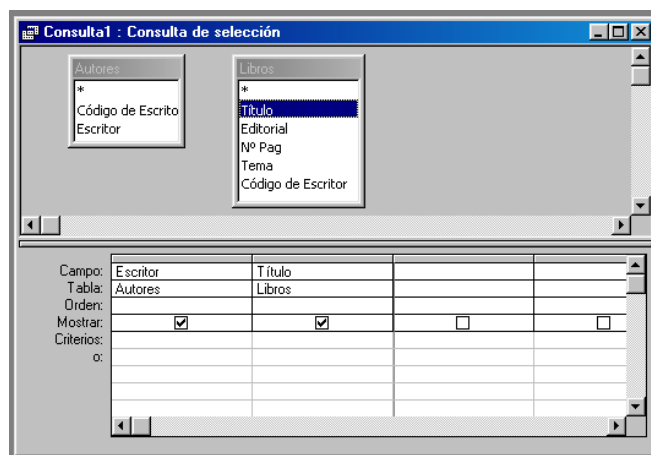
Si le decimos a alguien que “Marte Rojo” fue escrito por el escritor número 10, como mínimo se extrañará. Le deberemos decir que fue escrito por “Kim S. Robinson”. Necesitamos **información** de **varias tablas** a la vez. Para eso están las **consultas**.

Para crear una consulta tenemos que pinchar en la lengüeta consultas, y de modo similar a como diseñamos la tabla, pulsa en “Nueva”. En este caso también seleccionaremos la opción de “Diseño”, es decir, crearla desde cero. Se nos mostrará una pantalla con el siguiente aspecto:

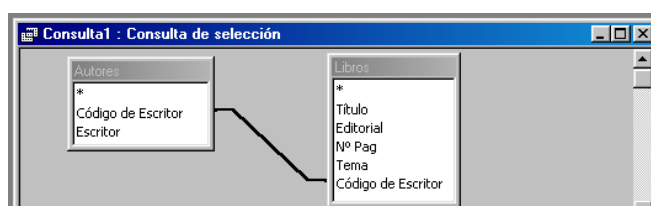


El cuadro de diálogo donde aparecen todas las tablas (y consultas) disponibles aparece por defecto al iniciar una consulta nueva. Para que vuelva a aparecer si queremos elegir más, se usa el botón .

Queremos ver el título del libro (tabla Libros) y el nombre del escritor (tabla Autores). Agregamos las dos. A continuación, hacemos doble clic sobre los campos que nos interesen (se irán añadiendo debajo de manera automática).



Ya casi está. Si pasamos a ver los datos... ¡Salen cosas absurdas! Está sacando todas las combinaciones posibles entre Autores y Libros. 10 autores por 14 libros, en la consulta salen 140 filas. NO le hemos indicado la relación entre las tablas. Para decirle que el campo “Código de Escritor” de la tabla Autores está relacionado con el campo “Código de Escritor” de la tabla Libros simplemente ponemos el ratón encima de uno de ellos y arrastramos hasta colocarnos encima del otro. Quedará:



Ahora el resultado sí que tendrá lógica. No lista todas las posibles combinaciones, sino aquellas en las cuales los códigos de ambas tablas coinciden. Por supuesto, una consulta se puede hacer con el número de tablas que se desee. Si este ejemplo no te ha quedado muy claro, repítelo pero añadiendo a la consulta (haciendo doble clic sobre ellos) los campos Código de Escritor de la tabla Autores y Código de Escritor de la tabla Libros. Verás que sin la relación salen las 140 filas, todas las posibles combinaciones, y con la relación sólo aquellas filas cuyos campos de código coinciden.

Debe quedar muy claro que las consultas prácticamente **no ocupan espacio**. Los datos están en las tablas. Las consultas son operaciones realizadas sobre las tablas, que se ejecutan cuando pasas a la pantalla de ver resultados. Internamente lo que has hecho realmente es escribir código SQL⁸, sólo que lo has “escrito” con el ratón. El código SQL correspondiente a la consulta anterior sería:

```
SELECT Autores.Escritor, Libros.Título FROM Autores INNER JOIN Libros ON Autores.[Código de Escritor] = Libros.[Código de Escritor];
```

Si quieres verlo, Usa el menú “Ver” o la flecha que hay junto al botón para pasar de datos a diseño o viceversa.

Cuando aprendamos más sobre bases de datos, las relaciones **nunca** las haremos dentro de las consultas. Nos encargaremos de que la base de datos las “aprenda” y las coloque automáticamente cuando creemos una consulta.

2.2.3 Cómo se hace en MySQL

Como ya habrás imaginado, Las consultas en MySQL son sencillamente sentencias SELECT. En las bases de datos profesionales también pueden darse nombre a las consultas. Consulta los apartados 7.9, “Consultas. Select y 7.10, “Vistas” para más información.

Suponiendo que hayamos creado unas tablas similares a las del ejemplo de Access, La consulta anterior quedaría:

```
SELECT AUTORES.ESCRITOR, LIBROS.TITULO FROM AUTORES JOIN LIBROS ON  
AUTORES.CODLIBRO=ESCRITORES.CODLIBRO;
```

⁸ El lenguaje SQL es un lenguaje estándar de base de datos diseñado para realizar las operaciones sobre las tablas. Una de las ventajas de Access es que el usuario no necesita aprender dicho lenguaje. El SQL usado por Microsoft no se ajusta del todo al estándar (para variar).

3

Diseño de Bases de Datos

En este capítulo aprenderemos a diseñar una base de datos, es decir, aprenderemos a decidir qué tablas necesitamos, qué campos deben tener esas tablas y cómo deberían estar relacionadas. También aprenderemos un concepto nuevo: Las claves.

3.1 Un método de diseño

No hay (de momento) un programa que diseñe bases de datos. No es un proceso mecánico que se pueda programar fácilmente. Sin embargo hay ciertos pasos que nos pueden ayudar mucho a la hora de decidir lo que necesitamos. El método “teórico” de diseño de bases de datos parte de los **campos**. El diseñador piensa en todos los campos que necesita, y después los agrupa en tablas. Ese método teórico esta bien, pero no es práctico: una base de datos real tiene demasiados campos. Nadie lo usa en la práctica. Lo que vamos a hacer nosotros es basarnos en las **tablas**. Nuestro método de diseño va a tener cuatro pasos:

- ¿Qué tablas necesito?
- ¿Qué campos tiene cada tabla?
- Relaciones.
- Claves.

No se puede explicar diseño de bases de datos en abstracto. Vamos a aplicarlo a un ejemplo: Una biblioteca. Queremos saber los libros que tenemos, los socios, libros prestados, autores de los libros, multas, libros más leídos...

Algo debe quedar claro. Antes de hacer una base de datos debes conocer **perfectamente** lo que quieres hacer. Si la haces para otra persona, interroga a esa persona todo lo que te haga falta. Si vas a informatizar un proceso que se hace con papeles, quédate con todos los papeles y estúdialos: De ahí sacaras las tablas directamente. **Depende** de lo que te pidan o necesiten, saldrán unas tablas u otras.

Un buen consejo. Ni se te ocurra acercarte al ordenador hasta que estés seguro de que el diseño está perfecto. Si empiezas a escribir la base de datos y el diseño es incorrecto, tendrás que borrar lo que has hecho y empezar de nuevo. Eso te pasará hasta con tu diseño perfecto. Cuantas menos veces ocurra, mejor.

¡Ah! Tus primeras cinco o seis bases de datos serán un churro. Cometerás errores que no cometerás en la séptima. Es lógico, nadie nace enseñado. Lo bueno que tienen las bases de datos es que te darás cuenta de que son malas, porque los datos no “encajarán”: No podrás saber cierta cosa, las consultas saldrán con datos de más o de menos, no podrás escribir cierta información... En ese caso, ya sabes: A borrar y empezar de nuevo.

3.2 Tablas

La pregunta es ¿Qué cosas hay? ¿Qué **conjuntos** de datos hay? Eso, a simple vista, serán las tablas. Seguramente no saldrán todas, o incluso pondremos alguna que no debería existir. No tiene demasiada importancia. En Relaciones es cuando decidiremos cuáles son las correctas y cuáles faltan.

Estamos hablando de una biblioteca y los préstamos de sus libros. Las “conjuntos de datos” que en este momento se me ocurren son⁹:



Libros se parece a la tabla libros del primer ejemplo, Autores Idem, los Socios serán los socios de la biblioteca que pueden sacar libros y Temas será una tabla con los posible temas de los cuales traten los libros. ¿Por qué he hecho una tabla de temas? Por el mismo motivo que en el ejemplo anterior hice la de Autores: **Nunca repitas información**. Mejor dicho, **NUNCA REPITAS INFORMACIÓN**. Si estás escribiendo “Literatura” en la mitad de las filas de la tabla de libros, lo estás haciendo mal. El tema no es un campo de la tabla libros, sino una tabla aparte.

En este paso no estoy hablado de “Nombre de Socio”, ni “Título del libro”. Eso son campos de Socios y de Libros. Lo que tratamos de encontrar (por encima) son las tablas.

3.3 Campos

¿Qué quiero saber de cada tabla?. Eso serán los diferentes campos que la forman. En nuestro caso:

- *Autores: Nombre y apellidos del autor, fecha de nacimiento*
- *Libros: Título, nº de páginas, editorial, tema, autor*
- *Temas: Nombre del tema*
- *Socios: Nombre y apellidos, teléfono, dirección*

Fíjate en los campos de la futura tabla libros. Ahí he dicho (como es lógico) que del libro quiero saber el tema y el autor. Pero ¡Si eso son otras tablas! Realmente, tema y autor NO son campos de la tabla libros. Sencillamente estás expresando que la tabla libros está Relacionada con la tabla Temas y la tabla Autores. Eso se hará en el paso siguiente del diseño.

Con los nombres y apellidos. ¿Son un único campo o hago tres distintos? La respuesta es sencilla: **EN UN CAMPO SÓLO CABE UN ELEMENTO**. Si NUNCA vas a usar por separado nombre y apellidos, se trata de un único campo. Pero si a veces vas a imprimir sólo nombre y apellido primero, o vas a ordenar por apellido segundo, apellido primer o y nombre, se tratan de tres campos. Con dirección del socio ocurre lo mismo: Es muy probable que necesites calle, número, piso, letra, portal, mano, municipio y provincia. En nuestro caso voy a suponer que sí que voy a usar por separado nombres y apellidos (tres campos), pero no la dirección. La uso simplemente para que el socio sepa que sabemos dónde vive y que no se puede quedar con los libros.

Vaya, quiero saber la editorial del libro: Nos va a pasar lo de siempre; No voy a escribir “Anaya” o “Edelvies”. NO voy a repetir información. Crearemos la tabla Editoriales. La base de datos nos queda por tanto:

- *Autores: Nombre autor, apellido primero, apellido segundo, fecha de nacimiento*

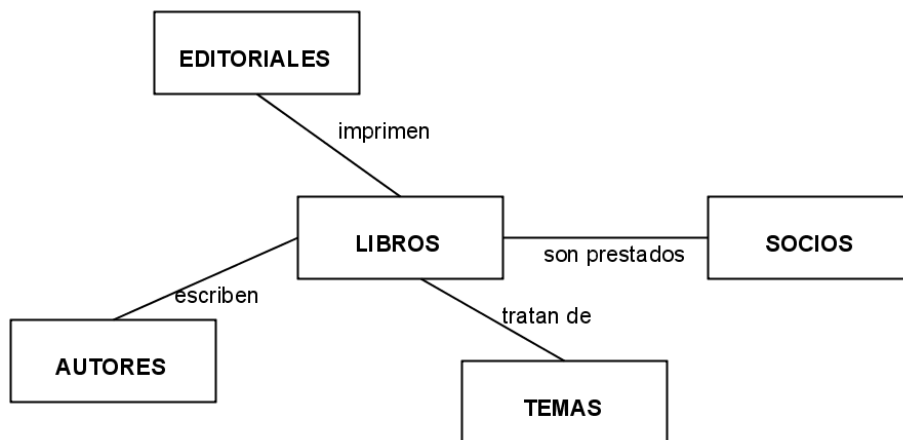
⁹ Es mentira. Esta base de datos la he hecho veinte veces. Me sé de memoria las tablas que tiene. Pero disimulemos...

- *Libros: Título, nº de páginas*
- *Temas: Nombre del tema*
- *Socios: Nombre y apellidos, teléfono, dirección*
- *Editoriales: Nombre de la editorial*

Si la base de datos no es para ti, ¿Cómo puedes saber la información que necesita tu cliente? Pregúntale. Pregúntale todas las dudas que tengas. Cuantos menos cosas te olvides en el diseño, menos correcciones harás en la base de datos.

3.4 Relaciones

Ahora nos queda relacionar las diferentes tablas, de tal modo que dibujemos un camino que una a todas ellas:



Estamos hablando de relaciones “lógicas”, “normales”. No tiene nada que ver con la informática (de momento). Los autores escriben libros, las editoriales los imprimen, los libros son prestados a los socios... He enganchado de alguna manera unas tablas con otras. Por supuesto, podría enganchar socios con autores, diciendo: “Los socios leen libros de ciertos autores”. Pero queda... rebuscado. En la propia forma de decirlo ya he incluido que los libros están por medio. Éste es el paso que da más problemas, pues depende totalmente de lo que entendamos que es cada tabla.

Entonces, ¿Cómo sabemos si está bien engarzado? Para eso está la teoría de bases de datos.

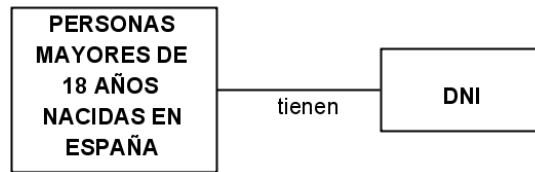
Por cierto, fíjate en que para describir una tabla siempre utilizas nombres, mientras que para indicar las relaciones casi siempre usarás verbos...

3.4.1 Tipos de relaciones

Entre dos tablas, entre dos conjuntos de datos, y atendiendo al número de elementos relacionados, sólo pueden existir tres tipos de relaciones:

- Relaciones de 1 a 1
- Relaciones de 1 a varios (se dice de 1 a N)
- Relaciones de varios a varios (se dice de N a M)

Es sencillo. Por ejemplo:

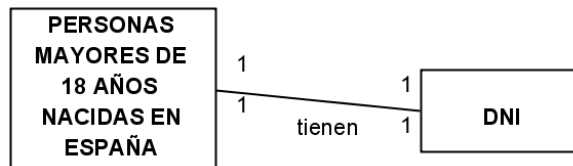


La relacion “con palabras” está clara. Los mayores de 18 tienen DNI'S. Para saber el tipo de relación hay que hacera siempre dos preguntas:

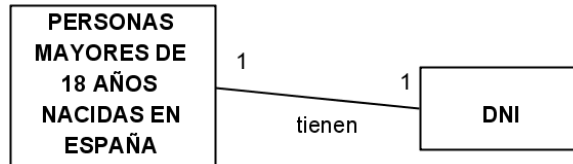
Una persona mayor de 18 años nacida en España, ¿Cuántos DNI'S tiene? Pues **uno**, evidentemente.

Un DNI, ¿A cuantas personas mayores de 18 años nacidas en España pertenece? Pues a **una**, o alguien tendrá serios problemas cuando haga la declaración de la renta.

He hecho la pregunta en **ambos sentidos**, y me ha salido algo como esto:

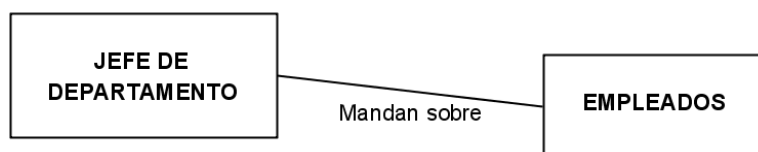


Simplificando:

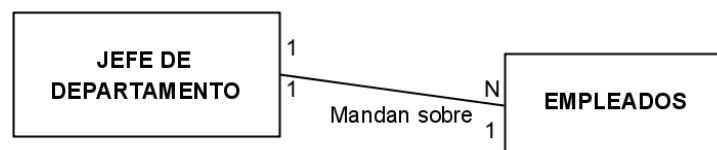


Es decir, hay una **relación de 1 a 1**

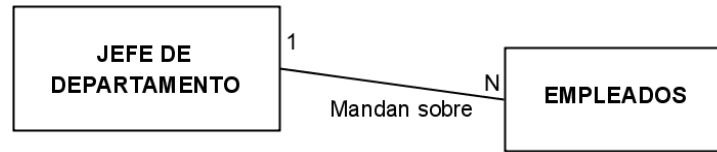
Otro ejemplo:



Hacemos la pregunta en ambos sentidos. **Un** jefe de departamento tiene a su cargo a **varios** empleados. Y **un** empleado obedece sólo a **un** jefe de departamento.

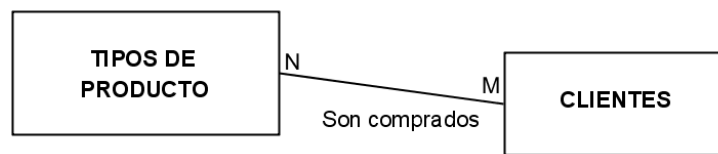


Simplificando:



Hay una relación de **uno a varios** entre jefes y empleados.

Por último nos queda un ejemplo de relación de **varios a varios**. Supongamos Clientes y Tipos de Producto. La relación, con palabras, sería que "los clientes compran productos". **Un** cliente puede comprar **varios** tipos de producto (eso espero) y **un** tipo de producto puede ser comprado por **varios** clientes. Simplificando, la relación quedaría:

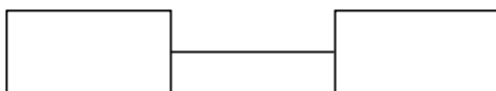


Cuidado. La relación entre "Coches" y "Plazas de Garaje" puede ser de N a M. En una plaza de garaje pueden aparcar varios coches... a lo largo del tiempo. La relación sería de uno a varios si lo que quieres es una "foto" de la situación actual. Pero si quieres el "histórico", saber qué coches aparcaron en qué plazas la relación será de varios a varios.

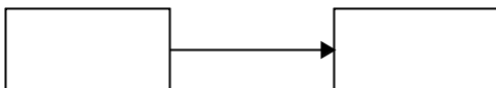
Ya que hablamos de coches y plazas de garaje, supongamos que lo único que quiero es la situación actual. No me interesan los coches que aparcaron hace media hora, sólo los que hay ahora. Una plaza de garaje ¿Cuántos coches puede tener? Uno... o ninguno. La teoría dice que eso es una relación de uno a uno. Yo lo trataré como una relación de uno a varios. Hay varios casos posibles (bueno son sólo dos, pero son más de uno), por lo tanto la relación es de 1 a N.

A partir de ahora las relaciones las representaré así:

- Relación de uno a uno.



- Relación de uno a varios.



- Relación de varios a varios.



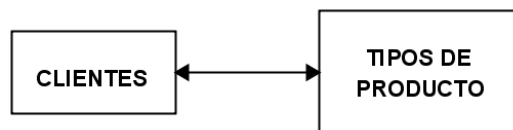
Muy bien ¿Y para que es necesario todo esto? Pues resulta que **Una base de datos relacional sólo puede representar las relaciones de uno a varios**. El resto de relaciones, generalmente, están mal. Afortunadamente el repararlo es una tarea casi mecánica.

3.4.2 Reparando relaciones

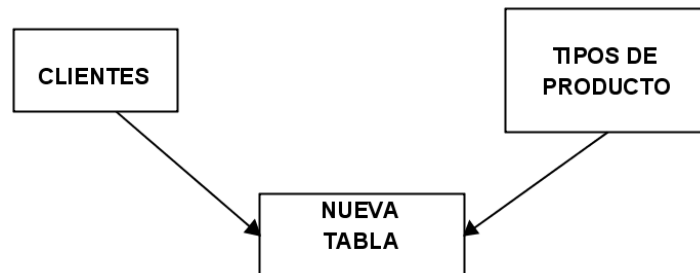
Casi siempre que haya una relación de **uno a uno**, esa relación está mal. No son dos tablas, sino una única tabla que engloba los campos de las dos primeras. En el caso del ejemplo anterior, No existe la tabla "Personas mayores de 18" y la tabla "DNI'S". Se convertirán en una única tabla con los campos (por ejemplo) Nombre de persona, apellido primero, DNI, etc.

En ciertos casos SI que se puede mantener una relación de uno a uno, pero son tan raros que cuando haga falta te darás cuenta; Lo harás a propósito por motivos de seguridad o rendimiento.

Siempre que aparezca una relación de **varios a varios**, significa que **falta** una tabla, y esa tabla que falta siempre tendrá el siguiente aspecto:



Se convierte en:



Fíjate en las flechas. Siempre la nueva tabla es la parte N de las dos nuevas relaciones. En este caso ¿Cuál puede ser esa nueva tabla? Es una tabla que habla de los clientes y los productos que han comprado... Las Facturas. En nuestro diseño sobre clientes y productos no habíamos incluido nada para almacenar las facturas: Cliente, Producto, Fecha de compra, cantidad...

Por tanto, no importa cómo nos salga el diseño inicial de nuestra base de datos. Siempre lo podremos traducir a relaciones de uno a varios. Siempre sabremos si nos sobran o faltan tabla. Suponiendo que las relaciones entre ellas estén bien, pero eso es cuestión de práctica.

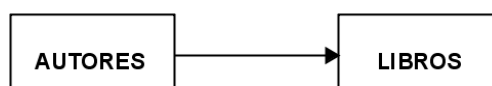
3.4.3 Qué significa una relación

Lo vimos por encima en el capítulo anterior. ¿Qué significa que dos tablas estén relacionadas? Significa que tienen un campo en común. A una de ellas le añades un campo (se le suele llamar código), numeras los registros y en la otra repites la operación, sólo que en vez de numerar lo que haces es copiar los códigos que te interesen de la primera tabla (recuerda Autores y Libros).

Ahora bien, si quiero relacionar libros y autores, ¿Qué campo me invento? Tengo dos opciones:

- Crear "Código de Libro" en las dos tablas. Numero los libros de 1 a los que sean y en el código de libro de la tabla Autores indico que han escrito
- Creo "Código de Autor" en las dos tablas, etc.

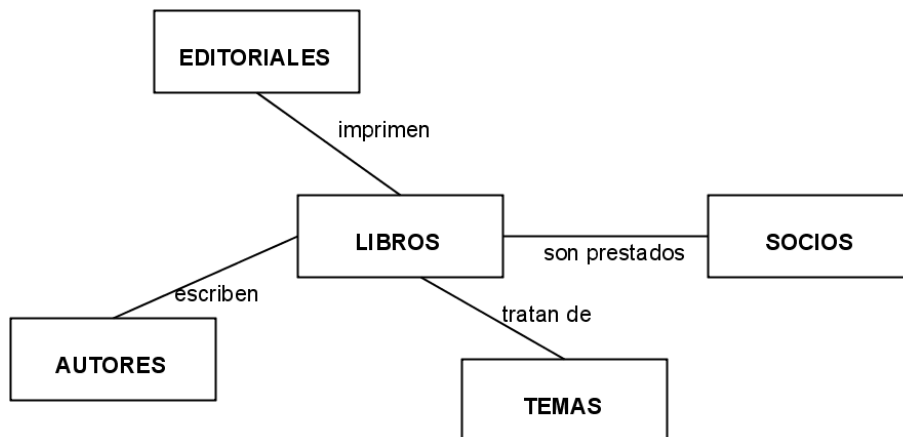
La respuesta es sencilla. **LA PARTE UNO DE LA RELACIÓN SIEMPRE MANDA.** En el ejemplo del capítulo anterior suponíamos (está mal, pero en fin) que la relación entre autores y libros era:



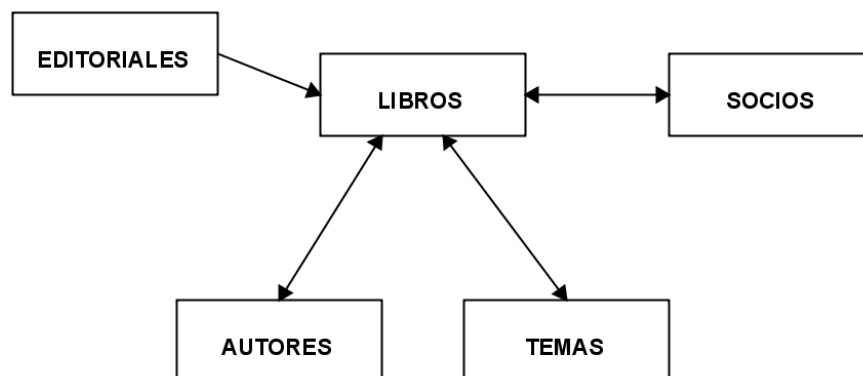
Es decir, un autor puede escribir varios libros, un libro pertenece a un autor. Relación de uno (Autores) a varios (Libros). Por tanto, la única posibilidad es crear “código de autor” en Autores y Libros. Prueba a hacerlo al revés y verás que no es posible: Te obligaría a meter varias cosas en un solo campo, y eso no puede suceder nunca (recuerda lo de apellidos y nombre)¹⁰.

3.4.4 Creando la biblioteca

Bien, ahora sí que podemos hacer algo con nuestro diseño inicial de la biblioteca:



Aplicando la teoría nos queda (He borrado los nombres de las relaciones para clarificar el dibujo, pero no las olvides, pueden ser de utilidad):



No, no me he equivocado: Un autor puede escribir varios libros, y un libro puede estar escrito por varios autores. Un Tema puede aparecer en varios libros y un libro puede tratar de varios temas. Un socio puede leer varios libros y un libro puede ser leído por varios socios (a lo largo del tiempo, claro).

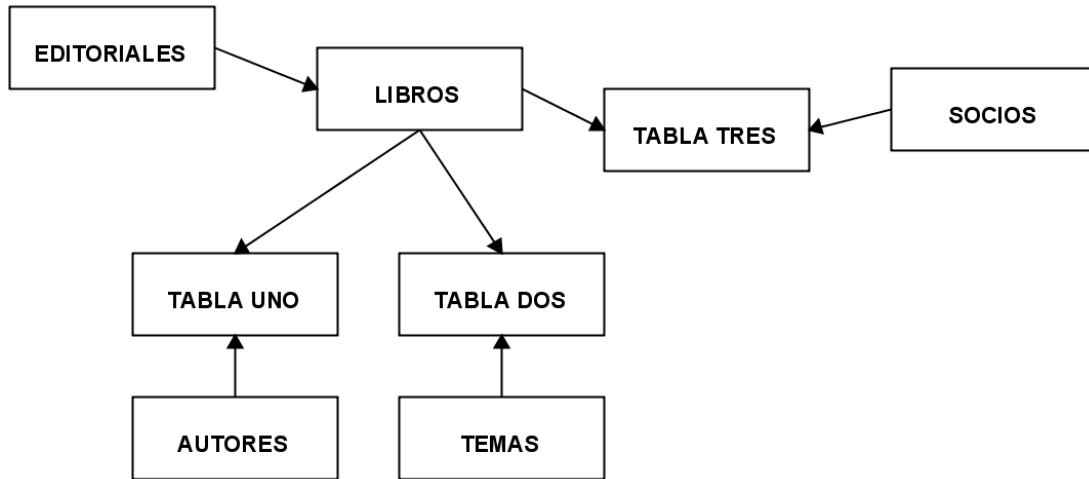
¿Y con las editoriales por qué no? Es posible que dos editoriales se unan para imprimir un libro. Pues no es una relación de varios a varios porque no he querido. O mejor dicho, porque le pregunté al director de la biblioteca y me dijo que eso nunca sucedía con sus libros (el sabrá)

Le pregunté si quería saber los préstamos actuales (relación de uno a varios entre socios y libros) o todos los que se han producido, y escogió esto último. Le pregunté si por “libro” entendía “tipos de libro” (tengo tres ejemplares del Quijote) o volúmenes (tengo un Quijote de tapas duras, otro encuadernado en rojo y otro en edición de bolsillo). Como dijo “volúmenes”, no añadí en campo cantidad a la tabla de

¹⁰ ¡Trampa! ¡Está mal hecho! El autor del libro “Programación en Windows 95” son dos: Petzold y Paul Yao, y están en el campo Nombre de Autor. ¡Has metido DOS cosas en un único campo! Pues sí, tienes razón. Está mal... Ya te dije que la primera tabla que hicimos era un desastre. La relación entre Libros y Autores no es de 1 a N.

libros, aunque quizás no estaría de más añadir “tipo de encuadernación”... Preguntad todo lo que se os ocurra.

Siguiendo la teoría, las relaciones de N a M se descomponen en:



Todavía no sé que pueden ser esas tres nuevas tablas, pero sí sé que tienen que ser como las he dibujado. Nunca hay excepciones.

El problema ahora es saber qué pueden ser. Hay veces que serán extrañas tablas que necesita la base de datos para relacionar registros y otras serán tablas fundamentales que me olvide en el primer paso del diseño. Se algo de esas tablas: El nombre de la relación que las ha creado y un par de campos:

La Tabla Uno sé que tiene “Código de Autor”, ya que es “la parte N” de la relación con Autores. Y Tiene “Código de Libro” por el mismo motivo. La relación era “escriben”. Es la tabla que relaciona libros con autores. Es ahí donde voy a indicar qué libros ha escrito qué autor. Su aspecto será extraño:

Tabla Uno	
Código de Libro	Código de Autor
1	1
1	2
2	23
3	12
4	1

Es ahí donde indicaré que el libro de código “1” fue escrito por el señor de código “1” y el de código “2”. También veo que el libro “4” fue escrito (al parecer en solitario) por el autor “1”. Por supuesto, para saber de qué estamos hablando necesitamos ver la tabla de libros y la de autores:

Tabla de Libros	
Código de Libro	Título del Libro
1	<i>Programación en Windows 95</i>
2	<i>El Quijote</i>
3	<i>Segunda Fundación</i>
4	<i>Programar en C</i>

Tabla de Autores	
Código de Autor	Nombre del Autor
1	Paul Yao
2	Petzold
12	Asimov
23	Cervantes

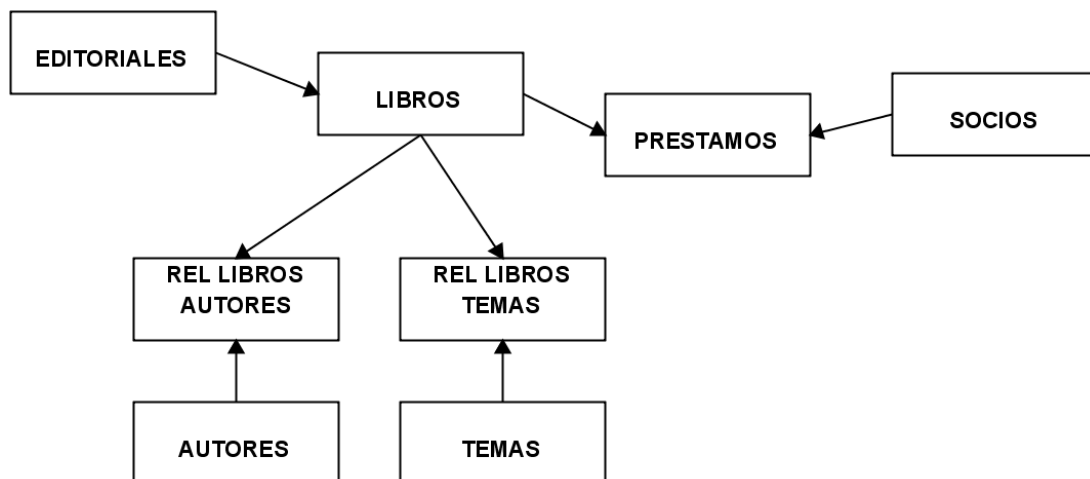
Rellenar a mano las tablas intermedias es muy pesado, pero para eso están hechos los formularios.

A la tabla uno la llamaré por ejemplo Relación Autores y Libros.

A la Tabla Dos le sucede exactamente lo mismo, sólo que con temas y libros. Un libro puede tener varios temas: Literatura y ciencia-ficción por ejemplo, o física y química, del mismo modo que podía tener varios autores. La llamaré Relación Temas y Libros.

Sólo nos queda la Tabla Tres. ¿Qué se puede decir de los socios y los libros? La relación se llamaba "son prestados". Es la tabla Préstamos, donde indicaremos la fecha de préstamo, la fecha de devolución, etc.

La base de datos se queda por tanto de la siguiente forma:

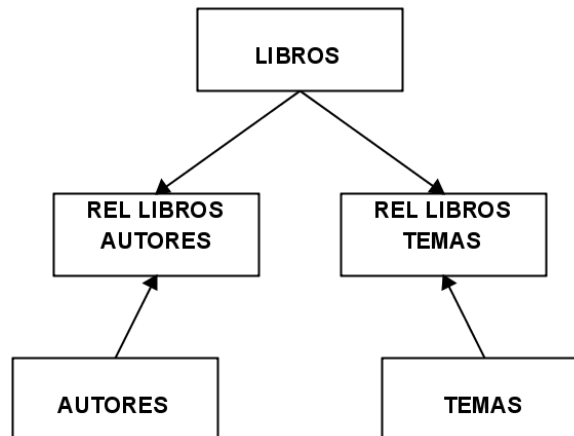


Y los campos de las tablas. Fíjate en qué tablas han provocado la creación de los códigos; Siempre la parte uno de la relación:

- *Editoriales:* Código de Editorial, Nombre de Editorial
- *Autores:* Código de Autor, Nombre de Autor, Apellido 1, Apellido 2, Fecha Nacimiento
- *Temas:* Código de Tema, Nombre del Tema
- *Socios:* Código de Socio, Nombre del Socio, Apellido 1, Apellido 2, Teléfono, Dirección
- *Libros:* Código de Libro, Título, Código de Editorial, Nº de Páginas
- *Rel Libros Autores:* Código de Libro, Código de Autor
- *Rel Libros Temas:* Código de Libro, Código de Tema
- *Préstamos:* Código de Libro, Código de Socio, Fecha de Préstamo, Fecha de Devolución

3.4.5 Árboles de relaciones

A veces las relaciones no ofrecen la información que esperas obtener de ellas. Fíjate en la siguiente construcción:



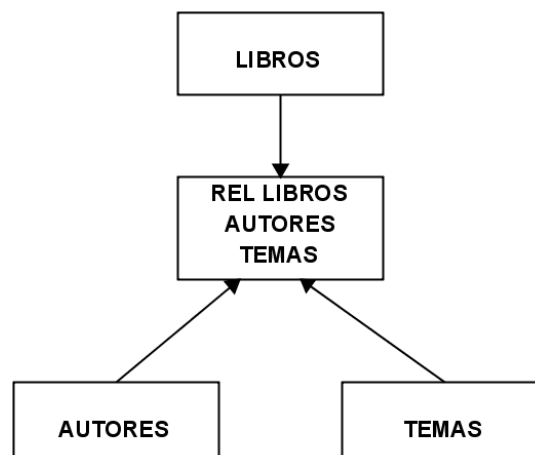
El problema se presenta en los extremos de la figura, Autores y Temas. Las cinco tablas están relacionadas. Sin embargo no tenemos toda la información posible. Dado un libro, y a través de las tablas “Rel Libros Autores” y “Rel Libros Temas” sabemos qué temas y qué autores pertenecen a cada libro. Pero NO sabemos sobre qué tema ha escrito cada autor.

Supongamos el libro “Física y Química”, escrito por Ana y por Pedro. Obviamente los temas de los que trata son Física (Pedro es físico) y Química (Ana es química). Quiero saber sobre qué temas ha escrito Ana. Tengo que recorrer todo el árbol, partiendo de un extremo hasta llegar al otro: Desde Autores tengo que llegar a Libros y de ahí a Temas.

Si parto del Autor “Ana”, llego a Rel Libros Autores y de ahí a Libros. Averiguo que Ana ha escrito “Física y Química”. Pero no me detengo ahí; Desde Libros, sigo hasta Rel Libros Temas y por fin llego a Temas. Averiguo que el libro “Física y Química” trata sobre “Física” y sobre “Química”. *Parece* que Ana ha escrito sobre Física y sobre Química. Y no es cierto. Lo que he averiguado es que el libro en que ha participado Ana trata de esos dos temas. Pero como ese libro ha sido escrito también por Pedro, no tengo forma de averiguar sobre qué temas ha escrito cada autor.

Conclusión. Siempre que dos tablas estén unidas a través de **dos o más relaciones “N”** (que tengan la pinta del dibujo anterior) no están realmente relacionadas. O al menos si preguntamos no obtendremos las respuestas que deseamos...

Si nos interesa saber qué autores han escrito sobre qué temas en cada libro es muy sencillo: Tenemos que rehacer el árbol y crear una estrella:



De esta manera sabremos de qué tema ha escrito cada autor. El problema es que la tabla intermedia es muy pesada de escribir: Código de Autor, Código de Libro, Código de Tema. ¿Merece la pena? La respuesta es fácil: Nos lo dirá el director de la biblioteca. Si el cliente necesita toda la información, tendremos que hacer este último diseño. Si no necesita tanto control, nos quedaremos con el diseño anterior. Para los ejemplos del manual, supongo esta alternativa.

En el apartado siguiente se profundiza sobre qué significa exactamente relacionar tablas, por lo que es posible que todo esto quede más claro.

3.5 Claves

Queda el último paso para que el diseño esté completo. Realmente, es uno de los más sencillos de aplicar. Clave es sinónimo de **ordenación**. Si una tabla está ordenada por nombres y apellidos se dice que tiene una clave por nombre y apellidos. La clave de las páginas amarillas es Profesión, Nombre del Negocio. Está ordenada por profesión. Cuando la profesión se repite, se ordena por nombre del negocio. La guía telefónica está ordenada por apellido primero, apellido segundo, nombre, etc.

A estas alturas ya te habrás dado cuenta de que todo lo que hagas lo vas a hacer a través de consultas. Casi siempre la información que te interese la encontrarás a través de las relaciones entre varias tablas. Con la base de datos actual, imagínate que quieres ver los autores y los libros que han escrito: Necesitas la tabla Libros, la tabla Autores y la tabla que dice qué autor ha escrito qué libro: Rel Autores Libros:



Ponemos las relaciones entre las tablas a mano, elegimos los campos y vemos el resultado:

Consulta1 : Consulta de selección			
	Título	Nombre de Escritor	Apellido 1
►	El Quijote	Miguel	De Cervantes
	Neuromante	William	Gibson
	El palacio de la Luna	Paul	Auster
	La Calle de la Judería	Toti	Martinez
	Leviatán	Paul	Auster
	El tambor de Hojalata	Günter	Grass
	La Música del Azar	Paul	Auster
	Pórtico	Frederik	Pohl
	Java 2	Agustín	Froute
	Programación en Windows 95	Kim	Robinson
	Programación en Windows 95	Paul	Yao
	Historia Universal	Varios	
	Marte Rojo	Kim	Robinson
	Marte Verde	Kim	Robinson
	Marte Azul	Kim	Robinson
*			
Registro: 1 de 15			

¿Qué tabla estamos viendo? ¿Autores o Libros? No estamos viendo NINGUNA de las dos. Fíjate en las líneas correspondientes al libro de "Programación en Windows 95"; aparece dos veces... Lo que estamos viendo es la tabla Rel Libros Autores. Voy a añadir los códigos de dicha tabla a la consulta para que se vea mejor:

Título	Nombre de Escritor	Apellido 1	Cod Libro	Cod Autor
El Quijote	Miguel	De Cervantes	1	1
Neuromante	William	Gibson	2	2
El palacio de la Luna	Paul	Auster	3	3
La Calle de la Judería	Toti	Martinez	4	4
Leviatán	Paul	Auster	5	3
El tambor de Hojalata	Günter	Grass	6	5
La Música del Azar	Paul	Auster	7	3
Pórtico	Frederik	Pohl	8	6
Java 2	Agustín	Froute	9	7
Programación en Windows 95	Kim	Robinson	10	10
Programación en Windows 95	Paul	Yao	10	11
Historia Universal	Varios		11	9
Marte Rojo	Kim	Robinson	12	10
Marte Verde	Kim	Robinson	13	10
Marte Azul	Kim	Robinson	14	10

Siempre que hagas una **consulta**, lo que te mostrará la base de datos será la **información común a todas las tablas**, que siempre corresponderá con la **parte N** de la relación. Si haces una consulta para ver los libros prestados y los nombres de los socios, lo que tendrás en pantalla será una fila por cada línea de la tabla Préstamos. Si quieres ver libros y temas, estarás viendo la tabla Rel Libros Temas.

¿Qué es lo que hace la base de datos? Como has pedido el título del libro y el nombre del escritor, y le has indicado la relación existente entre Rel Libros Autores y esas dos tablas, el ordenador sabe dónde buscar. Por cada fila de la tabla Rel Libros Autores **busca** “Cod Libro” en Libros, y devuelve el Título correspondiente a ese código. Hace lo mismo con “Cod Autor” y Autores. En este caso, ha realizado 30 búsquedas. Dicho de un modo más estricto: Cada registro de la parte N de la consulta implica una búsqueda en la parte uno de la misma.

Ahora pensemos qué significa buscar algo. Cuando buscamos un teléfono en la guía telefónica lo hacemos por apellidos y nombre, ya que está **ordenada** de esa manera. Nunca se nos ocurriría buscar en las páginas amarillas por apellidos y nombre, ya que como no está ordenado siguiendo esa clave tendríamos que leernos cada línea de cada página hasta que encontrásemos lo que buscamos. Si conocemos el teléfono de alguien, podemos averiguar su dirección. Basta con buscar en la guía telefónica ese número. El problema es que como no está ordenada por números...

Al ordenador le pasa exactamente lo mismo, sólo que él no se rinde y busca uno por uno si es preciso. Pues bien, vamos a ponérselo fácil. Ya que cada línea de cada consulta que hagamos provoca una búsqueda en la parte uno de la relación, **vamos a ordenar las tablas siempre por el código de la parte uno de la relación**. Es decir, Libros estará ordenada por código de libro, Autores por código de autor, Tema por código de tema, Socios por código de socio y Editoriales por código de editorial.

Una tabla puede estar ordenada a la vez por varias claves, eso no representa ningún problema para el ordenador. Access usará siempre la que más le convenga. En el capítulo siguiente veremos cómo se hace. Además, una clave tiene otras utilidades: Se usa también para permitir o no la existencia de **información repetida**. Al ordenar, es muy sencillo ver si un valor está repetido, y las bases de datos se aprovechan de ello. Por ejemplo, no tendría sentido escribir en la tabla Rel Libros Autores dos veces al mismo autor para el mismo libro. Nunca lo haríamos a propósito, pero podemos equivocarnos. Una clave que no se pueda repetir nos evitaría ese problema. A esas claves se las llama **claves únicas**.

3.5.1 Claves principales

Los códigos de las partes uno de las relaciones estarán por tanto ordenados, serán una de las claves de sus tablas. Además, serán claves únicas¹¹. Pero ya hemos visto que esos códigos se usan para relacionar unas tablas con otras. Aunque las claves colocadas en esos códigos no tienen nada que las diferencien de otras, las bases de datos las suelen distinguir de las demás para “recordarnos” que ese campo es la forma correcta de enganchar las tablas. Las claves de esos campos se denominan **claves**

¹¹ Es lógico; Imagínate que dos libros tienen el mismo código... O que dos personas tienen el mismo DNI.

principales. La clave principal es el “DNI” de cada fila de la tabla. Access nos recuerda que son la forma correcta de relacionar tablas impidiéndonos dibujar el diseño de la base de datos si no las usamos. Veremos como se hace en el siguiente capítulo.

3.5.2 Claves foráneas

Como ya he dicho, las relaciones entre las tablas no las dibujaremos en cada consulta, sino que las definiremos una vez en “algún sitio” y se quedará almacenado para siempre.

En capítulos siguientes veremos que en Access, a las relaciones entre tablas se las llama... relaciones. Pero ese no es nombre oficial. La denominación correcta es **Clave Foránea o Extranjera** (Foreign Key). El motivo es que los motores de bases de datos internamente tratan del mismo modo las claves y las relaciones.

4

Modelo Entidad/Relación

En este capítulo no vamos a ver conceptos nuevos. Voy a repetir parte de lo explicado en los dos capítulos anteriores, pero usando los métodos y términos formales.

¿Para qué? Con lo que hemos visto hasta ahora tienes más que suficiente para empezar a diseñar bases de datos aceptables. Pero si decides ampliar tus conocimientos y leer textos más técnicos, no vas a comprenderlos. Sencillamente, usan otro idioma. En este capítulo vamos a traducir lo que ya sabes a un lenguaje más riguroso.

Hay muchos métodos para modelar datos. Pero sin duda el más usado es el Modelo E/R. Se definió a mediados de los años setenta, por lo que te puedes imaginar el número de revisiones y ampliaciones que ha sufrido¹². Nosotros veremos sólo lo fundamental, y las mejoras más aceptadas.

Si de momento no tienes intención de leer textos más complicados, sáltate este capítulo. De todos modos, contiene unas cuantas ideas que pueden ser útiles cuando tengas que diseñar una base de datos.

4.1 Método de Diseño

Cuando te enfrentes al problema de informatizar una base de datos, es recomendable seguir una serie de pasos:

1. Captura de requisitos.
2. Esquema conceptual.
3. Implementación de la base de datos.
4. Implementación de las aplicaciones de usuario.

4.1.1 Captura de requisitos

Es un paso fundamental, al que nunca se le dedica el tiempo suficiente. Por sí sólo merece un curso aparte, y sólo la experiencia enseña a realizarlo bien.

Consiste en que te enteres de qué tienes que hacer.

Como ya expliqué en el apartado 3.1, “Un método de diseño”, debes hacer todas las preguntas que se te ocurran al cliente; No sólo al jefe de la empresa (es quien paga), sino a los empleados (son los que saben qué hay que hacer). El objetivo es conocer todo lo que necesitan guardar y qué quieren hacer con ello. Aprende qué hace la empresa, ponte en el lugar del usuario, mira todos sus papeles y programas antiguos... hay muchos libros dedicados al tema, aunque te advierto que sólo la práctica te enseñará lo que necesitas.

¹² El núcleo fundamental no ha cambiado ni un ápice.

Si la captura de requisitos ha ido más o menos bien, tendrás dos tipos de requisitos¹³:

- **Requisitos funcionales.** Qué es lo que hay que hacer y con qué. Operaciones a realizar y con qué datos. Ya que estamos hablando de diseño de bases de datos y no del desarrollo de aplicaciones, podemos subdividir estos requisitos en:
 - **Requisitos de datos.** Tablas, campos y relaciones. Tu trabajo.
 - **Requisitos funcionales** para programadores. Las tareas a realizar. En principio es problema del diseñador de aplicaciones, aunque afecta a tu trabajo. Si hay que imprimir a menudo listados por apellidos y nombre, te están diciendo qué clave secundaria tiene que tener la tabla Socio.
- **Requisitos no funcionales.** Tiempos mínimos de respuesta, base de datos y lenguaje de programación a usar, niveles de seguridad, tipo de red, estructura de la organización, equipos disponibles...

Nos centraremos en los datos. A partir de ellos obtendrás las tablas, los campos y las relaciones entre tablas.

4.1.2 Esquema conceptual. Modelo E/R

Una vez que sepamos de qué se compone nuestra base de datos la representaremos según el **modelo E/R**.

¿Por qué? Porque nos obliga a “pensar” de cierto modo (el número de símbolos es pequeño y su significado muy preciso) y porque es un estándar. Todo el mundo entenderá qué queremos decir. De hecho casi todas las bases de datos profesionales tienen herramientas para convertir el modelo en tablas físicas.

Este capítulo trata casi exclusivamente de cómo convertir nuestros esquemas al modelo E/R.

Además, si tenemos cierto conocimiento de las operaciones que los usuarios quieren realizar sobre nuestra base de datos, podemos usar el modelo para “jugar” con los datos y comprobar que hemos cumplido con todos los requisitos, o que los entendimos correctamente.

El modelado de una aplicación o de una base de datos es un proceso iterativo. Una y otra vez volvemos a los pasos anteriores hasta que estamos convencidos de que funciona adecuadamente.

4.1.3 Implementación del modelo de datos

Muchas bases de datos pueden construirlo todo a partir del modelo E/R. En capítulos posteriores aprenderemos a crear tablas, campos y relaciones en Access y MySQL, aunque sin usar herramientas de diseño.

4.1.4 Implementación de las operaciones

Dicho de otro modo, se trata de programar las aplicaciones que gestionarán la base de datos. Salvo en el caso de Access, que no necesita de programación para funcionar como una aplicación, será necesario escribir código en un lenguaje como Java o C#.

Consulta el capítulo 6, “Access”, para aprender a hacerlo con esta base de datos.

4.2 Componentes del Modelo

Describe los datos como entidades, atributos y relaciones.

¹³ Hay multitud de metodologías de análisis. Esta clasificación es una de tantas, aunque pienso que bastante aceptada.

Entidad es una cosa del mundo real. Un coche, un trabajador, un préstamo de un libro. Se corresponde con una fila de una tabla.

Atributo es una propiedad de la entidad. Un campo de una fila. La entidad correspondiente al Autor “Cervantes”, tendrá el atributo nombre (“Miguel”), el atributo apellido (“Cervantes”), el atributo libros ({{“Quijote”, “Rinconete y Cortadillo”, “Novelas Ejemplares”}}). ¿Eso último es correcto? Pues sí. Ya lo veremos.

Dominio de un atributo es el conjunto de valores que ese atributo puede tener. El dominio de “color” será rojo, verde, azul... El dominio de “teléfono” son todos los posibles números de teléfono del mundo (o de Álava), etc.

Relación o Vínculo: una asociación o relación semántica entre dos entidades. La entidad Socio (“Javi”) tiene una relación con la entidad Libro (“Quijote”): Préstamo (“el día 12 de diciembre”).

Tipo de Entidad es un conjunto de entidades con la misma estructura, con el mismo tipo de atributos. Una tabla, más o menos (pueden ser varias, una relación... depende de qué hayamos diseñado). Lógicamente casi siempre hablaremos de tipos en vez de entidades sueltas. A partir de ahora, cuando leamos el término entidad se sobreentenderá “tipo de entidad”, a no ser que indique lo contrario.

Tipo de Relación. El conjunto de vínculos entre dos o más tipos de entidad. Como en el caso anterior, siempre que veamos “relación” se interpretará como “tipo de relación”.

Ya que se usan los mismos nombres para referirse tanto a las entidades o relaciones como a sus conjuntos, es una buena práctica nombrar a ambos tipos de objetos en singular, para que el nombre valga tanto para el conjunto como para sus elementos.

4.3 Entidades

Las entidades (se sobreentiende “tipos o conjuntos de entidades”) se representan con un rectángulo etiquetado con un **nombre**. El rectángulo puede ser simple o doble. Cuando estudiemos las relaciones veremos qué diferencia hay entre ambos símbolos:



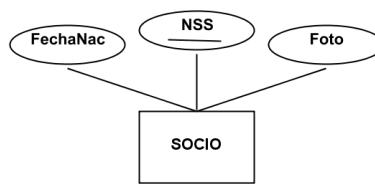
Para el ejemplo de la biblioteca, podemos pensar en las siguientes entidades. Simplemente las escribiré. No dibujaremos el esquema completo hasta haber visto todos los elementos del modelo:

SOCIO
LIBRO
AUTOR
TEMA
EDITORIAL

Como hemos hecho el ejemplo completo en el capítulo 3, sabemos las tablas que faltan: Préstamo, Rel Libros Autores, Rel Libros Temas. No las he incluido porque digamos “que no se me han ocurrido”. Las guardo para ejemplos posteriores con atributos y relaciones.

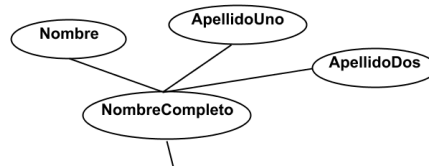
4.4 Atributos

En el modelo E/R se distinguen varios tipos de atributos, en función de diferentes conceptos. En el modelo E/R se representan como un óvalo con el nombre del atributo y conectado a la entidad a la que pertenecen:



Simples o Atómicos. No pueden dividirse en atributos más simples.

Compuestos. Son útiles para indicar que “algo” a veces será usado como un todo y a veces por sus componentes por separado; Por ejemplo, “Socio Javi (Javier, Rodríguez, Díez)”. Se representa:



Monovaluados. Sólo admiten un valor. Edad, nombre de pila, peso, precio.

Multivaluados. Admiten varios valores a la vez: Teléfonos del socio, titulaciones: “{900100010,615754234}”. Obviamente, cuando convirtamos el modelo a tablas no se quedarán así: Serán otras tablas, relaciones... Pero tal vez nos interese tenerlos de ese modo en el esquema E/R. Su dibujo:



Derivados. Datos que se pueden obtener de otros: Fecha de nacimiento y edad, precio y cantidad con respecto a total, etc.

Almacenados. Datos “normales”, que no se pueden calcular a partir de otros.

Atributos clave. Los atributos o conjuntos de atributos que forman la clave principal de una entidad. Se representan con el nombre del atributo subrayado (ver el ejemplo de Socio y NSS de la figura de arriba).

Y por supuesto, combinaciones de diferentes tipos.

Los atributos para las entidades anteriores:

```
SOCIO
NombreCompleto(Nombre, Apellido1, Apellido2), NSS,
{LibrosPrestados}, Teléfono, Dirección
```

NombreCompleto lo he definido como un atributo compuesto. Los libros que le han prestado es un atributo multivaluado. Ya sabemos que eso no es un atributo, sino que estamos expresando que hay una relación con la tabla libros. Lo veremos más adelante.

```
LIBRO
Código, Título, {Autores}, Editorial, Numpáginas, {Temas}
AUTOR
Código, NombreCompleto(Nombre, Apellido1, Apellido2) , {Libros}
TEMA
Nombre, {Libros}
EDITORIAL
Nombre, {Libros}
```

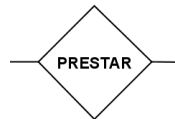
Como se puede ver, todos los atributos multivaluados y unos cuantos normales no son tales. Son expresiones de relaciones que tenemos que definir en el apartado siguiente.

¿Por qué he incluido el atributo “Código” en algunas entidades? Porque no había ningún atributo o combinación de ellos que me sirviera de clave principal. Sabemos que añadiremos claves numéricas a todas las tablas, pero no “queda elegante” hablar de detalles de implementación cuando estamos en esta

etapa de diseño. Es mejor buscar claves principales en los atributos existentes, ya que ayudan a comprender qué es exactamente la entidad.

4.5 Relaciones

Son los vínculos, las conexiones semánticas entre las entidades. Como ya he comentado, hablaremos por lo general de conjuntos de relaciones. Se representan con un rombo unido a las entidades que relaciona por líneas. En su interior un **verbo** describe la acción que realiza:

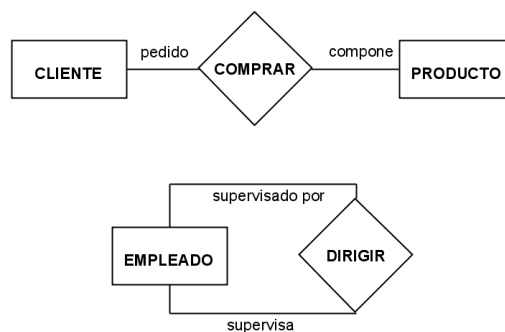


A menudo, mientras describimos las entidades y atributos que las componen aparecerán **relaciones implícitas**. Sucede cuando un atributo hace referencia a otras entidades; Por ejemplo, desde LIBRO se habla de {temas}, {autores} y editorial.

Por **grado** se entiende el número entidades que participan en la relación. Obviamente siempre será mayor o igual que dos.

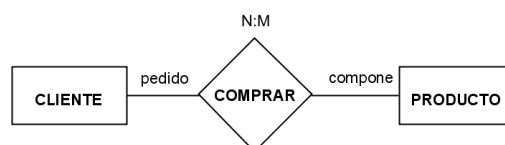
4.5.1 Roles

A veces es conveniente asignar **roles** a los papeles que desempeña cada entidad en la relación; Ayuda a comprender la semántica del modelo que estamos construyendo. No son obligatorios, salvo en las relaciones con redundancia. Sencillamente, se trata de poner nombres a las líneas que unen la relación con la entidad:



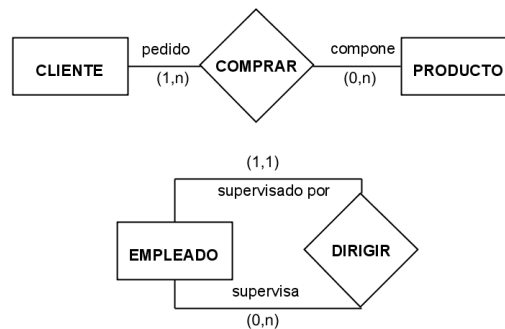
4.5.2 Multiplicidad

Y por supuesto, falta la **cardinalidad o multiplicidad** de la relación. Como ya viste en el anterior capítulo, lo único que puede representar una base de datos relacional son las cardinalidades de **1 a 1**, de **1 a N** y de **N a M**:



Se lee como siempre. Un Cliente puede Comprar “N” Productos, o un Producto puede ser Comprado por “M” Clientes

Pero ahora no estamos hablando de bases de datos implementadas, sino de nuestro modelo de los datos. Puedes representar la multiplicidad de la relación de la manera tradicional o bien de un modo ligeramente distinto:



En vez de expresar la multiplicidad de toda la relación, la indicamos para cada rol por separado. Se escribe “**(a,b)**”, donde “a” representa la cardinalidad de la entidad conectada a la relación (la que está pegada al rombo) y “b” la de la otra (la más alejada).

La relación entre cliente sigue siendo de 1 a N. Sin embargo lo que hemos escrito ahora expresa más significado:

- Empezando por el rol “pedido” se lee así: un Cliente Compra n Productos. Las cardinalidades expresadas de este modo pueden ser **cualquier** número, incluido el cero. Expresado de ese modo significa que en tu base de datos no quieres clientes que no tengan compras (otra cosa es que sea una buena idea No lo es).
- El segundo rol dice que cero o un Producto puede ser Comprado por n Clientes. Permite que haya Productos que nunca hayan sido comprados.

¿Y si escribes algo como “(3,34)”?

Es válido **en el modelo**, pero seguramente no podrás expresarlo en la base de datos real. El modelo es una representación idealizada, que sirve para comprender mejor el problema. Implementarlo físicamente es otra historia, y seguramente necesitarás que un programador añada restricciones en las aplicaciones que accedan a tus tablas.

En el caso de Empleado, tenemos una relación recursiva de 1 a N, pero hemos expresado la multiplicidad de cada rol.

- Los empleados tienen jefes que les Dirigen. En el rol “supervisado por” leemos (1,1). Un empleado es siempre supervisado por un “jefe directo. Un empleado siempre tiene un jefe directo y sólo uno.
- El rol “supervisa” expresa la relación entre Jefes y Empleados. Pone (0,n): “cero o un empleado” supervisa a n Empleados: Es decir, que no es obligatorio que “un” empleado sea jefe de nadie. Si hubiéramos escrito (1,n) sería una empresa muy rara...

4.5.2.1 Propagar atributos

Una diferencia con respecto a lo estudiado en el capítulo anterior, es que podemos representar atributos en las relaciones. Pero sabemos que las relaciones desaparecerán o se convertirán en tablas cuando implementemos en modelo. ¿Qué pasa con esos atributos?

La respuesta es muy sencilla, y ya hemos hecho tareas similares. Los atributos se trasladan a las tablas en función de su multiplicidad:

- Relación de **1 a 1**: El atributo se puede trasladar a cualquiera de las entidades participantes. En el capítulo tres vimos que generalmente unimos las dos entidades en una única tabla.

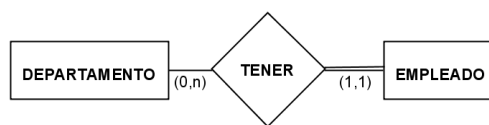
- Relación de **1 a N**: El atributo siempre va a la entidad n de la relación. Sabemos que la relación se convierte en códigos comunes a ambas tablas, por lo que “desaparece”. Obviamente, la información irá allá donde cabe, la tabla n.
- Relación de **N a M**: Los atributos pertenecen a la relación, ya que ésta se convierte en una tabla.

4.5.3 Participación Total

Cuando en un tipo de entidad perteneciente a un tipo de relación **todas** sus entidades participan en al menos una de las relaciones se dice que su participación es **total**.

Por lo general representarlo o no en el modelo E/R no cambiará el resultado final de tus tablas. Como en el caso anterior, se trata de comprender mejor el problema.

Se representa con una línea doble uniendo el tipo de entidad al tipo de relación:



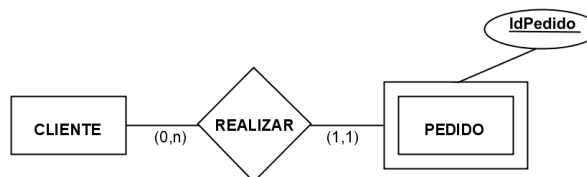
Un Departamento tiene de cero a n empleados. Permiso crear departamentos vacíos. Un empleado siempre pertenece a un departamento. Y además, lo he recalcado indicando que las entidades Empleado siempre interviene en la relación Tener.

Como ves, la participación total ya está expresada en la cardinalidad de cada rol. Úsala si piensas que aclara tu modelo. En el esquema del final del capítulo usaré ambas, aunque yo prefiero expresar simplemente la cardinalidad.

4.5.4 Entidades Débiles

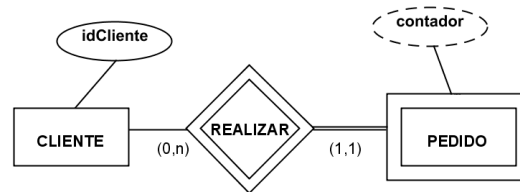
Puede ser útil incluirlas en nuestros modelos, ya que ayudan a entender el proyecto. Este apartado habla de cierto tipo de relación entre entidades, por eso lo estudiamos aquí.

Una entidad es **débil** cuando su existencia depende completamente de otra entidad a la que está vinculada a través de una relación. Se representa con un rectángulo doble:



No puede existir un pedido sin cliente. Cliente es la entidad fuerte de la relación y Pedido la entidad débil. Se dice que existe una **dependencia de existencia** entre Cliente y Pedido.

Hay casos en los que esa dependencia va más allá. Fíjate en el atributo clave que le he asignado a pedido. A cada entidad pedido le asigno un número diferente, por lo que todos estarán identificados correctamente. Pero supongamos que la clave de Pedido es una clave compuesta por el código de cliente y un contador en función del número de pedido de cada cliente:



Hay una **dependencia de identificación** de Pedido con respecto a la relación. Necesita la relación no sólo para existir, sino también para poder identificarse. El atributo de clave parcial se dibuja con una línea discontinua. El atributo clave que falta se puede poner en la entidad o en la misma relación, dependiendo de lo que nos interese.

La relación que establece la dependencia se simboliza con un rombo doble. Como siempre, se trata de clarificar el modelo. Si no te resulta útil o no aporta nada, no lo uses.

Ahora podemos por fin buscar las relaciones de nuestro modelo. Tal como hemos diseñado las entidades, resulta que todas las relaciones que necesitamos son implícitas. Ya están expresadas como atributos de entidades, por ejemplo entre Autor y Libro

```
LIBRO
Código, Título, {Autores}, Editorial, Numpáginas, {Temas}
AUTOR
Código, NombreCompleto(Nombre, Apellido1, Apellido2) , {Libros}
```

Autor tiene el atributo multivaluado {Libros} y Libro tiene {Temas}. Obviamente, hay una relación de n a m entre ambas entidades.

Las relaciones que podemos deducir:

```
PRESTAR
De N a M entre Socio y Libro. No hay dependencias totales, o dicho
de otro modo, la cardinalidad de cada rol es (0,n) y (0,n).
La relación tiene dos atributos: Fprestamo y Fdevolución.
EDITAR
De 1 a N entre Editorial y Libro. Dependencia total de Libro (no
puede existir un libro sin editorial), o (1,1)...
ESCRIBIR
De N a M entre Autor y Libro. Decido que hay dependencia total de
libro. No se me ocurren atributos de la relación.
TRATAR
De N a M entre Tema y Libro No hay dependencias totales, permito
libros sin temas. No hay atributos en la relación
```

Observa la dependencia total (1,n) de Libro con respecto de Escribir. ¿Cómo se implementa eso en el diseño de la base de datos? Necesitas de programación adicional para asegurarte que un libro tiene asociado al menos un escritor. Esa parte del diseño es para el programador, no para ti. Al menos hasta que aprendas Triggers (disparadores, eventos) de MySQL...

Si eliminamos los atributos que nos han servido para deducir las relaciones, los tipos de entidad quedan del siguiente modo:

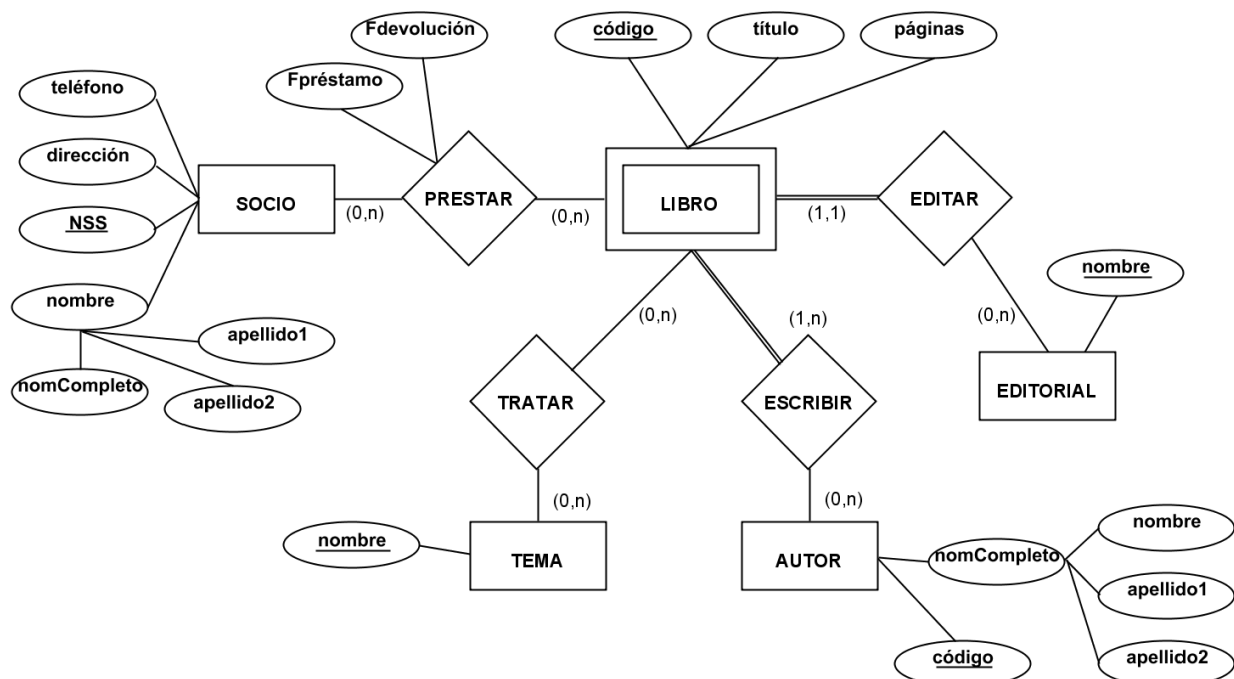
```
SOCIO
NombreCompleto(Nombre, Apellido1, Apellido2), NSS, Teléfono,
Dirección
LIBRO
Código, Título, Numpáginas
AUTOR
Código, NombreCompleto(Nombre, Apellido1, Apellido2)
```

TEMA
Nombre
EDITORIAL
Nombre

4.6 El Diagrama E/R

El modelo debe servir de ayuda para el diseño. Se trata de que el equipo de trabajo comprenda qué hay que hacer y sepa si se cumplen los requisitos de datos. Haz las modificaciones que consideres oportunas, siempre que el resto de la gente esté al tanto. Por ejemplo, es muy habitual no incluir los atributos en modelos extensos, para centrarse en las entidades y relaciones.

El diagrama completo del ejemplo queda del siguiente modo:

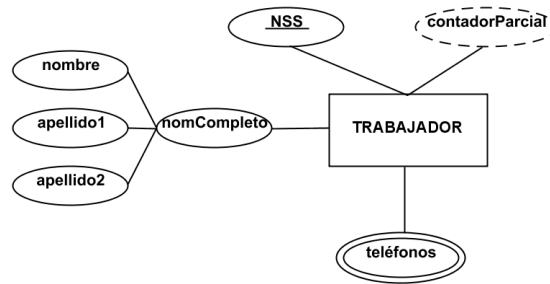


¿Cómo sería la implementación? Eso ya lo sabemos por los capítulos anteriores:

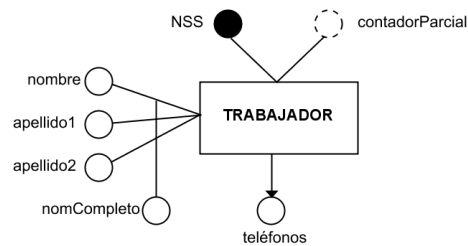
- Las relaciones de N a M son tablas. Crearemos las tablas Prestar (o Préstamo), Tratar (o RelaciónLibroTema) y Escribir (o RelaciónLibroAutor), con los campos adecuados (consulta el capítulo 3).
- En este caso el atributo compuesto “nomCompleto” se convertirá en tres campos: nombre, apellido1 y apellido2.
- Por motivos de eficiencia, no crearemos claves principales a partir de textos, sino que haremos campos “código” para tal función, a no ser que en nuestra base de datos se prefieran los textos.

4.6.1 Otros Símbolos

Los atributos suelen representarse a veces de otra manera, usando pequeños círculos y colocando el nombre del atributo aun lado de éstos. Supongamos el siguiente diagrama:



Con el conjunto alternativo de símbolos para los atributos quedaría:



4.7 Modelo Extendido

Unos cuantos de los símbolos y conceptos que hemos usado no forman parte del modelo E/R original. Pero hace mucho que se aplican, por lo que se consideran parte del “modelo clásico”.

Sin embargo hay otra serie de ideas que no hemos visto, y que forman parte del **Modelo Extendido** o **Modelo ERR** (Enhanced Entity Relationship). Permite relaciones de especialización y exclusividad, así como agregación de entidades. Sin entrar en detalles, puede representar mejor que el modelo tradicional la realidad que se quiere implementar en una base de datos. Por ejemplo, se puede describir la herencia de los lenguajes orientados a objetos.

El problema es que las capacidades de una base de datos siguen siendo las de siempre. Al final tendrás que traducir el modelo a algo que se pueda escribir.

Es útil para representar bases de datos orientadas a objetos. Pero parece que la gente prefiere usar diagramas de UML¹⁴ para esta tarea... No te metas con ERR a menos que lo necesites (que te lo pida tu jefe).

¹⁴ Lenguaje de Modelado Unificado. Se suele usar junto a Proceso Unificado (PU), la metodología de diseño de objetos más aceptada en la actualidad (en alguna de sus decenas de variantes).

5

Formas Normales

5.1 Definición

Las **formas normales** son reglas que tienen que cumplir las tablas de una base de datos para que se considere que están **normalizadas**, esto es, para evitar redundancias, problemas de integridad referencial, de actualización de datos, etc. Obviamente nos interesa mucho que una base de datos esté normalizada.

Pero hay un problema. Se basan en la descripción matemática de las relaciones entre las tablas. Las formas normales son definiciones matemáticas de la teoría de conjuntos.

No quiero perder un montón de horas explicando teoría de conjuntos y lógica. Por tanto, explicaré las formas normales en lenguaje llano (bueno, más o menos). Es más sencillo entender “los campos deben identificarse la clave principal al completo” que “cada atributo no principal debe tener dependencia funcional completa respecto de las claves candidatas”. Pero **no es lo mismo**. A medida que veamos formas normales más complejas tendré que complicar el lenguaje.

Como siempre, eso dará problemas en bases de datos avanzadas. Con el contenido de este capítulo tendrás más que suficiente para las que vas a diseñar a medio y largo plazo. De hecho, tendrás demasiado... Y no te preocupes. Si necesitas métodos de diseño más profesionales significa que ya has metido un montón de horas de práctica, con lo que te será fácil aplicarlos.

Dos consejos antes de seguir con el capítulo:

- Haz tus diseños lo más perfeccionados posibles. Normalizar es un proceso laborioso. Si tu diseño está cuidado es más que probable que esté normalizado; Al menos con bases de datos sencillas.
- Cuidado al normalizar. Que una tabla esté normalizada no quiere decir que sea correcta. Es posible que al aplicar las formas normales rompas la semántica de la relación. En vez de una base de datos con redundancias tienes una base de datos que no funciona.

5.2 Conceptos Básicos

No voy a aplicar demostraciones matemáticas, pero sí debo explicar algunos conceptos. Alguno de ellos ya ha sido explicado en el capítulo 4, “Modelo Entidad/Relación”.

Atributo. Es lo mismo que campo o columna, pero desde un punto de vista más teórico. El campo sería “código de editorial”, mientras que el atributo sería “editorial”, sin especificar componentes, o si es un número, etc.

Dominio. Un atributo pertenece a un dominio. Es el conjunto de datos que puede almacenar un atributo. Algo así como el tipo de datos, pero de forma abstracta. “Código de editorial” tiene el tipo de datos “integer”, pero el atributo “editorial” tiene como dominio el conjunto de todas las posibles editoriales.

Dependencia funcional. La teoría se basa en las dependencias funcionales entre los atributos de una tabla. Para no meternos en definiciones matemáticas, entenderemos el término “dependencia” como “**implica, determina**”. Por ejemplo, existe una dependencia entre “DNI” y “Nombre y Apellidos” de alguien. Dado cierto DNI, sólo puede asociarse a cierto Nombre y Apellidos. El DNI “implica” ese Nombre y Apellidos¹⁵. A la hora de referirnos a alguien podemos usar su nombre o su DNI.

Se representa $DNI \rightarrow \text{Nombre}$. DNI es el **determinante** de la dependencia funcional.

Dependencia funcional transitiva. Muy sencillo. Si A implica B, y B implica C, A implica C:

1. La calle Dato está en Vitoria
2. Vitoria está en Álava
3. La calle Dato está en Álava

Dependencia multivaluada o multivalor. También muy sencilla de entender. En el siguiente ejemplo, esta dependencia se produce entre Autor (el determinante) y Libro:

Autor	Libro
Terry Pratchett	¡Guardias, Guardias!
Terry Pratchett	Pies de Barro
Terry Pratchett	Imágenes en Acción
Cervantes	Novelas Ejemplares

Existe dependencia multivaluada cuando el determinante implica a varios valores a la vez. Sencillamente, a cierto campo le “corresponden” varias filas de la tabla. Se representa $\text{Autor} \twoheadrightarrow \text{Libro}$.

Clave candidata. En una tabla la clave principal es fundamental. Sin embargo, cuando usamos modelos más estrictos se prefiere el término Clave Candidata. Son todas las claves que podrían funcionar como claves principales de una tabla, aunque no lo sean. Aunque al diseñar escojas una clave principal, es posible que cierta combinación de campos pueda también identificar de manera única a la fila. Siempre que hablemos de claves en las formas normales nos referiremos a claves candidatas.

Atributo principal, primario o primo. Es un atributo que pertenece a una clave candidata. No principal, que no pertenece a ninguna clave.

Tabla o Relación. En el capítulo anterior hemos hecho una distinción entre los dos conceptos. Tabla es el sitio donde están los datos y Relación es la unión semántica que existe entre dos tablas: A veces se representa con códigos comunes a ambas y otras veces implica crear una tabla intermedia.

Las formas normales no distinguen entre ambos conceptos. Desde un punto de vista estricto, sólo hablan de “relaciones entre los atributos”, sin distinguir si después se convertirán en una o varias tablas. De hecho el objetivo de la normalización es que el paso de “relaciones teóricas” a “tablas de verdad” sea lo más eficiente posible¹⁶.

Nosotros sí que vamos a diferenciar entre ambos conceptos; He explicado esto por si decides leer textos más teóricos sobre las formas normales.

5.3 Primera Forma Normal (1NF)

La primera y segunda forma normal se consideran un paso intermedio en la definición de la tercera. Se suelen representar por su número seguido de sus iniciales en ingles.

¹⁵ Se dice: “Existe una dependencia funcional total entre DNI y Nombre y Apellidos”.

¹⁶ El paso del “modelo entidad/relación” al “modelo relacional”.

La primera forma normal dice que el dominio de los atributos debe ser atómico, indivisible y los contenidos de los campos deben ser únicos. O “en un campo sólo cabe una cosa”.

El ejemplo típico de un dominio incorrecto:

Nombre	Apellidos
Ana	Gonzalez Perez
Pedro	Armendi Lopez
Juan	Arregui Olaza

El dominio del atributo “apellidos” no es atómico. Se puede separar en dos dominios distintos, “apellido 1” y “apellido 2”. Convertida a 1FN queda:

Nombre	Apellido 1	Apellido 2
Ana	Gonzalez	Perez
Pedro	Armendi	Lopez
Juan	Arregui	Olaza

Con respecto al contenido único de los campos es fácil de corregir, aunque es un error común entre principiantes. Es el típico ejemplo de valores separados con comas o espacios:

Nombre	Titulaciones
Ana	Medicina
Pedro	Teología, Psicología
Juan	Tornero Fresador

La tabla anterior es una pesadilla a la hora de buscar y actualizar. Para pasarla a 1FN tenemos que añadir más filas a la tabla.

Nombre	Titulaciones
Ana	Medicina
Pedro	Psicología
Pedro	Teología
Juan	Tornero Fresador

5.4 Segunda Forma Normal (2NF)

Se aplica a tablas cuyas claves se compongan de varios campos. Una tabla está en 2NF si está en 1NF y el resto de atributos no principales de la tabla “dependen funcionalmente” de todas las claves candidatas al completo. Si se elimina un campo de la clave, esa dependencia debe romperse.

Lógicamente, las tablas cuyas claves sólo se compongan de un campo ya están en 2NF.

Es decir, que si existe una clave compuesta, trozos de esa clave no pueden funcionar como clave candidata de ningún otro campo de la tabla. Supongamos el siguiente ejemplo. Los campos que forman la clave principal están subrayados:

<u>CodSocio</u>	<u>CodLibro</u>	<u>FechaPréstamo</u>	<u>FechaDevolución</u>	NombreSocio
1	1	01/12/2009	06/12/2009	Javi
1	12	07/12/2009	NULL	Javi
2	3	5/12/2009	NULL	Ana
1	1	07/12/2009	NULL	Javi

La clave es compuesta: {CodSocio, CodLibro, FechaPréstamo}. La fecha de devolución depende de los tres campos a la vez. No tiene sentido una fecha de devolución sin el código de socio o de libro. Pero el nombre del socio depende únicamente de código de socio. No hay una dependencia total de la clave principal.

Ya que está en 1NF, para pasar la tabla a 2NF:

<u>CodSocio</u>	<u>CodLibro</u>	<u>FechaPréstamo</u>	<u>FechaDevolución</u>
1	1	01/12/2009	06/12/2009
1	12	07/12/2009	NULL
2	3	5/12/2009	NULL
1	1	07/12/2009	NULL

<u>CodSocio</u>	<u>NombreSocio</u>
1	Javi
2	Ana

Veamos un ejemplo con una clave candidata que no es la clave principal (son un poco más retorcidos). Supongamos que cierto producto *sólo* puede venderlo un proveedor.

<u>CodProveedor</u>	<u>CodProducto</u>	<u>NombreProd</u>	<u>TelProveedor</u>
1	P3	Tornillos del 37	900100010
2	P7	Bielas	991122334
2	P8	Conducciones	945945945
3	P1	Bisagras	678123456

Hemos escogido como clave principal el nombre del producto. Como no se puede repetir vale perfectamente. Está en 2NF porque obviamente no lo puedo partir en dos. Pero otra clave principal podría haber sido CodProveedor, CodProducto. Como ya he indicado al principio, en mi base de datos no puede repetirse esa combinación. Y TelProveedor depende sólo de CodProveedor. La tabla no está en 2NF. Para normalizarla:

<u>CodProveedor</u>	<u>CodProducto</u>	<u>NombreProd</u>
1	P3	Tornillos del 37
2	P7	Bielas
2	P8	Conducciones
3	P1	Bisagras

<u>CodProveedor</u>	<u>TelProveedor</u>
1	900100010
2	991122334
3	678123456

Si tienes un poco de práctica, *parece* que la tabla de arriba está pidiendo a gritos que la rompas y hagas otra tabla sólo con código de producto y nombre. Pero recuerda la condición inicial: "un producto sólo puede venderlo un proveedor".

5.5 Tercera Forma Normal (3NF)

Para que una tabla esté en 3NF debe estar en 2FN y además ningún atributo no principal puede depender transitivamente de una clave.

Supongamos la siguiente tabla:

<u>NumeroTrab</u>	<u>NombreTrab</u>	<u>NumeroDep</u>	<u>NombreDep</u>
1	Ana	1	Contabilidad
2	Pedro	1	Contabilidad
3	María	1	Contabilidad
4	Aitor	2	Mantenimiento

Está en 2NF, ya que su clave no se puede descomponer. Claves candidatas, dos: NúmeroTrab, que la he escogido como clave principal y NombreTrab. Cualquiera de las dos valdría para identificar de manera unívoca a la tabla. En el mundo real nunca escogeríamos los nombres y apellidos como clave porque sabemos que pueden repetirse, pero ahora estamos trabajando en modelos teóricos...

La tabla no está en 3NF. Dado el número del trabajador (o el nombre, en este caso da igual) tenemos el número de departamento en el que trabaja. Y dado el número de departamento, tenemos el nombre del mismo. Por tanto, dado un trabajador, tenemos el nombre del departamento:

1. Número del trabajador → Número de departamento
2. Número de departamento → Nombre de departamento
3. Número del trabajador → Nombre de departamento

Para pasar la relación a 3NF:

<u>NumeroTrab</u>	<u>NombreTrab</u>	<u>NumeroDep</u>
1	Ana	1
2	Pedro	1
3	María	1
4	Aitor	2

<u>NumeroDep</u>	<u>NombreDep</u>
1	Contabilidad
2	Mantenimiento

Este ejemplo es fácil de ver a simple vista. Pero cuidado; a veces las relaciones semánticas entre los atributos son difíciles de detectar.

5.6 Forma Normal de Boyle-Codd (BCFN)

Se pensó como una simplificación de 3NF, pero resultó una forma normal más restrictiva que la anterior.

Una tabla está en BCFN si está en 3NF y además los atributos no principales únicamente dependen de las claves candidatas. No pueden existir dependencias entre atributos, a no ser que sean claves (por supuesto, de **toda** la clave, o no estaría en 2NF y por tanto tampoco en 3NF).

Casi todas las relaciones que están en 3NF ya están en BCFN. Hay que ser un poco retorcido para encontrar un caso en el que no sea así. Veamos un ejemplo¹⁷.

Supongamos un grupo de profesores que imparten ciertas asignaturas a un grupo de estudiantes. Se cumplen las siguientes restricciones:

1. Un profesor sólo puede impartir una materia.
2. Un estudiante sólo tiene un profesor por cada materia (Mi profesor de mates es el mismo durante todo el curso)

De forma más estricta las “dependencias funcionales” anteriores se representarían:

DF1 {Estudiante, Materia} → Profesor

DF2 Profesor → Materia

<u>Estudiante</u>	<u>Materia</u>	<u>Profesor</u>
Ana	Lenguaje	Mateo
Nagore	Lenguaje	Marcos
Nagore	Matemáticas	Lucas
Nagore	Dibujo	Juan
Luis	Lenguaje	Mateo
Luis	Matemáticas	Pedro
Javier	Lenguaje	Tiago
Aitor	Lenguaje	Marcos

Las clave candidata de la relación es {Estudiante, Materia}. Si se cumple BCNF, las únicas dependencias posibles son las que dependan de la clave, es decir, DF1. Pero está DF2. El atributo Profesor (que no es una clave) implica la Materia. Para pasarla a BCNF:

<u>Estudiante</u>	<u>Profesor</u>
Ana	Mateo
Nagore	Marcos
Nagore	Lucas
Nagore	Juan
Luis	Mateo
Luis	Pedro
Javier	Tiago
Aitor	Marcos

<u>Profesor</u>	<u>Materia</u>
Mateo	Lenguaje
Marcos	Lenguaje
Lucas	Matemáticas
Juan	Dibujo
Pedro	Matemáticas
Tiago	Lenguaje

¿No puedo normalizar la tabla de otra manera? Otras dos soluciones:

- {Estudiante, Profesor} y {Estudiante, Materia}
- {Profesor, Materia} y {Materia, Estudiante}

¹⁷ Lo he copiado de “Fundamentos de Sistemas de Bases de Datos”, de Ramez Elmasri y Shamkant B Navathe, ed. Pearson.

Si las piensas verás que están mal. Están **normalizadas**, pero has roto las dependencias entre los atributos, la **semántica** de la relación.

En el primer caso, sé que Materia y qué Profesor tiene cada estudiante. Pero si trato de averiguar qué Materia imparte cada Profesor, me resulta imposible. Obtendré el producto cartesiano de los Profesores y Materias de cada estudiante. Dicho de otro modo, ¿Dónde expresas DF2, Profesor \rightarrow Materia?

En el segundo caso, DF2 está escrita de forma explícita. Pero si quiero saber qué Profesor imparte clases a cada Estudiante, obtengo una lista de todos los Profesores que imparten esa Materia. No sé qué Profesor tienen cada Estudiante. He roto DF1, {Estudiante, Materia} \rightarrow Profesor.

Estudiemos ahora la solución correcta. En el ejemplo con datos se ve perfectamente que cumple con la semántica de la relación, pero vamos a explicarlo del mismo modo que las otras dos soluciones; Es un poco liso, pero merece la pena:

- {Estudiante, Profesor} y {Profesor, Materia}

¿Se ha mantenido la semántica de las relaciones? Vemos que DF2, Profesor \rightarrow Materia está escrita de forma explícita. DF1 es más difícil de ver. No está expresada directamente, pero podemos llegar a ella; Usando matemáticas la demostración sería trivial. No se pudo tener todo...

Sabemos qué Profesores tiene cada Estudiante: Es la relación {Estudiante, Profesor}. Como DF2 está expresada ({Profesor, Materia}), sabemos qué Materia cursa cada Estudiante. Por tanto, de manera informal, podemos “reemplazar” Profesor por Materia en la primera tabla; Por otro lado, como la combinación Estudiante – Profesor es única nos queda DF1, {Estudiante, Materia} \rightarrow Profesor.

5.7 Cuarta Forma Normal (4NF)

Una tabla está en 4NF si está en BCNF y todas sus dependencias multivalor tienen como determinante únicamente claves candidatas.

Es muy fácil de entender con un ejemplo. Supongamos las ventas de unos comerciales. Se quiere guardar lo vendido y dónde lo han hecho. Los vendedores tienen asignadas ciertas zonas y ciertos productos. En sus zonas pueden vender cualquiera de sus productos, aunque no tienen por qué tener exclusividad. La tabla muestra todas las posibles combinaciones:

DF1 Vendedor \rightarrow Producto

DF2 Vendedor \rightarrow Zona

<u>Vendedor</u>	<u>Producto</u>	<u>Zona</u>
Ana	Tornillos	Vitoria
Ana	Tuercas	Bilbao
Eva	Brocas	Bilbao
Luis	Tornillos	Vitoria
Luis	Tornillos	Bilbao
Luis	Brocas	Vitoria
Ana	Clavos	Vitoria
Ana	Tuercas	San Sebastián

La única clave candidata es {Vendedor, Producto, Zona}. Cumple todas las formas normales anteriores, ya que sólo hay atributos principales (vale, es hilar muy fino, pero es sólo un ejemplo). Sin embargo vemos que las dependencias entre Vendedor y Producto y Vendedor y Zona son multivaluadas: Hay varias filas por cada vendedor en ambos casos:

DF1 Vendedor \twoheadrightarrow Producto

DF2 Vendedor \twoheadrightarrow Zona

No cumple la 4NF. Para que sí lo haga creamos las siguientes tablas:

<u>Vendedor</u>	<u>Producto</u>
Ana	Tornillos
Ana	Tuercas
Ana	Clavos
Eva	Brocas
Luis	Tornillos
Luis	Brocas

<u>Vendedor</u>	<u>Zona</u>
Ana	Vitoria
Ana	Bilbao
Ana	San Sebastián
Eva	Bilbao
Luis	Vitoria
Luis	Bilbao

Observa que hemos eliminado algunas filas (hemos eliminado redundancia). Cuidado al aplicar esta forma normal. Es posible que rompas la semántica de las relaciones.

Por lo general, hace falta esta normalización si en el diseño no has detectado alguna relación de varios a varios. Cuanto más depurado sea tu diseño, menos tendrás que normalizar.

5.8 Quinta Forma Normal (5NF)

Definirla explícitamente exige aprender unos cuantos conceptos nuevos: dependencias de unión, proyecciones... y además NO vas a aplicarla, al menos durante bastante tiempo. Por ello simplemente voy a describir qué es y cómo se usa.

Se aplica cuando una tabla tiene muchos campos, o pocos campos pero un número muy grande de filas: En ambos casos se puede reducir el rendimiento de la base de datos al trabajar con esas tablas. La forma de solucionarlo es **dividir** la tabla en otras más pequeñas, de tal manera que si unimos dichas tablas usando únicamente las claves candidatas de la tabla original restauremos toda la información. Si además ya estaba en 4NF, esa relación (todas esas tablas) está en 5NF.

Supongamos la tabla de Préstamos de la biblioteca; Contiene qué socio se ha llevado un libro, y cuándo lo ha hecho. Lógicamente es una tabla con muchos registros, y que crece rápidamente:

<u>Socio</u>	<u>Libro</u>	<u>Fecha</u>
S1	L1	F1
S2	L2	F2
S1	L20	F1
...		
S50	L1	F1
S1	L1	F99

La relación está en 4NF. Para que esta relación esté en 5NF tenemos que dividirla en las siguientes proyecciones:

<u>Libro</u>	<u>Fecha</u>
L1	F1
L2	F2
L20	F1

<u>Socio</u>	<u>Libro</u>
S1	L1
S2	L2
S1	L20

<u>Socio</u>	<u>Fecha</u>
S1	F1
S2	F2
S1	F1

...	
L1	F1
L1	F99

...	
S50	L1
S1	L1

...	
S50	F1
S1	F99

Las filas sombreadas desaparecerían de las tablas. Las he dejado para demostrar que eliminamos redundancia si aplicamos la 5NF. Para reconstruir la información original haríamos una consulta con las tres tablas usando los campos de la clave original (¡no hay otros!). Por supuesto, hay que considerar si la eliminación de la redundancia compensa las búsquedas que originará la división de la tabla original en tres...

El otro motivo para usar 5NF es que una tabla tiene demasiados campos. Cuando dividimos una tabla de grado elevado no estamos eliminando redundancias. Se trata de optimizar el tiempo de acceso de una base de datos real. Supongamos el caso de una tabla que ya está en 4NF con datos de una persona y docenas de campos.

<u>DNI</u>	C1	C2	C3	C4	C5	...	C87	C88	C89	C90
...

Por simplificar, suponemos que la única clave candidata es DNI. Sencillamente dividimos la tabla en varias, usando el campo DNI como clave de todas las demás. En un caso práctico agruparíamos los campos por datos laborales, personales, etc.

<u>DNI</u>	C1	C2	C3	C4	C5
...

<u>DNI</u>	C6	C7	C8	C9	C10	...	C50
...

<u>DNI</u>	C51	C52	C53	C54	C55	...	C90
...

Obviamente, si hacemos una consulta uniendo las tres tablas por su clave principal obtendremos la tabla original. La relación está en 5NF.

En Access (base de datos muy poco eficiente) he usado tablas con más de 200.000 registros (y grandes) sin problemas. Y he diseñado tablas con más de cien campos sin notar pérdidas de rendimiento. Como ya he dicho, de momento no vas a usar 5NF.

6

Access

El objetivo de este capítulo es estudiar con detalle los diferentes componentes de Access. Ampliaremos lo explicado en el capítulo 2, “Bases de datos relacionales” y profundizaremos en conceptos más avanzados.

La pantalla principal de Access nos proporciona acceso a una serie de elementos:

- **Tablas.** El lugar dónde almacenaremos los datos: Diseñaremos campos y asignaremos las claves
- **Consultas.** Son la manera que tienes, sin programar, de relacionar esos datos: Autores y libros, clientes y productos comprados, socios de la biblioteca con libros sin devolver... Información de varias tablas. Veremos diferentes tipos de consultas, criterios, campos calculados, etc.
- **Formularios** Ni tablas ni consultas están pensadas para que las manejen los usuarios, es decir, no están hechas para el uso diario de la base de datos. Imagínate nuestro pequeño ejemplo de Códigos de Autores con miles de libros y cientos de autores.

Los formularios están pensados para hacer **pantallas**, con botones, barras de desplazamiento y diversos controles que por ejemplo, permitan al usuario manejar la base de datos sin tener que usar los códigos directamente. En vez de escribir un tres para indicar que un libro lo ha escrito Paul Auster, el usuario elegiría al autor de una lista desplegable.

- Los **informes** son **listados** que saldrán por impresora. Las consultas no paginan, no pueden hacer encabezados para cada autor, etc. El usuario que maneje nuestra base de datos no tiene por qué saber qué consultas existen. Simplemente, pedirá que se imprima cierto informe.
- **Macros.** Automatización tareas sencillas. No se verán en este manual.
- **Módulos.** Programación en Visual Basic. Al igual que las macros, no las estudiaremos de momento; quizás en el siguiente manual...
- También veremos las **Relaciones**. Cómo plasmar en Access el diseño de la base de datos.

Si para tablas y consultas usábamos siempre la opción de Diseño, para formularios e informes usaremos siempre los **Asistentes**. El modo habitual de trabajo es usar el asistente para que Access complete la pantalla o el listado y después pasar a la parte de diseño para cambiar lo que no nos guste (el color, el tipo de letra, los campos que se ven en pantalla...). Una vez que los asistentes han acabado su trabajo, lo habitual es pasar a la opción de “diseño” para modificar o completar lo que han hecho.

6.1 Tablas

Ya vimos en capítulos anteriores la creación de tablas (Apartado 2.1.2, “Cómo se hace en Access”). Siempre las escribiremos usando la opción de diseño, es decir, creándolas desde cero. No voy a repetir lo explicado. Lo que sí voy a hacer es mostrar en qué consisten las diferentes propiedades de los campos y cómo se crean claves.

6.1.1 Propiedades de los campos

Siempre que pinches sobre el nombre de un campo, en la parte inferior de la pantalla aparecerán sus propiedades. Por ejemplo, paso a diseño en la tabla libros y pulso sobre el campo “N1 de Páginas”:

Propiedad	Valor
Tamaño del campo	Entero largo
Formato	Automático
Lugares decimales	Automático
Máscara de entrada	
Título	
Valor predeterminado	0
Regla de validación	
Texto de validación	
Requerido	No
Indexado	No

- Tamaño del campo es el número de bytes que ocupa. Lo vimos anteriormente.
- Formato. El aspecto con el que aparecerá en pantalla. El formato NUNCA modificará el contenido, sólo la pinta. Hay una serie de formatos que pueden aplicarse, aunque también puedes fabricártelos usando los códigos adecuados. Usa la ayuda de Access (ahorremos papel, quedan pocos árboles) para ver dichos códigos. Sin embargo, uno típico para dinero sería **###0,00 “€”**, ó **###0,00 “€”**; **“Negativo. Error”**; **“Cero €”** para que positivos, negativos y ceros salieran diferentes. También hay códigos para textos: **>** haría que los textos salieran en mayúsculas, **<** en minúsculas... ¡Eh! Léete la ayuda.
- Lugares decimales. Pues eso. La pena es que hace lo que quiere. Prueba.
- Máscara de entrada. Dice cómo va a escribirse la información. Este sí que cambia los datos. Los transforma para que se ajusten a la máscara. Los códigos están en la ayuda, pero por ejemplo **000**, significa tres números obligatorios, **>LL** dos letras (en mayúsculas) obligatorias, etc
- Título. El texto que aparecerá por defecto en consultas, formularios e informes en vez del nombre del campo.
- Valor predeterminado. Si no escribes nada el valor que tomará el campo. Tiene la manía de poner cero por defecto en todos los campos numéricos.
- Regla de Validación. Una regla que el valor tendrá que cumplir para que sea aceptado. Por ejemplo, para números de páginas será fácil evitar errores de introducción de datos poniendo **>10 y <10000**.
- Texto de Validación. El texto que sale al incumplirse la regla de validación. No pongas “Error”. Describe por qué está mal: **“El número de páginas debe ser mayor que 10 y menor que 10.000”**
- Requerido. ¿Es obligatorio escribir algo en ese campo?. Muy útil.
- Indexado. Ordena por ese campo. En Access no funciona (supongo que estará por compatibilidad con otras bases de datos). Da igual, porque nosotros lo haremos con claves¹⁸.

¹⁸ En teoría no es lo mismo indexar un campo que ponerle una clave. Se supone que hay diferencias internas de funcionamiento. Da igual. He probado con tablas muy grandes y el indexado no hacía nada.

6.1.2 Claves

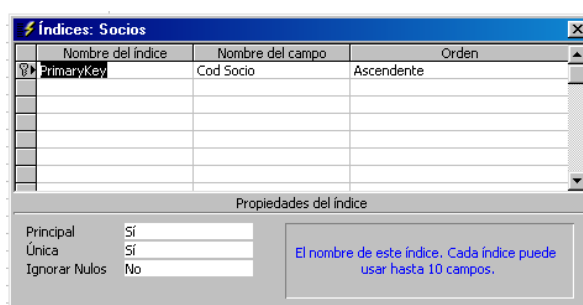
Para asignar la famosa **clave principal** a uno o varios campos de las tablas hay un botón diseñado específicamente para eso en la barra de herramientas:



Es el que se usará habitualmente. Pero si queremos aplicar más claves a una base de dato, usaremos el botón que tiene a su lado:



Supongamos que queremos hacer una clave para socios por apellido primero, apellido segundo y nombre. Sacamos a menudo listados ordenados de ese modo y queremos que se hagan un poco más rápidos. La consulta calcula los datos cada vez que es ejecutada. Si los nombres ya están ordenados, debería sacar los resultados más rápidamente. Al pulsar en el botón aparece el siguiente cuadro de diálogo (supongo que la clave principal ya estaba creada con el botón de la llave):

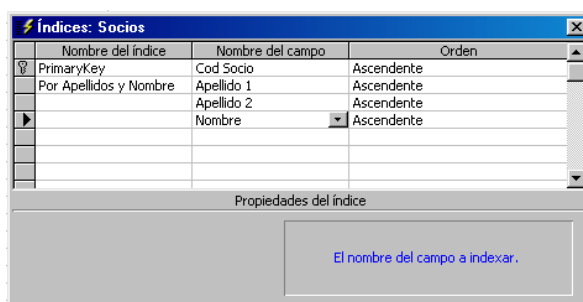


Al pinchar sobre el nombre de una clave, aparecen sus propiedades en la parte inferior de la ventana.

- Principal. Indica que es la clave principal de la tabla. Los campos aparecerán en negrita, dejará hacer relaciones (ya lo veremos), y obligatoriamente debe ser única y no ignorar nulos.
- Única. Si permite o no repeticiones
- Ignorar Nulos. Aquí diremos si también ordenará los valores en blanco. Si decimos que sí los ignore, no los tendrá en cuenta a la hora de hacer las ordenaciones.

El nombre de la clave puede ser cualquiera, mientras no se repita en esa tabla. El nombre sólo sería importante si fuéramos a usar programación.

Estábamos creando una nueva clave para los nombres y apellidos de los socios. La clave no podrá ser única, ya que es posible que algún nombre se repita. Poco probable pero posible, sobre todo si de vez en cuando dejamos en blanco el segundo apellido. Sobre ignorar nulos, ya que muchas veces dejaremos en blanco los campos, diremos que no:



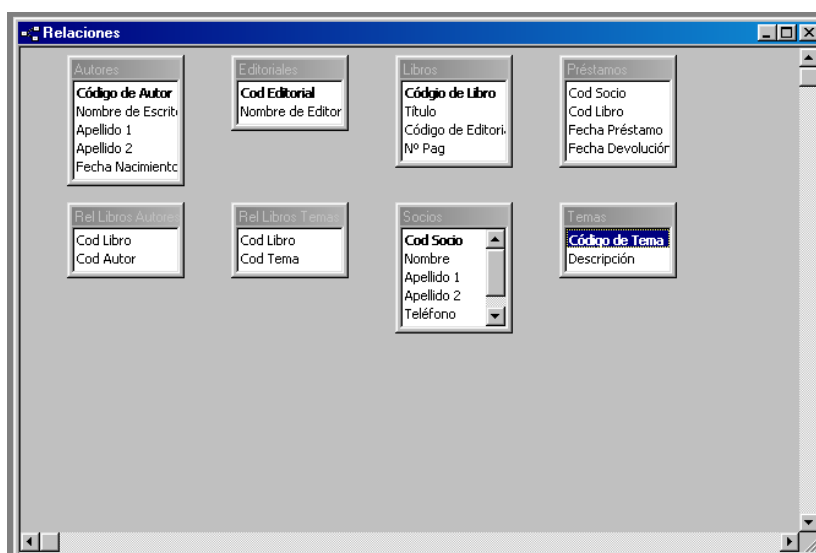
Access usará esa ordenación cuando le convenga. Cuidado; no la aplicará hasta que guardemos la tabla. Si en ese momento la clave falla porque hemos exigido que sea única pero hay datos repetidos, tendremos que salir sin salvar, modificar los datos duplicados y volver a crearla.

6.2 Relaciones

Una vez que hemos creado **todas** las tablas y hemos aplicado las claves principales a **todas** las partes uno de las relaciones, podemos pasar a dibujar el diseño de la base de datos. El objetivo es decirle a Access qué tablas están relacionadas entre sí, de modo que aplique dichas relaciones de manera automática. El botón para pasar a la pantalla de relaciones es



La pantalla que aparecerá a continuación nos preguntará qué tablas queremos añadir a continuación. Por supuesto, todas. Si más adelante queremos añadir más tablas, el botón con el símbolo más amarillo nos permitirá hacerlo. Una vez añadidas las tablas el aspecto de la pantalla será el siguiente:



Lo que tenemos que hacer es pulsar sobre cualquier código y arrastrarlo al código de la tabla correspondiente. En ese momento aparecerá el siguiente cuadro de diálogo (por ejemplo voy a unir los códigos de libro de Libros y Préstamos):



Elegiremos **siempre** “**exigir integridad referencial**”. Esa frase tan rara es el quid de la cuestión. Exigir integridad referencial significa que en la parte N de la relación no podrán existir códigos que no existan en la parte uno: No podremos introducir en prestamos libros que no existan. En ese momento Access entenderá que entre las dos tablas hay una relación de uno a N y actuará en consecuencia.

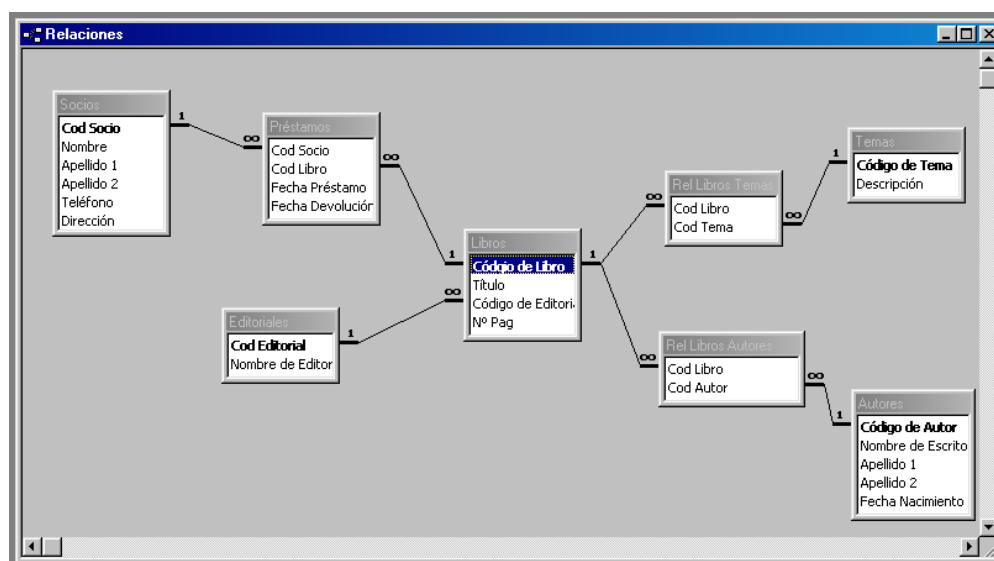
Al activar la integridad referencial, Access nos permite:

- Actualizar en cascada campos relacionados: Si por algún motivo decidimos cambiar el código de la parte uno de la relación, también cambiarán los códigos de la parte N, de modo que no haya ninguno que se quede “descolgado”
- Eliminar en cascada campos relacionados. Si borramos una fila de la parte uno, se borran todas las filas relacionadas de la parte N. Cuidado al usarlo.

Es posible que no podamos activar la integridad referencial por dos motivos:

- Los tipos de datos o los tamaños de los campos son distintos. Si los códigos de la parte uno y de la parte N están relacionados, es que van a guardar el mismo tipo de información. Deberían ser del mismo tipo de datos y tener el mismo tamaño. Access nos obliga a hacer así. Un error típico es definir sin querer uno de los códigos como texto.
- Los datos preexistentes incumplen la integridad referencial. Había datos mal escritos anteriormente, por ejemplo, un código de libro en Préstamos que no existe en Libros. Hasta que no corriamos ese dato, no podremos activar la integridad referencial.

Si no hay errores, y relacionamos todas las tablas, las arrastramos para que quede legible, etc., Tendremos el siguiente “mapa” de la base de datos¹⁹:



A partir de ahora, siempre que hagamos una consulta, un informe, etc., Access aplicará este diseño de base de datos.

Como expliqué con anterioridad, un informático no dirá “hemos establecido las relaciones entre las tablas”, sino “hemos creado las claves foráneas”.

6.3 Consultas

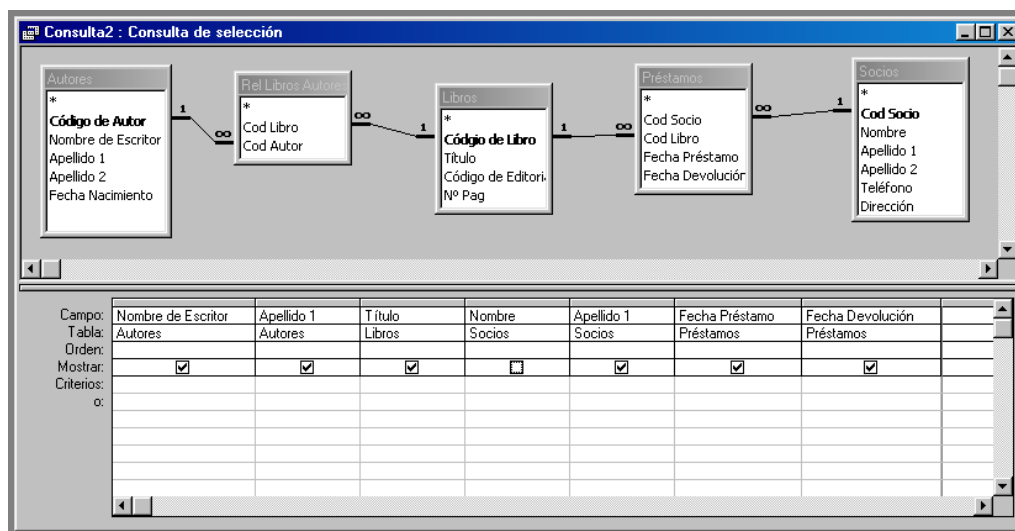
Con una consulta se pueden hacer muchas más cosas de las que hemos visto hasta ahora. La forma de crearlas es la misma: Siempre desde diseño, y escogiendo las tablas que necesitamos: El proceso será siempre:

1. Escoger las tablas donde están los datos que queremos ver.

¹⁹ Access usa el símbolo de infinito para indicar la parte N de la relación.

2. Escoger las tablas que tienen el camino entre las anteriores. Siempre tiene que haber una relación entre las tablas.
3. Hacer doble clic o arrastrar a la parte inferior los campos que nos interesen
4. Aplicar los criterios y ordenaciones que necesitemos.

Un ejemplo. Supongamos que quiero ver el autor, título del libro y nombre de socio de los libros prestados actualmente. Las tablas que contienen los datos son Libros, Autores, Socios y Préstamos (la fecha de devolución). Pero si nos fijamos en el diseño de la base de datos, la tabla Autores se queda descolgada: Necesitamos también la tabla Rel Libros Autores. Si hacemos todo esto y escogemos los campos que nos interesan:



Access ya conoce las relaciones entre las tablas; Por eso las ha dibujado automáticamente. El resultado de la consulta:

Nombre de Escritor	Autores.Apellido 1	Título	Socios.Apellido 1	Fecha Préstamo	Fecha Devolución
Miguel	De Cervantes	El Quijote	Perez	01/01/99	24/01/99
William	Gibson	Neuromante	García	05/06/01	17/06/01
William	Gibson	Neuromante	García	04/06/01	16/06/01
William	Gibson	Neuromante	Arregui	05/06/01	22/06/01
William	Gibson	Neuromante	García	10/01/01	
William	Gibson	Neuromante	Martínez de Antofiana	08/12/01	
Paul	Auster	El palacio de la Luna	Perez	05/06/01	08/06/01
Paul	Auster	El palacio de la Luna	Arregui	08/07/01	
Toti	Martínez	La Calle de la Judería	Fernandez	06/06/01	10/06/01
Paul	Auster	Leviatán	Perez	02/03/00	14/03/00
Paul	Auster	Leviatán	Fernandez	03/04/00	01/05/00
Paul	Auster	Leviatán	Martínez de Antofiana	09/02/01	
Paul	Auster	Leviatán	Fernandez	04/06/01	
Günter	Grass	El tambor de Hojalata	Martínez de Antofiana	04/03/00	06/03/00
Günter	Grass	El tambor de Hojalata	Arregui	01/02/99	16/02/99
Günter	Grass	El tambor de Hojalata	Fernandez	22/11/01	
Paul	Auster	La Música del Azar	Perez	03/12/01	
Kim	Robinson	Marte Rojo	Perez	02/03/99	14/03/99
Kim	Robinson	Marte Verde	García	01/01/99	04/02/99

Nos aparecen todos los libros prestados, y no solamente los no devueltos como queríamos. Para conseguir la información necesaria tenemos que aplicar **criterios**.

6.3.1 Criterios

Los criterios son **filtros** que aplicamos a las consultas para que sólo muestren cierta información. Si por ejemplo quiero ver sólo los libros de Paul Auster:

Campo:	Nombre de Escritor	Apellido 1	Ti
Tabla:	Autores	Autores	Lit
Orden:			
Mostrar:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Criterios:	"Paul"	"Auster"	
O:			

En la línea de Criterios, y debajo de los campos correspondientes, escribo la condición que dichos campos deben cumplir. En este caso que el nombre de Escritor sea igual a "Paul" y que el apellido sea igual a "Auster". Por supuesto, puedo poner el número de criterios que me interese. Para el ejemplo que estábamos haciendo, quiero los libros que no han sido devueltos, es decir, aquellos cuyo valor en el campo Fecha de devolución sea "ninguno en absoluto"; Nulo:

Préstamo	Fecha Devolución	
os	Préstamos	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
	Es Nulo	

Basta con escribir "nulo" o "null". Access suele completar las expresiones, poniendo comillas, escribiendo "Es Nulo", etc. Y si quisiera los libros de Paul Asuter no devueltos, sólo tendría que escribir los tres criterios a la vez.

Todas las líneas debajo de criterios también sirven para añadir condiciones. ¿En que se diferencia ponerlos en la misma línea o en líneas diferentes? En que cada línea se comporta como una pregunta **independiente**. Si escribo estos criterios:

Nombre	Apellido 1	Fecha Préstamo	Fecha Devolución
Socios	Socios	Préstamos	Préstamos
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
			Es Nulo
"Aitor"			

Tendré todos los libros no devueltos (independientemente del socio) y todos los libros usados por los socios que se llamen Aitor (independientemente de la fecha de devolución).

No sólo se puede preguntar por las cosas "iguales":

>	<i>Mayor que</i>
>=	<i>Mayor o igual</i>
<	<i>Menor que</i>
<=	<i>Menor o igual</i>
=	<i>Igual (lo entiende por defecto)</i>
<>	<i>Diferente</i>
Como	<i>Para textos; permite el uso de asteriscos.</i>

Hay más operadores y muchos ejemplos en la ayuda de Access. Consúltala.

También posee dos operadores muy útiles, el operador "Y" y el operador "O". Sirven para unir preguntas. Con el operador "Y" todas las preguntas deben ser ciertas para que el criterio se cumpla. Con el operador "O" basta con que sea cierta una de las preguntas para que el criterio se haga:


>=#1/4/2000# Y <#30/4/200 fechas de Abril
 ="Ana" O ="Pedro" O "Juan" Ana o Pedro o Juan

Con criterios, tiene sentido la casilla de **Mostrar**. Si está activa, el campo se ve. Si no lo está el campo permanece oculto. ¿En qué se diferencia de no ponerlo? La diferencia es que ese campo puede tener algún criterio activo. Si pido los libros no devueltos, aquellos cuya fecha de devolución vale Nulo, ¿Para

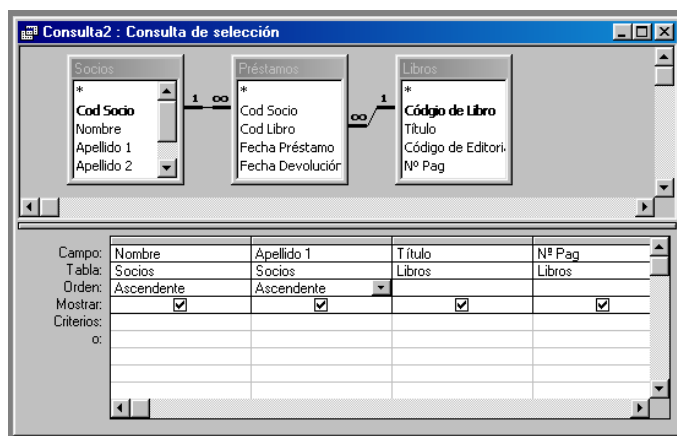
qué quiero ver la fecha de devolución? Ya sé que es Nulo. Por supuesto, no la puedo quitar, porque borraría el criterio y volverían a salir todos los libros. Lo lógico es ocultarla.

La casilla de **Orden** sirve para ordenar los resultados. Siempre actúa de izquierda a derecha. Si queremos que ordene primero por un campo, éste deberá ser el que esté situado más a la izquierda de los campos por los cuales quiero ordenar.

6.3.2 Totales

Hay muchas utilidades dentro de las consultas. Una de las más usadas son las consultas de totales, que se activan al pulsar el botón: 

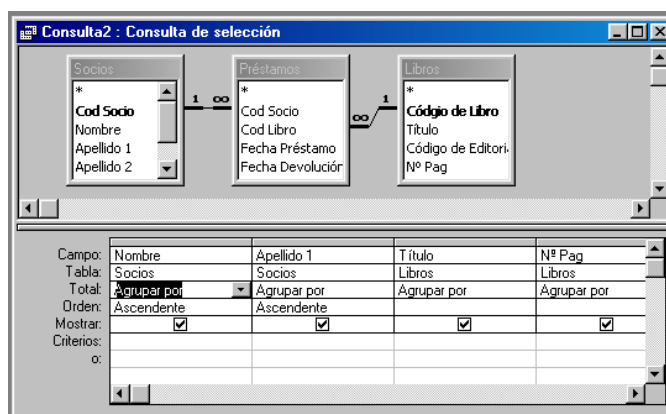
Sirven para **agrupar** los resultados de una consulta. Supongamos que quiero saber el número de páginas ha leído cada socio (o lo que me debe un cliente: todas las bases de datos son muy parecidas). Empiezo con la consulta "normal"; El número de páginas está en Libros, socios en socios y hay una tabla en medio, Préstamos.



El resultado:

Nombre	Apellido 1	Título	Nº Pag
Aitor	Martinez de Antofiana	Leviatán	266
Aitor	Martinez de Antofiana	El tambor de Hojalata	655
Aitor	Martinez de Antofiana	Neuromante	316
Ana	García	Neuromante	316
Ana	García	Marte Verde	692
Ana	García	Neuromante	316
Ana	García	Neuromante	316
Antonio	Perez	Marte Rojo	650
Antonio	Perez	Leviatán	266
Antonio	Perez	El Quijote	1200
Antonio	Perez	El palacio de la Luna	308
Antonio	Perez	La Música del Azar	251
Eva	Arregui	Neuromante	316
Eva	Arregui	El tambor de Hojalata	655
Eva	Arregui	El palacio de la Luna	308
Gaizka	Fernandez	Leviatán	266
Gaizka	Fernandez	El tambor de Hojalata	655
Gaizka	Fernandez	Leviatán	266
Gaizka	Fernandez	La Calle de la Judería	490

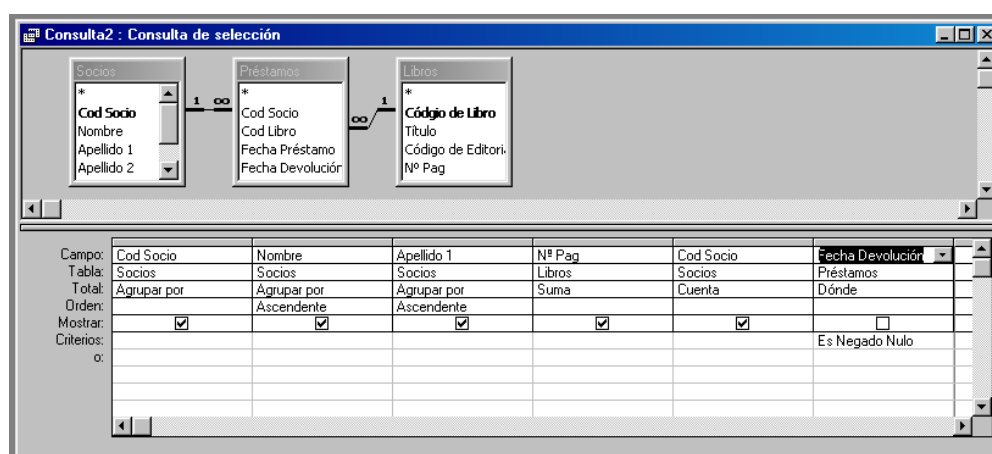
Pero lo que quiero es la suma. Lo que voy a pedirle a la consulta es que me saque **una fila** por cada socio, y que **sume** el número de páginas correspondientes a ese socio. En diseño activo el botón de Totales. Aparece una fila más en la cuadrícula inferior, la fila de **Total**:



Se pueden escoger tres acciones distintas:

- Si se quiere sacar **una fila por cada...** se escogerá **"Agrupar por"**. En nuestro caso quiero una fila por cada socio. Puedo agrupar por nombre de socio, pero no sería correcto: Es posible que haya dos socios que se llamen igual. Si hay dos socios que se llamen "Pedro" me saldría una sola fila para ambos. Una solución intermedia sería agrupar por Nombre y Apellidos. Estamos en las mismas. Sería muy raro, pero también pueden repetirse. La solución en estos casos es agrupar por Código de Socio. Eso es seguro que no se repite. Se suelen dejar los nombres y apellidos y agrupar también por ellos. No van a molestar (estoy agrupando por el código de esa tabla), y son más claros que un simple número.
- Tengo tres filas para el socio Aitor. Ahora me saldrá sólo una. ¿Qué hago con las 266, 355 y 316 páginas de los libros de esas tres filas? En este caso las sumo. Habrá campos con los que querrás **operar**. Las operaciones típicas son suma, cuenta (muy usada), promedio, mínimo, máximo...
- Por último quedan las **condiciones**. Los criterios normales se aplican **después** de realizado el agrupamiento, por lo que no serán validos si deseo contar por ejemplo sólo las páginas de los libros devueltos. Si se quieren hacer ese tipo de excepciones se debe usar la cláusula **dónde**. La condición expresada será lo primero que se haga.

El resto de campos de la consulta los debo quitar, o de lo contrario no saldrá lo que queremos. Si dejamos el título del libro tendré que hacer algo con él... y en este caso no me hace falta. El diseño de la consulta queda:



Lo primero que se realiza es la consulta normal. A continuación se aplica la condición de la columna "donde". Los libros cuya fecha de devolución sea nula (los no devueltos) son filtrados. Después se

realiza la agrupación y se opera con los campos agrupados. Lo último que se ejecutaría serían los criterios “normales”.

He añadido la operación contar. Es una operación un poco extraña. No importa sobre que campo la apliques, cuenta el número de filas de la consulta (siempre que el valor de dicho campo no sea nulo). Como en esta consulta la parte N es Préstamos, se supone que cuenta todos los libros prestados. Como hay un “donde” que filtra los libros cuya fecha de devolución sea diferente de nulo, cuenta libros devueltos. El resultado:

Cod Socio	Nombre	Apellido 1	SumaDeNº Pa	CuentaDeCod Socio
1	Aitor	Martinez de Antofiana	655	1
2	Ana	García	1324	3
1	Antonio	Perez	2424	4
5	Eva	Arregui	971	2
3	Gaizka	Fernandez	756	2

6.3.3 Campos calculados

En muchas ocasiones lo que nos interesa no es el dato en sí, sino una operación que realicemos sobre él. En nuestro caso, queremos saber el número de días que un libro ha sido prestado para imponer o no multa, etc. Otras veces un dato se saca fácilmente de otro, como el IVA que añadiremos a un precio. Todas esas operaciones las realizaremos con **campos calculados**.

Para añadir un campo calculado a una consulta basta con escribir en una columna libre el nombre que queramos darle a ese nuevo campo²⁰: por ejemplo, para sacar el número de días que el libro lleva prestado, suponiendo que hoy es 15/12/01:

Campo:	Nombre	Apellido 1	Título	Fecha Préstamo	Fecha Devolución	Días: #15/12/01#[Fecha Préstamo]
Tabla:	Socios	Socios	Libros	Préstamos	Préstamos	
Orden:						
Mostrar:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criterios:					Es Nulo	

En la pantalla sale:

²⁰ Usa shift+F2 o el botón derecho del ratón para que aparezca el Zoom...

Nombre	Apellido 1	Título	Fecha Préstamo	Fecha Devolución	Días
Eva	Arregui	El palacio de la Luna	08/07/01		160
Aitor	Martínez de Antuña	Leviatán	09/02/01		309
Ana	García	Neuromante	10/01/01		339
Gaizka	Fernández	El tambor de Hojalata	22/11/01		23
Antonio	Pérez	La Música del Azar	03/12/01		12
Gaizka	Fernández	Leviatán	04/06/01		194
Aitor	Martínez de Antuña	Neuromante	08/12/01		7

El campo calculado ha quedado de la siguiente manera:

Días: #15/12/01# - [Fecha Préstamo]

Es muy fácil de escribir, si se siguen una serie de reglas de sintaxis:

<i>Siempre se escribe Nombre del Campo: Operaciones a realizar</i>	<i>Días : el resto de operaciones</i>
<i>Las fechas van entre almohadillas</i>	<i>#15/12/01#</i>
<i>Los textos van entre comillas</i>	<i>"Un texto"</i>
<i>Los nombres de campos siempre entre corchetes</i>	<i>[Fecha Préstamo]</i>
<i>Las operaciones típicas que podemos realizar son:</i> <ul style="list-style-type: none"> • Suma + • Resta - • Multiplicación * • División / • Exponente ^ • Resto mod • Pegar Texto & 	<i>[Precio] + 1200</i> <i>#15/12/10# - [Fecha Prestamo]</i> <i>[Precio] * 0,16</i> <i>[Precio]/166.386</i> <i>[Longitud] ^ 3</i> <i>[Num Asiento] mod 4</i> <i>[Nombre Cliente] & " " & [Apellido 1]</i>
<i>Las funciones que usemos siempre llevarán paréntesis</i>	<i>Raíz2([Area])</i>

Access a veces completa la sintaxis, pero otras no, por lo que es una buena idea escribirlo siempre bien.

Las **funciones** son operaciones "especiales". Si queremos sumar o multiplicar tenemos un símbolo para ello, pero ¿Si queremos hacer una raíz cuadrada o sacar el año de una fecha? En vez de inventarse cientos de símbolos para esas nuevas operaciones (además, no hay bastantes en el teclado) lo que se ha hecho es darles un nombre: Por ejemplo si quieres sacar el año, mes y día por separado de una fecha:

Día Prestamo: Día([Fecha Préstamo])	Mes Prestamo: Mes([Fecha Préstamo])	Año Prestamo: Año([Fecha Préstamo])
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Fíjate en la sintaxis. Siempre es el nombre de la función y entre los paréntesis e la misma los datos que la función necesita para trabajar. En este caso los **parámetros** en cada caso eran uno sólo, el nombre del campo "Fecha Devolución"; va entre corchetes por ser el nombre de un campo. El resultado en pantalla:

Nombre	Apellido 1	Título	Fecha Préstamo	Día Prestamo	Mes Prestamo	Año Prestamo
Antonio	Perez	El Quijote	01/01/99	1	1	1999
Antonio	Perez	Marte Rojo	02/03/99	2	3	1999
Ana	García	Marte Verde	01/01/99	1	1	1999
Antonio	Perez	Leviatán	02/03/00	2	3	2000
Aitor	Martinez de Antofiana	El tambor de Hojalata	04/03/00	4	3	2000
Eva	Arregui	El tambor de Hojalata	01/02/99	1	2	1999
Gaizka	Fernandez	Leviatán	03/04/00	3	4	2000
Ana	García	Neuromante	05/06/01	5	6	2001
Antonio	Perez	El palacio de la Luna	05/06/01	5	6	2001
Gaizka	Fernandez	La Calle de la Judería	06/06/01	6	6	2001
Ana	García	Neuromante	04/06/01	4	6	2001
Eva	Arregui	Neuromante	05/06/01	5	6	2001
Eva	Arregui	El palacio de la Luna	08/07/01	8	7	2001
Aitor	Martinez de Antofiana	Leviatán	09/02/01	9	2	2001
Ana	García	Neuromante	10/01/01	10	1	2001
Gaizka	Fernandez	El tambor de Hojalata	22/11/01	22	11	2001

Hay cientos de funciones distintas. Léete la ayuda de Access. Funciones habituales:

<i>Año(fecha), Mes (fecha), Día(fecha)</i>	<i>Año, Mes y Día de una fecha</i>
<i>SerieFecha(Año; Mes; Día)</i>	<i>Crea una fecha a partir de números normales. Fíjate que esta función pide tres parámetros. Para separar unos de otros se usa el punto y coma.</i>
<i>Raíz2(valor)</i>	<i>La raíz cuadrada de un número</i>
<i>Silnm(pregunta; Valor1; Valor2)</i>	<i>Una función MUY ÚTIL. Si la pregunta se cumple, vale lo contenido en el segundo parámetro. Si no, devuelve el tercer parámetro. Ejemplos de la función "Sí Inmediato" (el nombre es s-i-i-n-m): Silnm([Altura]>180;"ALTO";"BAJO") Silnm([Precio]>400;[Precio]*0.9;[Precio]) Silnm([Valor]<0;"*****";"") Silnm(Año([Fecha Préstamo])<1999;"Se lo ha quedado en casa";"")</i>
<i>Ahora(),Fecha()</i>	<i>La fecha y hora actuales ó solo la fecha. No tienen parámetros, pero son funciones...</i>

6.4 Formularios

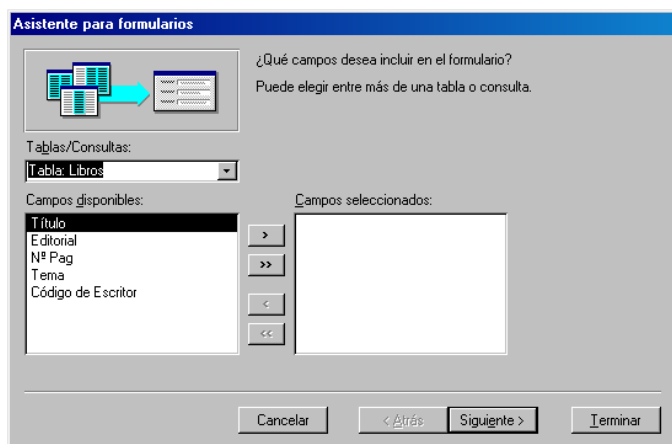
Los formularios e informes no son parte propiamente dicha de una base de datos relacional, pero sí de las bases de datos de Access. Las tablas y consultas están pensadas para el "diseñador" de la base de datos (es decir, para ti), mientras que los formularios e informes serán creados (también por ti) para la persona que vaya a usar la base de datos y que es posible que no sepa nada de informática. La idea es "vestir" las tablas y consultas para que su manejo sea cómodo.

Generalmente las pantallas (los formularios) se usarán como menús o para introducir datos. Lo habitual por tanto es que un formulario esté conectado a una tabla, aunque por supuesto también se puede hacer un formulario de una consulta²¹.

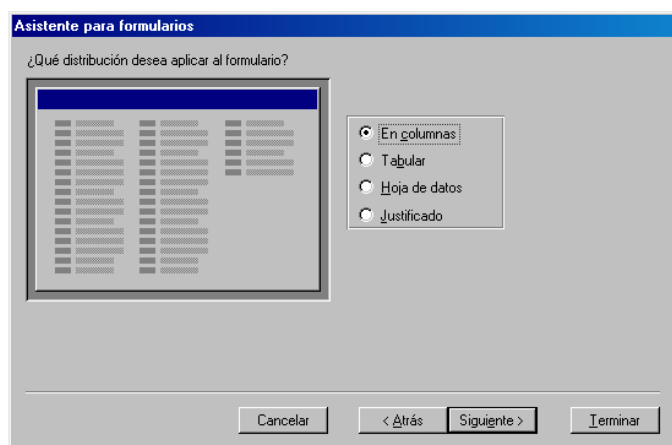
6.4.1 Crear un formulario con el asistente.

Para crear un formulario, tenemos que pasar a la sección de formularios y pulsar la opción del asistente:

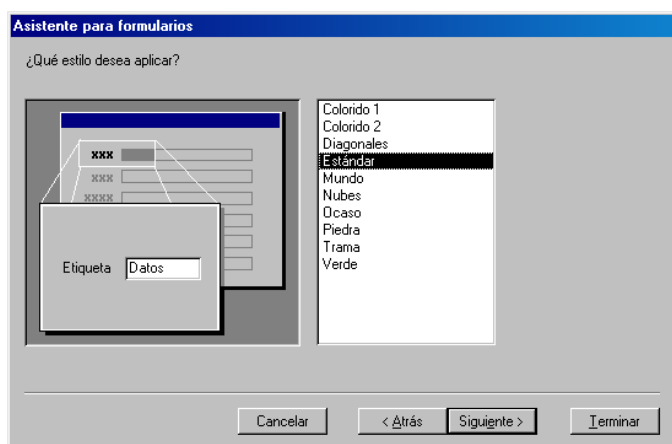
²¹ De hecho, ningún elemento de Access distingue entre tablas y consultas. Para todos los componentes de Access tablas y consultas son lo mismo: Un conjunto de filas y columnas. Todo lo que puedas hacer con una, se lo puedes hacer a la otra.



Lo primero que nos pregunta es sobre qué tabla o consulta queremos hacer el formulario y qué campos queremos ver. En “Campos disponibles” tenemos los campos que forman la tabla o consulta. En “Campos seleccionados”, los que hemos escogido. Yo voy a hacer una pantalla para ver la tabla libros y poder modificarla, así que escogeré todos los campos. Los botones con flechas entre los dos recuadros sirven para hacerlo de diversas formas. Una vez hecho esto, pulsamos el botón **siguiente**:



Ahora nos pregunta si queremos ver un registro por pantalla, **columnas**, o bien todos los registros uno debajo de otro, **tabular**. Los otros carecen de interés. Haz pruebas si quieres ver cómo quedan. Ya que los registros de libros tienen pocos campos, elegiré Tabular. Si hubiera muchos campos, por ejemplo la típica tabla de clientes, no podría verlos todos en la misma línea, por lo que escogería En columnas. Pulsamos el botón de Siguiente.



Ahora nos pregunta el aspecto general de la pantalla. Hazme, caso, escoge estándar. Cuantas menos cosas raras te ponga, menos tendrás que modificar en Diseño (además, el resto son horribles).
Siguiente:

Por último nos pregunta el nombre que asignaremos a esta pantalla. Ya que sirve para ver y modificar la tabla Libros, lo voy a llamar... Libros. Ya hemos acabado, por lo pulsamos el botón Terminar. El resultado:

Título	Editorial	Nº Pag	Tema	Código de Escritor
El Quijotes	Alfaguara	1200	Literatura	1
Neuromante	Minotauro	316	Literatura - CF	2
El palacio de la Luna	Anagrama	308	Literatura	3
La Calle de la Judería	Abra	490	Literatura - HIST	4
Leviatán	Anagrama	266	Literatura	3
El tambor de Hojalata	Alfaguara	655	Literatura	5
La Música del Azar	Anagrama	251	Literatura	3
Pórtico	Ed. B	363	Literatura - CF	6
Java 2	RaMa	650	Informática	7
Programación en Window	Mc Graw Hill	1100	Informática	8
Historia Universal	Anaya	900	Historia	9
Marte Rojo	Minotauro	650	Literatura -CF	10
Marte Verde	Minotauro	692	Literatura -CF	10
Marte Azul	Minotauro	724	Literatura -CF	10

Pues vaya, para eso se pueden usar las tablas directamente. Es cierto; Pero cuando veamos como se modifica el diseño, nos quedará algo como esto (fíjate en el código del escritor):



Título	Editorial	Nº Pag	Tema	Escritor
El Quijotes	Alfaguara	1200	Literatura	Cervantes
Neuromante	Minotauro	316	Literatura - CF	William Gibson
El palacio de la Luna	Anagrama	308	Literatura	Paul Auster
La Calle de la Judería	Abra	490	Literatura - HIST	Toti Martinez
Leviatán	Anagrama	266	Literatura	Paul Auster
El tambor de Hojalata	Alfaguara	655	Literatura	Günter Grass
La Música del Azar	Anagrama	251	Literatura	Paul Auster
Pórtico	Ed. B	363	Literatura - CF	Frederik Pohl
Java 2	RaMa	650	Informática	Agustín Froute
Programación en Win	Mc Graw Hill	1100	Informática	Petzold - Paul Y
Historia Universal	Anaya	900	Historia	Varios
Marte Rojo	Minotauro	650	Literatura -CF	Kim S. Robinson

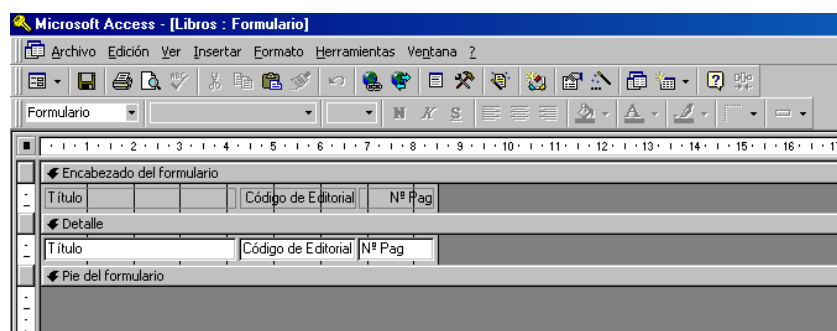
6.4.2 Diseño y manejo de controles

Es difícil que el asistente deje el formulario como queremos. Siempre tendremos que ir a diseño para modificarlo.

Veremos que por lo general, la pantalla creada por el asistente se divide en tres partes: encabezado, detalle y pie.

El **pie** y el **encabezado** se pueden poner o quitar a través del menú Ver. Aparecen respectivamente al principio y al final de la pantalla. El **detalle** es la parte del formulario donde se ven los datos, y ocupa toda la pantalla que le hayan dejado el pie y el encabezado. Si es un formulario Tabular, se repetirá una y otra vez. Si es en Columnas, aparece sólo un detalle por pantalla.

Los datos y textos del formulario aparecen en los diferentes **controles** que lo forman. Hay muchos tipos de controles; Los más habituales son botones, **etiquetas**, que sirven para mostrar textos simples y **cuadros de texto**, que de algún modo están enganchados con los campos de las tablas o consultas para mostrarnos su contenido. Por supuesto, si modificamos el contenido del cuadro de texto estamos modificando el contenido del campo de la tabla o consulta:



He hecho con el asistente un formulario para la tabla de Libros. NO he añadido el campo código de libro. ¿Para qué? Es un Autonumérico que el ordenador rellenará automáticamente cada vez que haga un libro nuevo. Ni siquiera puedo modificarlo. Y enseguida vamos a comprobar que no he hecho falta saber el código del libro para poder prestarlo o indicar quién lo ha escrito.

Las etiquetas las suele colocar en el encabezado si es un formulario tabular. En este caso, tienen fondo transparente. Los cuadros de texto son los controles de fondo blanco que ha colocado en el detalle. Para

cambiar el fondo, el tipo de letra, el color, etc. basta con pinchar sobre el control y elegir la opción adecuada de la barra de herramientas. Es idéntico que en Word, por ejemplo.

Para seleccionar varios controles a la vez puedes pulsar en las reglas horizontal y vertical, ir pinchando los controles con la tecla “mayúsculas” (“shift”) pulsada o pinchar con el ratón en un sitio en el que no haya nada (por ejemplo la zona gris oscuro) y empezar a arrastrar. Todo lo que toques con el rectángulo que irá apareciendo quedará seleccionado.



Si un control está seleccionado aparecerá con el siguiente aspecto


Los cuadrados que aparecen alrededor del control sirven para estirarlo en una u otra dirección. El cuadrado de la esquina superior izquierda es especial. Hay veces que un control está vinculado a otro, o sencillamente tienes varios seleccionados. Si quieres mover sólo uno de ellos, pincha sobre el ese cuadrado y arrastra.

Si te colocas dentro del control, el ratón cambiará de forma y aparecerá una mano; de esa manera puedes arrastrarlo. Pero la forma más cómoda de mover un control es arrastrándolo directamente **sin** seleccionarlo. No pinches sobre él, sencillamente arrástralo.

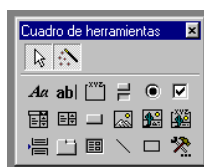
Para escribir algo dentro de un control, basta con seleccionarlo y pinchar dentro.

Por último, el menú formato tiene una serie de opciones **muy útiles** para manejar **varios controles** a la vez. Son: Alinear, Tamaño, Espaciado Horizontal y Espaciado Vertical. Aprender a manejarlas con soltura te ahorrará mucho tiempo.

6.4.3 Añadiendo controles

En ocasiones tendrás que añadir nuevos cuadros de texto, etiquetas u otros controles. El botón para que aparezca la barra de herramientas de los controles se llama “Herramientas”: 

Al pulsarla aparecerá o desaparecerá la siguiente barra de herramientas:



Los dos primeros botones no son controles. El primer sirve para quitar la selección, si has elegido un control y no lo quieres poner. El segundo, la varita mágica, sirve para activar o desactivar los asistentes. Algunos controles tienen asistentes propios para facilitar la tarea.

El tercer y cuarto botón son los más habituales. Etiquetas y Cuadros de Texto. Generalmente, cuando colocas un cuadro de texto, Access también crea una etiqueta.

Los cuatro siguientes sirven para listas de opciones y campos del tipo “Sí/No”

A continuación están el Cuadro Combinado (una lista desplegable) y el Cuadro de Lista. Son muy útiles, y los veremos en un apartado propio.

El Botón de Comando sirve para crear botones. Es muy fácil de usar, ya que tiene asistente propio. Basta con elegir el tipo operación que deseamos que ejecute el botón.

Los tres siguientes se usan para insertar imágenes u objetos OLE (cualquier cosa que puedas copiar y pegar) en el formulario.

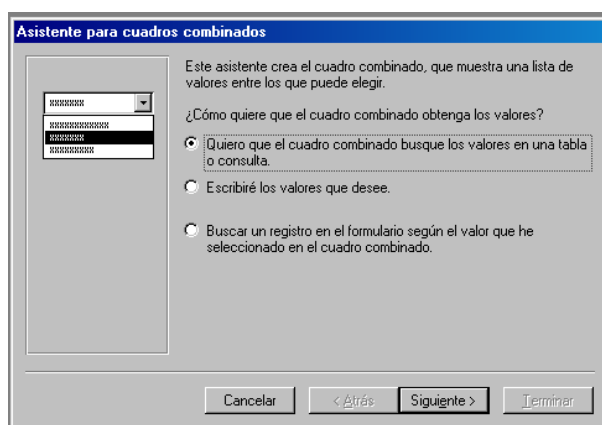
El siguiente es salto de página. Es muy usado en informes. Después tenemos un control para crear fichas y el control para crear subformularios o subinformes. Lo veremos después.

Los últimos sirven para dibujar líneas o rectángulos. El botón de llave y el martillo se usa para añadir controles especiales: Cualquier ActiveX que tengas en tu ordenador.

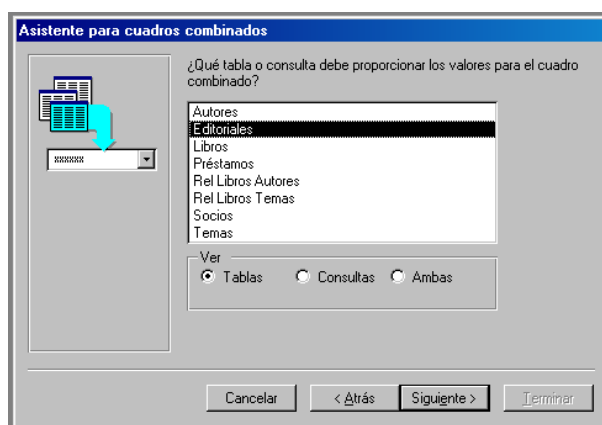
6.4.4 Cuadros combinados y cuadros de Lista

Sirven para evitar el uso de códigos en los formularios. Ya he comentado que los códigos autonuméricos no son necesarios: El ordenador los rellena automáticamente. ¿Pero qué hacemos con el código de editorial de Libros?

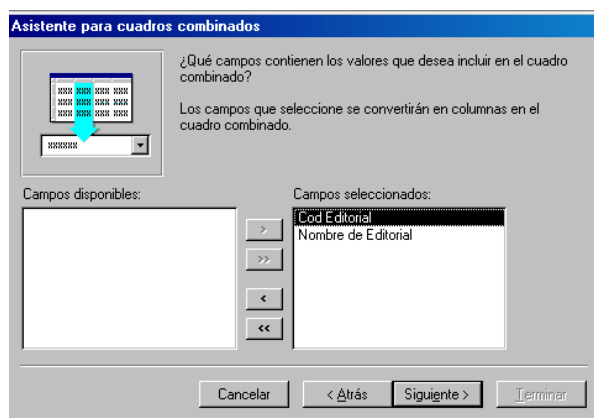
Los cuadros combinados (o los cuadros de lista, hacen exactamente lo mismo) son capaces de en este caso enseñarnos el contenido de la tabla editoriales, dejarnos elegir una de ellas y copiar sin que nos demos cuenta el código de editorial elegido al campo Código de Editorial. Si colocamos un cuadro combinado en el detalle (y está activada la varita mágica) aparecerá el asistente:



Lo primero que nos pregunta es dónde están los datos que queremos ver. Pues están en la tabla...

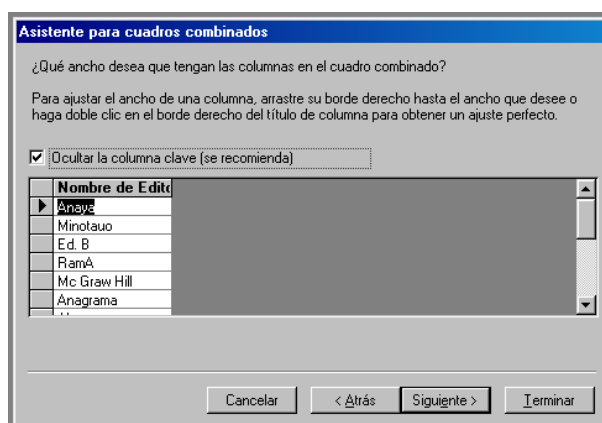


Editoriales. A continuación nos preguntará que campos queremos usar de dicha tabla:

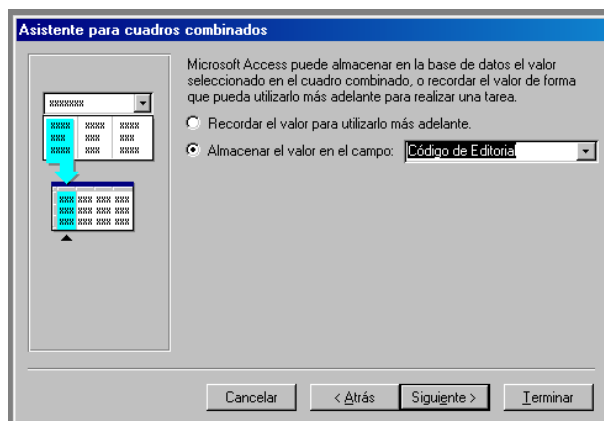


Queremos **los dos**. El texto, porque es lo que nos va a indicar qué estamos eligiendo. Y el código porque es lo que realmente vamos a usar. Todo esto lo estamos haciendo porque no queremos escribir códigos; Queremos que el ordenador escriba el código de manera automática cuando nosotros escojamos algún nombre; Evidentemente, el código es necesario. Dicho de otra manera: Hagamos lo que hagamos, la tabla de Libros necesita el código de editorial. Este control puede hacer maravillas, pero necesita ese código.

El paso siguiente es de presentación:

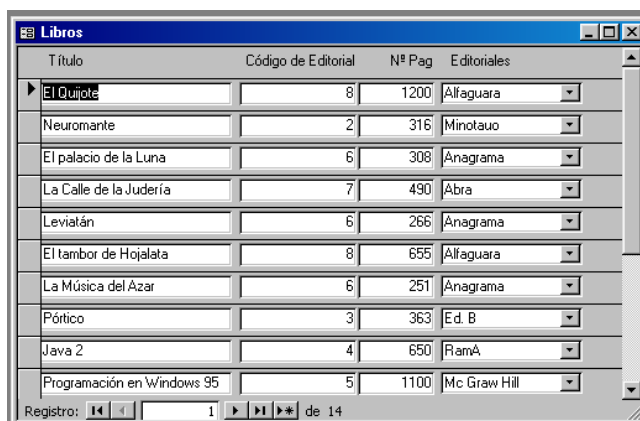


Como sabe que entre Libros y Editoriales hay una relación de uno a N, el ordenador se ha dado cuenta de lo que queremos hacer (ya era hora). Por eso ha ocultado el código de manera automática. Aunque esté ahí, nosotros no lo veremos. Que quede bien claro que cuando escojamos por ejemplo "Minotauro" estaremos cogiendo un "2" (su código). Si las relaciones no estuvieran hechas, el ordenador nos hubiera presentado un cuadro de diálogo adicional, en el que nosotros le diríamos cuál de los dos campos queríamos usar. La respuesta, por supuesto, hubiera sido el código.



Bien. Ya sabemos que cuando escojamos una editorial, estaremos cogiendo por ejemplo un “2” para Minotauro o un “1” para Anaya. ¿Para qué queremos ese número? Para escribirlo en el campo “Código de Editorial”. Es aquí donde enganchamos este control con el campo de la tabla.

Y Ya está. El último paso simplemente es ponerle un nombre a la etiqueta que acompaña a este control (da igual, la vamos a borrar). El formulario tendrá este aspecto:



El cuadro de texto Código de Editorial es redundante y lo eliminaremos.

6.4.5 Subformularios

En otras ocasiones lo que necesitamos es ver **varias tablas** a la vez. No me refiero a una consulta, sino a ver por ejemplo los datos del socio y por separado sus datos sobre préstamos. La solución es sencilla: dos formularios, uno para cada tabla, pero vistos a la vez en la misma pantalla: El formulario de la parte N de la relación (préstamos) dentro del formulario de la parte uno (socios).

Para eso necesitamos los dos formularios:

Socios

SOCIOS

Cod Socio: 1

Nombre: Antonio

Apellido 1: Perez

Apellido 2: Robles

Teléfono: 12345679

Dirección: Avenida Judizmendi S/N

Registro: 1 de 5

De paso, le he puesto una par de botones. He dejado código de socio, a pesar de ser Autonumérico para que se vea mejor lo que vamos a hacer. Además, en muchas bibliotecas el código del socio es un dato que sí utilizan las personas. El formulario de préstamos:

Préstamos

Cod Socio	Libro	Fecha Préstamo	Fecha Devolución
1	El Quijote	01/01/99	24/01/99
2	Neuromante	05/06/01	17/06/01
2	Neuromante	04/06/01	16/06/01
5	Neuromante	05/06/01	22/06/01
2	Neuromante	10/01/01	
4	Neuromante	08/12/01	
1	El palacio de la Luna	05/06/01	08/06/01
5	El palacio de la Luna	08/07/01	
3	La Calle de la Judería	06/06/01	10/06/01
1	Leviatán	02/03/00	14/03/00
3	Leviatán	03/04/00	01/05/00
4	Leviatán	09/02/01	
3	Leviatán	04/06/01	
4	El tambor de Hojalata	04/03/00	06/03/00
5	El tambor de Hojalata	01/02/99	16/02/99
3	El tambor de Hojalata	22/11/01	
1	La Música del Azar	03/12/01	
1	Marte Rojo	02/03/99	14/03/99

Registro: 1 de 19

Y ahora sólo queda que este formulario aparezca dentro del de socios: En diseño de socios usamos el control Subformularios/subinformes:

Tal vez se dispare algún asistente. En tal caso, basta con elegir el formulario que queremos ver hay dentro (el de préstamos) y dar a Terminar. Si no es así, lo único que hay que hacer es pinchar dentro del control y escribir "Préstamos". El resultado:

Cod Socio	Libro	Fecha Préstamo	Fecha Devolución
2	Marte Verde	01/01/99	04/02/99
2	Neuromante	05/06/01	17/06/01
2	Neuromante	04/06/01	16/06/01
2	Neuromante	10/01/01	

¿Por qué saca sólo los préstamos de cada socio? Porque sabe que están **relacionados**. En este caso, el campo "Cod Socio" de Préstamos es redundante y se podría eliminar.

6.4.6 Campos calculados

También se pueden añadir campos calculados a formularios e informes. La diferencia con las consultas es que aquí se escriben dentro de los cuadros de texto²², y que además no se pone “nombre_que_sea: operación”. Sencillamente, se usa el signo igual:

Nuevo Campo	= [Nombre] & " " & [Apellido 1] & " " & [Apellido 2]
-------------	--

Nuevo Campo	= [Precio] * 0,16
-------------	-------------------

Nuevo Campo	= Fecha() - [fecha Préstamo]
-------------	------------------------------

Aquí he cometido un error. El campo que contiene el nombre del socio se llama “Nombre”. Pues bien, no funcionará. En pantalla, en vez de salir el nombre del socio, me mostrará ¡el nombre del formulario! Hay una serie de nombres que están reservados para su uso dentro de formularios e informes. Los más comunes son:

- [Nombre]: El nombre del formulario o informe
- [Página]: El número de página actual
- [Páginas]: El número de páginas totales

¿Cómo puedo usar el nombre del socio para campos calculados? No puedo. Moraleja: Nunca llames a un campo de una tabla “nombre”, “página” o “páginas”. Bueno, puedes hacer una consulta de la tabla, cambiar ahí el nombre del campo haciendo un campo calculado (*nombre socio:[Nombre]*) y creando el formulario a partir de dicha consulta, pero en fin...

Hay algo que sólo se puede hacer en un formulario o informe: **Campos calculados de totales**. Consiste en ver a la vez el detalle (por ejemplo el número de páginas) y los totales (cuántas páginas hay en total, cuántas ha leído un socio...).

Por ejemplo, voy a aprovechar el formulario y subformulario²³ anterior, para saber cuántos días acumulados llevan los socios con libros prestados. Modifico el subformulario para que sólo me muestre los libros no devueltos (sencillamente, hago una consulta y baso el subformulario en dicha consulta):

Cod Socio	Libro	Fecha Préstamo	Fecha Devolución
8	El palacio de la Luna	08/07/01	
4	Leviatán	09/02/01	
2	Neuromante	10/01/01	
3	El tambor de Hojalata	22/11/01	
1	La Música del Azar	03/12/01	
3	Leviatán	04/06/01	
4	Neuromante	08/12/01	

Quito el campo fecha de devolución (ya sé que siempre es nulo) y añado un campo calculado “normal”:

10	Días
10	= Fecha() - [Fecha Préstamo]

²² NO escribas campos calculados en los controles que ha creado el asistente; Crea tus propios cuadros de texto. Cuando el asistente crea los cuadros de texto, le asigna un nombre idéntico al del campo de la tabla con la que está asociado: Si muestra “Precio”, el cuadro de texto se llama “Precio”. Pues bien, Access no permite hacer campos calculados en controles que tienen como nombre un nombre de campo existente. ¿Liado? Aplica la primera frase.

²³ Lo llamo “subformulario” no porque sea más feo o más pequeño que el resto. Es idéntico a todos los demás. Lo que pasa es que sé que después lo voy a usar dentro de otro...

Y ahora añadido otro campo en el **encabezado**:

The screenshot shows the 'Préstamos : Formulario' window. The 'Encabezado del formulario' section contains fields for 'Cod Socio', 'Libro', 'Fecha Préstamo', and 'Días'. The 'Días' field is calculated using the formula '=Suma(Fecha)-(Fecha Préstamo)'. Below this, the 'Detalle' section shows a table with columns for 'Cod Socio', 'Cod Libro', 'Fecha Préstamo', and a calculated field for 'Días'.

He usado una función nueva. Para crear un campo calculado de totales hay que hacer dos cosas:

1. Crearlo en un encabezado o en un pie
2. Usar una función de totales. Hay muchas: Suma(), Mínimo(), Máximo(), Cuenta(), Promedio()...

El resultado (arreglando un poco los campos):

The screenshot shows the 'Préstamos' window with a list of loan records. The columns are 'Cod Socio', 'Libro', 'Fecha Préstamo', and 'Días'. The 'Días' column shows the calculated number of days for each loan. The records are as follows:

Cod Socio	Libro	Fecha Préstamo	Días
4	El palacio de la Luna	08/07/01	155
4	Leviatán	09/02/01	304
2	Neuromante	10/01/01	334
3	El tambor de Hojalata	22/11/01	18
1	La Música del Azar	03/12/01	7
3	Leviatán	04/06/01	189
4	Neuromante	08/12/01	2

Y si vemos el resultado en el formulario principal:

The screenshot shows the 'Socios' window. The top section displays the details of a member with 'Cod Socio' 4, 'Nombre' Aitor, 'Apellido 1' Martinez de Antoñana, 'Apellido 2' Ruñiz, 'Teléfono' 66655544, and 'Dirección'. Below this, a list of loans is shown with columns for 'Cod Socio', 'Libro', 'Fecha Préstamo', and 'Días'. The 'Días' column shows the calculated number of days for each loan. The records are as follows:

Cod Socio	Libro	Fecha Préstamo	Días
4	Leviatán	09/02/01	304
4	Neuromante	08/12/01	2

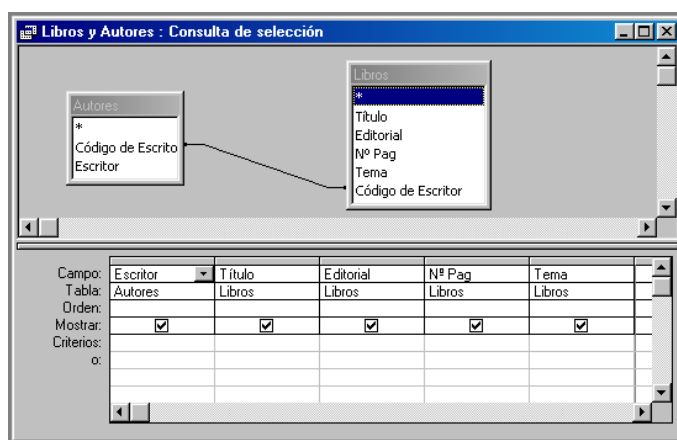
6.5 Informes

Informes y formularios son prácticamente iguales. La diferencia está en que los informes están pensados para impresora, y en un listado es habitual agrupar la información y hacer encabezados. Con los informes se suelen usar consultas. Los listados son los datos que vas a usar, y es raro que los tengas todos en la misma tabla.

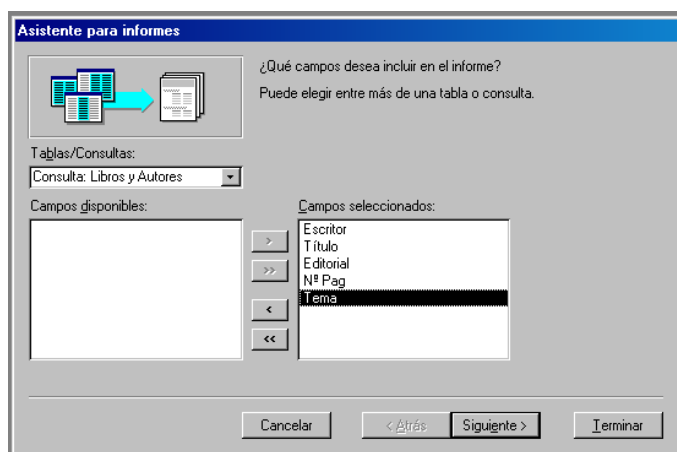
Prácticamente todo lo dicho para formularios se puede aplicar a informes. La diferencia fundamental es que estos últimos tienen un número de pies y encabezados variable.

6.5.1 Crear un informe con el asistente

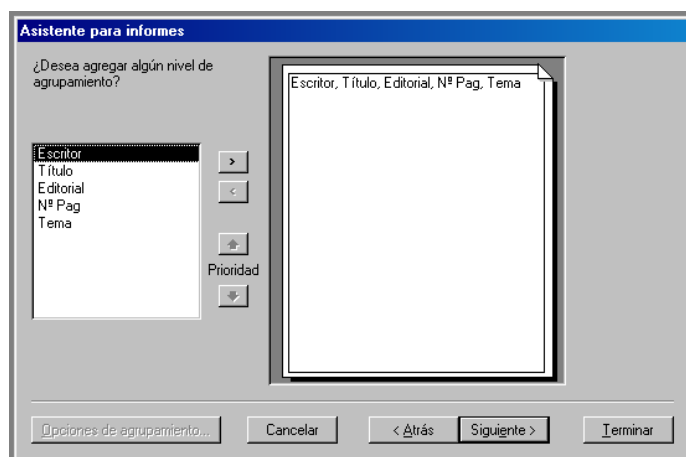
Voy a hacer un listado con los nombres de los autores y la información de los libros. Evidentemente, necesito una consulta:



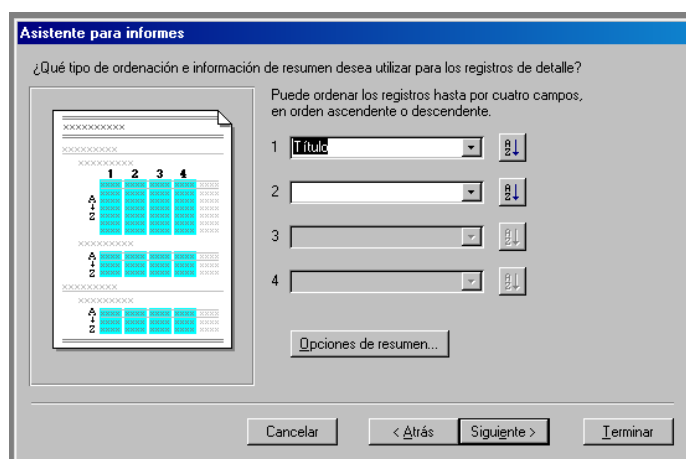
Le he dado un nombre lógico: Libros y Autores. Bien, una vez que tengo los datos “fabricados”, puedo hacer un informe para presentarlos de un amanaera más adecuada. Vamos a la sección de informes y usamos el asistente. La primera pantalla es idéntica a la del asistente para formularios:



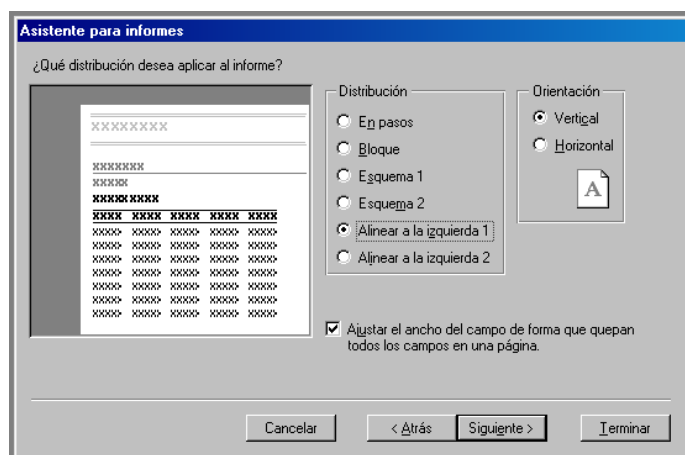
Pulsamos siguiente, y aquí vemos que empiezan a haber diferencias:



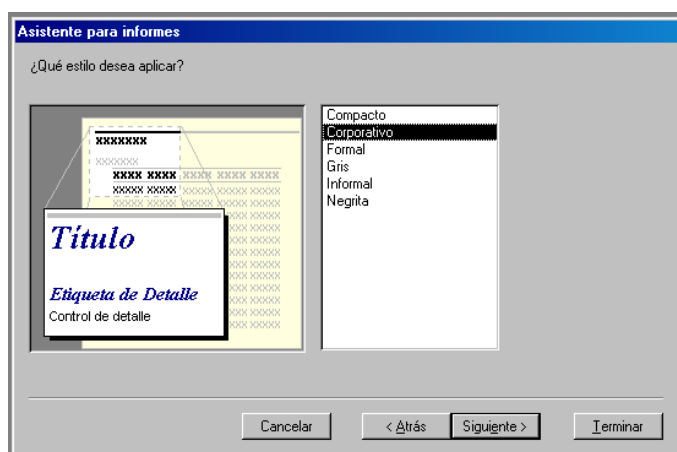
Nos está preguntando si deseamos hacer algún encabezado, algún agrupamiento. Pues sí. Vamos a agrupar el informe por escritores. Elegimos el campo “Escritor” y pulsamos el botón de la flecha. Podemos agrupar por el número de campos que queramos, pero de momento no necesitamos más. Si las relaciones entre las tablas estuvieran bien hechas (ya lo veremos) Access se hubiera dado cuenta de la relación entre autores y libros, y este paso lo habría completado automáticamente. Una vez realizada la agrupación, pulsamos siguiente.



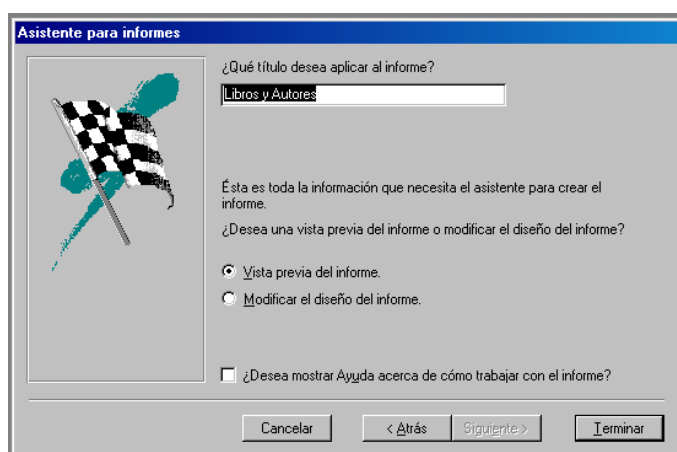
Agrupará y ordenará por nombres de autor, pero dentro de la sección de cada autor, ¿Cómo queremos ordenada la información? El asistente nos da hasta cuatro posibilidades de ordenación. Si no son bastantes se pueden hacer más modificando el diseño del informe. También podemos indicar el orden (ascendente o descendente) y “resúmenes” automáticos: Que al pie del informe aparezcan las sumas de los campos, medias, etc. Por supuesto es útil sólo para campos numéricos. Es más cómo hacerlo en diseño con campos calculados. Lo veremos más adelante.



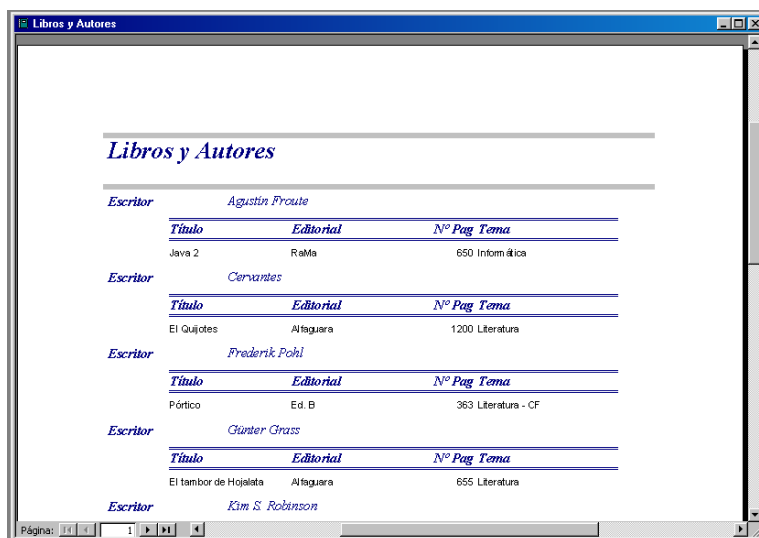
El siguiente paso es el aspecto general de los datos. Apaisado o vertical, posiciones de las cabeceras... cuanto más sencillo lo hagamos, más claramente se leerá el informe y menos cosas tendremos que modificar en el diseño. Pulsamos siguiente.



Tipos y tamaños de letra. Siguiente.



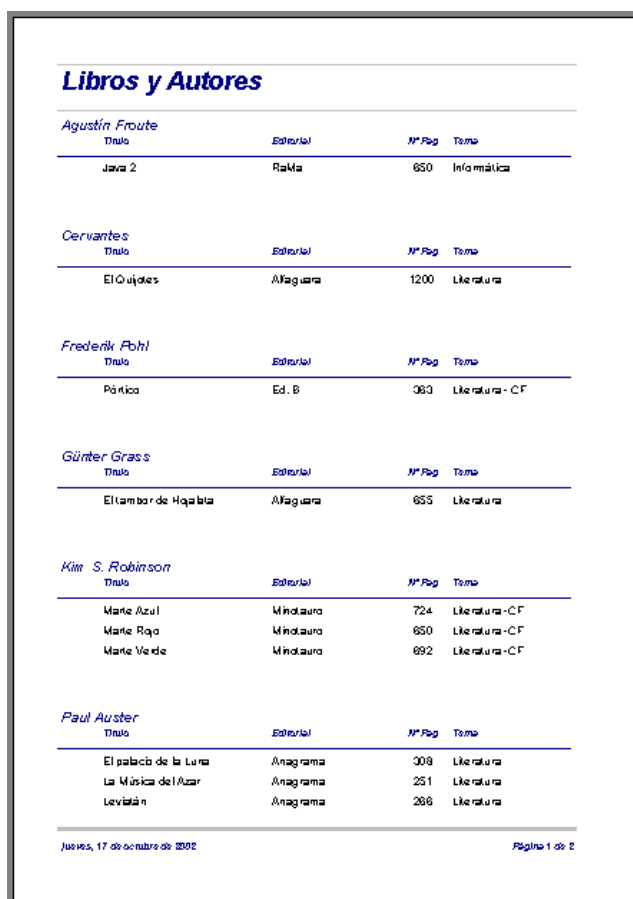
Como en el caso de los formularios, para acabar nos pregunta el nombre con el que queremos guardar el informe. Lo llamaré igual que la consulta: "Libros y Autores". El resultado final será el siguiente:



The screenshot shows a web application window titled "Libros y Autores". It displays a list of authors and their books in a table format. The authors listed are Agustín Froute, Cervantes, Frederik Pohl, Günter Grass, and Kim S. Robinson. Each author's name is followed by a table of their books, with columns for Title, Editorial, Nº Pag, and Tema.

Escritor	Título	Editorial	Nº Pag	Tema
Agustín Froute	Java 2	RaMa	650	Informática
Cervantes	El Quijotes	Alfaguara	1200	Literatura
Frederik Pohl	Pórtico	Ed. B	363	Literatura - CF
Günter Grass	El tambor de Hojalata	Alfaguara	655	Literatura
Kim S. Robinson				

Por supuesto, si en diseño modificamos un poco su aspecto, quedará mucho mejor:



This refined version of the report shows a clean layout with author names as section headers. Each author's name is followed by a table of their books, with columns for Title, Editorial, Nº Pag, and Tema. The layout is more professional and easier to read.

Agustín Froute	Título	Editorial	Nº Pag	Tema
	Java 2	RaMa	650	Informática
Cervantes	Título	Editorial	Nº Pag	Tema
	El Quijotes	Alfaguara	1200	Literatura
Frederik Pohl	Título	Editorial	Nº Pag	Tema
	Pórtico	Ed. B	363	Literatura - CF
Günter Grass	Título	Editorial	Nº Pag	Tema
	El tambor de Hojalata	Alfaguara	655	Literatura
Kim S. Robinson	Título	Editorial	Nº Pag	Tema
	María Azul	Minotauro	724	Literatura - CF
	María Rojo	Minotauro	650	Literatura - CF
	María Verde	Minotauro	692	Literatura - CF
Paul Auster	Título	Editorial	Nº Pag	Tema
	El palacio de la Luna	Anagrama	308	Literatura
	La Música del Azar	Anagrama	251	Literatura
	Leviatán	Anagrama	266	Literatura

Jueves, 17 de octubre de 2002 Página 1 de 2

6.5.2 Diseño de un informe

Veamos cómo se modifica el aspecto de un informe. Voy a hacer un informe que muestre los autores y los libros que han escrito. Hago una consulta, uso el asistente para informes y le digo que agrupe los resultados por autor. De paso, lo modifico un poco:

Autores y Libros		
Agustín Foule	Java 2	950
Frederik Poln	Pánico	360
Günter Grass	El tambor de Hojabia	855
Kim Robinson	María Azul	724
	María Rojo	850
	María Verde	892
	Programación en Windows 95	1100
Miguel De Cervantes Saavedra		

El diseño:

He borrado casi todas las etiquetas, he unido el nombre del escritor en un único campo calculado, y he movido las cosas de sitio. Si nos fijamos, el único elemento distinto entre este informe y los formularios que hemos visto hasta ahora son esos encabezados de página²⁴ y ese extraño “Encabezado Nombre de Escritor”.

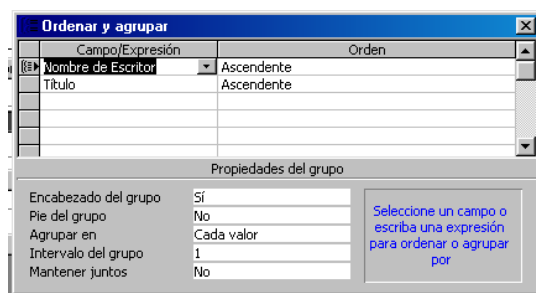
Los encabezados se “ejecutan” cuando que son necesarios. El encabezado y pie de informe se ve en pantalla o impresora una sola vez al inicio y final del formulario o informe. El encabezado y pie de página aparecerían al inicio y final de cada **página** (fíjate en el campo calculado del pie de página), y el encabezado o pie de Nombre de Escritor **cada vez que el escritor cambia**.

Voy a hacer que los escritores aparezcan más separados. Voy a aumentar el tamaño del pie de escritor... Mmm, el asistente no lo ha puesto. Para modificar los pies y encabezados “de datos” de un informe pulsaremos el botón:



²⁴ Un formulario también tiene encabezados de página (menú Ver). No suelen usarse, ya que sólo se ven al imprimir (o en vista previa).

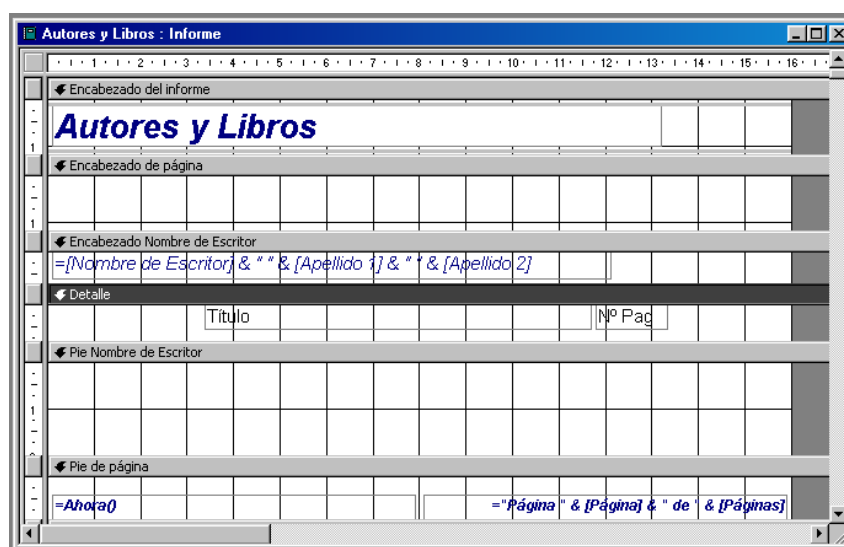
Está situado junto al botón de herramientas. Aparecerá el siguiente cuadro:



Nombre de escritor sirve para agrupar la información²⁵, ya que tiene encabezado. No tiene pie. Bien, se lo ponemos. El resto de opciones se usan para agrupar números. En vez agrupar por cada número diferente (podría valer para un código) puedes agrupar de cinco en cinco, de mes en mes, por cada mil euros, etc.

Título no tiene ni pie ni encabezado. ¿Qué hace ahí? Simplemente ordena el detalle.

Una vez puesto el pie en Nombre de Escritor el diseño queda:



Y el listado:

²⁵ ¿Qué pasaría si tengo dos nombres de escritor iguales? Prueba... Acuérdate de consultas de totales

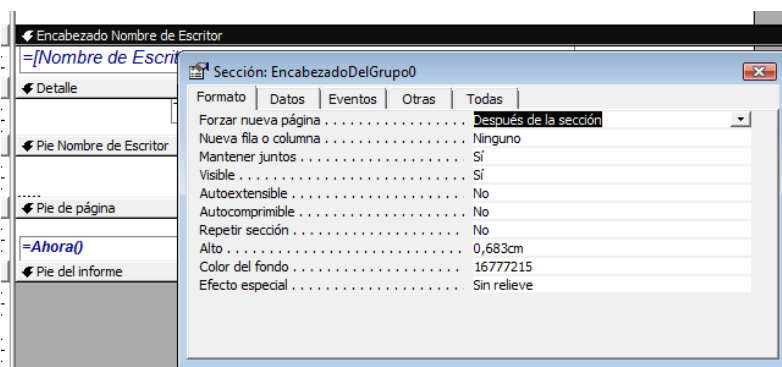
Autores y Libros		
Agustín Fraile	Java 2	850
Frederik Pohl	Párricos	360
Günter Grass	El tambor de Hojalata	655
Kim Robinson	Marte Azul	724
	Marte Rojo	850
	Marte Verde	892
	Programación en Windows 95	1100
Miguel De Cervantes Saavedra	El Quijote	1200
Paul Auster	El palacio de la Luna	308
Jueves, 10 de diciembre de 2009		Página 1 de 2

Es habitual añadir saltos de página en los pies o encabezados, para conseguir que cada grupo de datos salga en páginas distintas. Si añades un salto de página asegúrate de no dejar espacio entre la marca de salto y el final del pie o encabezado, o de lo contrario saldrán páginas en blanco:

Pie Nombre de Escritor		
.....		
Pie de página		
=Ahora()		
Pie del informe		

Pie Nombre de Escritor		
Pie de página		
=Ahora()		
Pie del informe		

Una forma más correcta de conseguir lo mismo es modificando las propiedades de formato del encabezado que nos interese (en este caso el Encabezado de Nombre de Escritor):



Forzar nueva página "Después de la sección" conseguirá que se realice el salto de página sin tener que añadir nada al pie de la sección.

7

Introducción a MySQL. Lenguaje SQL

En este capítulo explicaré las características básicas de MySQL y los comandos más usados del lenguaje SQL.

Con respecto a MySQL el objetivo es que sepas crear diseños, no que lo sepas administrar. Si vas a escribir bases de datos “profesionales” en MySQL **no tendrás suficiente** con el contenido de este apartado. Léete el manual de referencia de MySQL, el del administrador... todo está en la página oficial <http://dev.mysql.com/doc>. Y en Google, claro.

El segundo objetivo del capítulo es aprender a crear y manipular tablas con el lenguaje SQL. Por supuesto no vamos a ver todo el lenguaje en detalle; Pero con lo explicado sabrás realizar las operaciones habituales en una base de datos. Ya que MySQL nos obliga a manejar las bases de datos con SQL, vamos a aprovechar el capítulo para aprenderlo con casos prácticos, en vez de estudiar la las sentencias en frío. En Google y en el manual de referencia de MySQL tienes la sintaxis completa del lenguaje.

Como se verá en los siguientes apartados, no vamos a usar ningún tipo de herramienta gráfica. El objetivo es entender los conceptos de MySQL y de SQL, por lo que es preferible trabajar directamente en la consola de comandos. Por supuesto, cuando la intención sea hacer el trabajo rápido usa las herramientas gráficas gratuitas que te proporciona MySQL. Todo, base de datos y herramientas, lo puedes bajar de <http://dev.mysql.com/downloads>. No hace falta registrarse si no lo deseas.

7.1 Conceptos Básicos de MySQL

Al contrario que en Access, las diferentes bases de datos que crees con MySQL estarán “todas juntas”, almacenadas en algún lugar del disco duro al que tú no tendrás acceso²⁶. La única forma de acceder a los diseños o a los datos es entrando en MySQL. Por tanto, desde el principio tendrás que identificarte con el habitual “login/password”.

La primera vez que ejecutes MySQL sólo habrá un usuario creado, **root**. Es el **administrador principal** de MySQL. Puede hacerlo todo, borrarlo todo, verlo todo. El usuario root es quien habitualmente creará al resto de usuarios, les asignará permisos y creará las bases de datos vacías: El espacio de disco “protegido” donde se almacenarán los diseños y los datos de cada base de datos.

Lo habitual es que haya varias bases de datos creadas dentro de MySQL: Contabilidad, el sitio Web de la empresa, Nóminas... Lógicamente cada base de datos es distinta de la otra, y en principio independiente: No me gustaría que desde la página Web un trabajador pueda cambiarse el salario por un error de programación. Generalmente, el usuario root no mantiene ni crea los diseños de dichas bases de datos, sino que se crean diferentes **usuarios administradores para cada base de datos**.

²⁶ Si lo tienes en tu ordenador Windows estará en algún sitio de “C:/Archivos de programa/MySQL”. En instalaciones serias ni siquiera sabrás donde están los archivos.

Por ejemplo, root crea a los usuarios contabilidad, web y nomina (qué original). ¿Qué pueden hacer esos tipos? Lo que a root le apetezca. Puede crear un gemelo que pueda hacerlo todo o un usuario que únicamente pueda leer ciertos campos de cierta tabla. Lo normal es que root les conceda permisos totales sobre su base de datos y ninguno en absoluto sobre el resto. Se suelen usar para cambiar o crear las tablas.

A partir de aquí la forma de trabajar varía mucho. Se pueden crear usuarios de “sólo lectura”, que únicamente puedan consultar los datos, o de “lectura/escritura”, que además puedan modificarlos pero que no puedan tocar el diseño de las tablas. O cosas mucho más complejas; MySQL permite ajustar los permisos a campos concretos de una tabla.

7.2 Conceptos Básicos de SQL

En primer lugar, una mala noticia. El lenguaje SQL es estándar, pero la implementación que hacen las bases de datos de él no lo es. Te puedes encontrar que ciertas instrucciones funcionan en Access pero no en Oracle o MySQL. En este capítulo vamos a ver lo más típico, por lo que no tendrás problemas de compatibilidad entre bases de datos, pero ten cuidado.

El lenguaje no distingue entre mayúsculas o minúsculas, pero es posible que tu sistema operativo o tu base de datos sí. MySQL para Windows no hace diferencias. Eso se puede cambiar en las opciones de configuración de la base de datos. Y lo mismo pasa con el **contenido** de las tablas.

En los ejemplos del manual, voy a escribir siempre las instrucciones de SQL en mayúsculas, para distinguirlas de los comandos de MySQL, que aparecerán en minúsculas.

Tradicionalmente, las sentencias de SQL se agrupan en dos tipos distintos:

- Sentencias de Definición de Datos (DDS). Instrucciones para crear o modificar estructuras de datos (bases de datos, tablas, vistas). Son las instrucciones CREATE, ALTER, DROP y RENAME.
- Sentencias de Manipulación de Datos (DMS). Instrucciones para crear o modificar filas de las tablas. Hay unas cuantas, pero las que vamos a ver son INSERT, UPDATE, DELETE y sobre todo SELECT.

Por lo general son instrucciones muy sencillas; Las únicas algo más complejas son CREATE y SELECT.

En el manual no mostraré la definición de las sentencias de SQL de forma estricta. Seré clemente (chapucero) con la forma de expresar la sintaxis, a favor de una mayor claridad. Obviaré todos los corchetes y paréntesis que pueda, siempre que se pueda usar la lógica para comprender cómo se escribe correctamente.

Asimismo, en SQL suele haber varias maneras de escribir lo mismo (demasiadas versiones de SQL). Usaré la sintaxis que a mi parecer es la más habitual.

7.3 Iniciando por primera vez

Los programas ejecutables que de momento interesan son estos dos:

- **mysqld**. Se encarga de lanzar el servicio (“el demonio” en sistemas UNIX) MySQL. Si la instalación ha sido correcta, se lanzará y parará sólo y nunca tendrás que tocarlo. Es un servicio de red. Usa el puerto 3306 para atender las peticiones.
- **mysql**. El programa que lanza la consola de comandos. Es aquí donde realizarás todas las tareas relacionadas con MySQL, desde crear usuarios a insertar datos en las tablas (salvo que escribas una aplicación que use la base de datos).

La primera vez que entres en MySQL, si no has dicho nada al instalarlo, entrarás como root, sin necesidad de clave. Obviamente, hay que cambiar eso de inmediato. Para entrar vete a la consola de comandos, busca el programa y ejecútalo sin parámetros. En mi sistema:

```
C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 5.1.31-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> _
```

“**mysql -u root**” hace que entremos como usuario root, sin clave. La primera vez que entres en MySQL no hará falta ni siquiera que especifiques que quieres entrar como root.

Aparecerá el prompt de la base de datos “mysql>”. Ya podemos empezar a ejecutar órdenes. Con la instalación por defecto en Windows MySQL no distingue entre mayúsculas y minúsculas, aunque eso puede variar en otros sistemas. Toda instrucción debe acabar en punto y coma. No importan los espacios o los saltos de línea; Hasta que no escribas el punto y coma MySQL no ejecutará nada.

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| test       |
+-----+
3 rows in set (0.00 sec)

mysql>
```

la instrucción “**show databases;**” muestra las bases de datos existentes. Al parecer, viene con unas cuantas creadas de fábrica:

- **test**. Una base de datos vacía, para hacer pruebas. Lo más recomendable es borrarla. De ese modo, los posibles atacantes no tendrán pistas sobre qué bases de datos “con poca protección” existen en tu sistema.
- **information_schema**. MySQL tiene que almacenar un montón de información sobre qué bases de datos tiene, qué tablas y con qué campos... Las bases de datos, para almacenar toda esa información, usan... una base de datos. Es el **diccionario** de la base de datos o los **metadatos**.
- **mysql**. La base de datos que almacena qué usuarios hay y los permisos que tienen.

Por supuesto, sólo root debería tener permisos para acceder a estas bases de datos. Bien, por defecto es así, pero seguimos teniendo el problema de que cualquiera puede entrar como root. Tenemos que asignarle una clave.

Hay comandos especiales de MySQL para hacer estas cosas, pero es más sencillo usar las sentencias estándares de SQL. Ya que los permisos y usuarios son simples datos de la base de datos “mysql” vamos a conectarnos a ella y a cambiar sus datos:

```
mysql> use mysql;
Database changed
mysql> show tables;
+-----+
| Tables_in_mysql |
+-----+
| columns_priv     |
| db               |
| event            |
| func             |
| general_log      |
| help_category    |
| help_keyword     |
| help_relation    |
| help_topic       |
| host             |
| ndb_binlog_index |
| plugin           |
| proc             |
| procs_priv       |
| servers          |
| slow_log         |
| tables_priv      |
| time_zone        |
| time_zone_leap_second |
| time_zone_name   |
| time_zone_transition |
| time_zone_transition_type |
| user             |
+-----+
23 rows in set (0.01 sec)

mysql>
```

“**use nombre_de_base_de_datos**” nos conecta con una base de datos. Todas las operaciones que realicemos a partir de ahora se harán con sus tablas y datos. “**show tables**” nos muestra un listado con las tablas de la base de datos actual.

Muchas tablas, pero de momento nos interesa sólo una: User. El comando “**desc nombre_de_tabla**” nos muestra los campos de una tabla:

```
mysql> desc user;
```

Field	Type	Null	Key	Default	Extra
Host	char(60)	NO	PRI		
User	char(16)	NO	PRI		
Password	char(41)	NO			
Select_priv	enum('N','Y')	NO		N	
Insert_priv	enum('N','Y')	NO		N	
Update_priv	enum('N','Y')	NO		N	
Delete_priv	enum('N','Y')	NO		N	
. . .					
x509_subject	blob	NO		NULL	
max_questions	int(11) unsigned	NO		0	
max_updates	int(11) unsigned	NO		0	
max_connections	int(11) unsigned	NO		0	
max_user_connections	int(11) unsigned	NO		0	

```
39 rows in set (0.01 sec)

mysql>
```

El sistema la usa para almacenar los usuarios existentes y qué tipos de permisos pueden tener asignados. Para una descripción de cada campo, lee el manual de referencia. Nosotros sólo vamos a utilizar los campos Host, User y Password.

Tienes que comprender cómo funciona el diccionario de la base de datos: No hay “información oculta” que “se ve de algún modo” en estas tablas. El contenido de las tablas ES la información. Si borras una fila de la tabla User, borras al usuario. Si la añades, lo has creado. Vamos a aprovecharnos de eso para

administrar a los usuarios: Usaremos sentencias de SQL estándar para modificar el contenido de la tabla, y por tanto, modificar a los usuarios.

Vamos a ver qué usuarios existen en nuestra base de datos:

```
mysql> SELECT USER,HOST,PASSWORD FROM USER;
```

USER	HOST	PASSWORD
root	localhost	

1 row in set (0.00 sec)

La instrucción del SQL SELECT que hemos usado tiene el formato **SELECT lista_de_campos FROM tabla [WHERE condiciones]**. Si quieres ver todo, basta con poner "*" (sin las comillas) en vez de la lista de campos. Veremos la sintaxis completa de SELECT en el apartado 7.9, "Consultas"

MySQL es un servicio de red. Permite asignar diferentes claves o permitir la entrada del usuario al sistema en función del ordenador desde el que se conecta. En nuestro caso, sólo existe el usuario root, que únicamente puede entrar desde localhost²⁷ (tu propio ordenador). Eso sí, puede entrar sin clave. Ojo, el contenido de Password es "" (cadena vacía), no null.

Si quiero permitir entrar a root desde la red, desde cualquier ordenador, basta con poner "cualquiera" en el campo Host. En MySQL se dice "%".

Si quiero ponerle una clave, no es tan sencillo. La clave no puede estar en texto plano: Sería un fallo de seguridad. Tiene que estar codificada con el algoritmo MD5²⁸. ¿Y cómo se hace eso? Pues usando la función **PASSWORD("texto_a_convertir_a_MD5")**. Suponiendo que la clave que voy a poner es "claveroot", la instrucción del SQL quedaría:

```
mysql> UPDATE USER SET PASSWORD=PASSWORD("claveroot") WHERE USER="root";
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1 Changed: 0 Warnings: 0

mysql> SELECT USER,HOST,PASSWORD FROM USER;
```

USER	HOST	PASSWORD
root	localhost	*BE15B6B074517BBC3F784C4A977CEB4FF2E31564

1 row in set (0.02 sec)

Sólo permito la entrada de root en local (así se dice), y le he asignado una clave. He usado la instrucción de SQL **UPDATE tabla SET lista de campos=valores [WHERE condiciones]**. Si no hubiera puesto la condición, habría cambiado las claves de todos los usuarios. En el ejemplo sólo tengo uno, así que da igual; Pero dependiendo de la instalación, se pueden crear una serie de usuarios "de prueba", que sólo dan problemas de seguridad. Es una buena idea ejecutar la siguiente línea:

```
mysql> DELETE FROM USER WHERE USER<>"root";
Query OK, 0 rows affected (0.00 sec)
```

La sintaxis es **DELETE FROM tabla [WHERE condiciones]**. Cuidado con lo que borras, no hay vuelta atrás.

Queda un detalle importante. Ya que hemos usado sentencias de SQL, el programa en ejecución de MySQL no sabe que sus permisos han cambiado. Tenemos que decirle que vuelva a leer sus tablas:

²⁷ También conocido sobre 127.0.0.1. Busca en Internet cosas sobre el protocolo TCP/IP

²⁸ Un Algoritmo Hash o Resumen. Pregunta a Google, es interesante.

```
mysql>
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye

C:\Program Files\MySQL\MySQL Server 5.1\bin>
```

“**flush privileges**” hace que los cambios realizados en la base de datos mysql se hagan efectivos en ese instante, sin tener que reiniciar el servicio mysqld. “**exit**” finaliza la ejecución de la consola de comandos.

Por último, la instrucción “**help [comando]**” puede proporcionar mucha ayuda, si tener que consultar el manual de referencia.

7.3.1 Formato de almacenamiento

MySQL es una base de datos muy versátil, que permite que los datos se almacenen internamente de diferentes formas. Depende de para qué vayas a usar la base de datos, usarás una u otra.

No voy a explicar ese tipo de conceptos. Sin embargo, para el uso que vas a darle a tus bases de datos, sería recomendable usar el formato de almacenamiento **INNODB**. Lo puedes especificar cada vez que crees una tabla, o bien configurar MySQL para que sea el formato por defecto. Es posible que tu sistema ya esté configurado de esa forma. Para comprobarlo o cambiarlo debes ir al fichero de configuración de MySQL, “**mi.ini**”. En mi sistema está en “C:\Program Files\MySQL\MySQL Server 5.1\mi.ini”:

```
# SERVER SECTION
# -----
#
# The following options will be read by the MySQL server. Make sure that
# you have installed the server correctly (see above) so it reads this
# file.
#
[mysqld]
# The TCP/IP Port the MySQL server will listen on
port=3306

#Path to installation directory. All paths are usually resolved relative to this.
basedir="C:/Program Files/MySQL/MySQL Server 5.1/"

#Path to the database root
datadir="C:/ProgramData/MySQL/MySQL Server 5.1/Data/"

# The default character set that will be used when a new schema or table is
# created and no character set is defined
default-character-set=latin1

# The default storage engine that will be used when create new tables when
default-storage-engine=INNODB
```

Debes buscar la sección sobre configuración del servidor, “[mysqld]” y modificar o escribir la línea “**default-storage-engine=INNODB**”. Eso hará que todas las tablas se almacenen por defecto con ese formato. Tienes que decirle a MySQL que vuelva a leer su fichero de configuración. Si no sabes hacerlo, para y lanza el servicio o reinicia el ordenador.

7.3.2 Elegir una clave

Cuando tengas que escoger una clave, hazlo bien. Hay una serie de reglas básicas:

- Que tenga unos diez caracteres, mezcla de letras, números y símbolos. Truco típico: Cambiar letras por números u otros caracteres (la letra “o” por el número cero...), mayúsculas, puntuaciones.
- No pongas palabras de uso común: mesa, escritorio, llavero, clavesecreta... Truco típico: Parte de la letra de una canción, escritores y obras, películas. Si lo usas junto al anterior, crearás claves muy seguras.

- No uses la misma clave para diferentes ordenadores o sistemas: Si cae uno, caen todos.
- Si necesitas muchas claves, no las crees con sistemas tontos: ordenador1, ordenador2, miclave2008, miclave2009, etc.
- No te enamores de tus claves; Sólo son palabras. Cámbialas de vez en cuando.
- Si no consigues acordarte de todas tus claves, cambiadas cada seis meses, diferentes para cada ordenador, con caracteres raros, **escríbelas**. Por supuesto, no en un posit pegado a la pantalla²⁹, pero sí en un papel que lleves en la cartera, o que guardes en un sitio privado.
- No se la digas a nadie. Si lo haces, cámbiala después.
- Si haces un manual o impartes un curso, usa claves de ejemplo que jamás utilizarías en la vida real.

¿Todo esto es necesario? Pues sí. Sé paranoico, es gratis. Si piensas que “no merece la pena, nadie va a entrar en mi ordenador”, pues no la pongas. Poner una clave a medias se parece mucho a no ponerla.

7.4 Creación de Bases de Datos, Usuarios y Permisos

Si ya tenemos el sistema correctamente configurado, podemos empezar a trabajar con él. Vamos a crear la base de datos “biblioteca”, y un usuario para que la administre: Suele ser una buena idea delegar las responsabilidades de root a otros usuarios, para simplificar la administración.

Podríamos usar la instrucción del SQL **INSERT INTO tabla (lista_de_campos) VALUES (lista_de valores)**. Pero hay dos problemas. Primero, que tenemos que modificar varias tablas. Segundo, que cuando añadimos valores tenemos que especificar todos los campos, a no ser que queramos los valores por defecto (en este caso no nos interesan). Dado que la tabla USER tiene 39 campos, vamos a usar las instrucciones de MySQL para crear bases de datos, usuarios y permisos:

```
C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 5.1.31-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database biblioteca;
Query OK, 1 row affected (0.01 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| biblioteca      |
| mysql          |
| test           |
+-----+
4 rows in set (0.00 sec)

mysql>
```

Para entrar en la consola de comandos he usado “**mysql -u root -p**”³⁰. El parámetro “-u nombre_de_usuario” sirve para indicar el usuario con el que queremos entrar. “-p” indica que a continuación nos pregunte la clave. Si no lo ponemos, trataríamos de entrar sin clave. El comando de SQL **CREATE DATABASE nombre_de_base_de_datos** crea una nueva base de datos. **DROP DATABASE nombre_de_base_de_datos** la borra.

El usuario root puede hacer todas las operaciones con biblioteca. Sin embargo vamos a delegar ese trabajo en el usuario “javi”, con su propia clave y con permisos sólo para “biblioteca”. Podemos usar la instrucción de SQL UPDATE sobre la tabla User, pero es un engorro. Es mucho más cómodo esto otro:

²⁹ Patéticamente habitual.

³⁰ Si no quieres lanzar mysql desde la consola de MSDOS, crea un acceso directo con los parámetros que necesites.

```
mysql> grant all privileges on biblioteca.* to javi identified by 'clavejavi';
Query OK, 0 rows affected (0.00 sec)

mysql>
```

He usado **grant PRIVILEGIOS on BASEDATOS.TABLAS to USUARIO@HOST identified by 'clave'**. he creado un usuario llamado "javi" con "all privileges" sobre todas las tablas de la base de datos "biblioteca". Como no he indicado desde qué ordenador se puede conectar, el sistema habrá escrito "todos". La clave se indica en texto plano; La instrucción grant la codifica automáticamente:

```
mysql> use mysql
Database changed
mysql> SELECT USER,HOST,PASSWORD FROM USER;
+-----+-----+-----+
| USER | HOST   | PASSWORD                                     |
+-----+-----+-----+
| root | localhost | *BE15B6B074517BBC3F784C4A977CEB4FF2E31564 |
| javi | %       | *AED717E762147F7EF626B0111692D35129C6CFF4 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

La sintaxis de grant es muy extensa. Consulta el manual de referencia para más detalles.

En las nuevas versiones de MySQL ha cambiado la forma de almacenar las claves, y ya no se permite crear un usuario y asignarle permisos en un único paso; ahora se deben usar dos comandos distintos:

```
mysql> create user javi identified by 'clavejavi';
Query OK, 0 rows affected (0.10 sec)

mysql> grant all privileges on biblioteca.* to javi;
Query OK, 0 rows affected (0.11 sec)
```

Ya tenemos un nuevo usuario. Para usarlo tenemos que salir de la consola y entrar con el nuevo identificador:

```
C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql -u javi -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 22
Server version: 5.1.31-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use mysql;
ERROR 1044 (42000): Access denied for user 'javi'@'%' to database 'mysql'
mysql> use biblioteca;
Database changed
mysql> show tables;
Empty set (0.00 sec)

mysql>
```

Ya que "soy javi" sólo puedo hacer las tareas permitidas a javi. No puedo entrar en otra base de datos, ni cambiar mis permisos, etc. Tal como lo he creado, puedo hacer cualquier cosa a la base de datos biblioteca, pero sólo a biblioteca. Vamos a emplear a este usuario para administrar dicha base de datos.

7.4.1 Create y Drop Database

Las sentencias de SQL para el manejo de una base de datos son muy sencillas. Para crearla:

```
CREATE DATABASE nombre_base_de_datos [especificaciones]
```

Las especificaciones de la base de datos permiten indicar el juego de caracteres a usar, si distingue entre mayúsculas y minúsculas (y vocales acentuadas) a la hora de comparar contenidos, etc. Consulta el manual de referencia para saber cómo aplicarlo, y ver la sentencia completa del comando.

Para borrar la base de datos:

```
DROP DATABASE nombre_base_da_datos
```

Borra toda la base de datos (tablas y contenido) sin preguntar y sin posibilidad de deshacer, así que úsalo con cuidado. Y haz copias de seguridad. Hay un truco muy útil en MySQL para hacer esto. En la línea de comandos del sistema, en vez de entrar de la manera habitual escribe lo siguiente:

```
C:\Program Files\MySQL\MySQL Server 5.1\bin>mysqldump -u javi -p biblioteca > biblioteca.sql
Enter password: *****

C:\Program Files\MySQL\MySQL Server 5.1\bin>type biblioteca.sql
-- MySQL dump 10.13  Distrib 5.1.31, for Win32 (ia32)
--
-- Host: localhost    Database: biblioteca
--
-- Server version      5.1.31-community

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
--
--
--
DROP TABLE IF EXISTS `libros`;
SET @saved_cs_client      = @@character_set_client;
SET character_set_client  = utf8;
CREATE TABLE `libros` (
  `CODLIBRO` int(11) NOT NULL DEFAULT '0',
  `TITLIBRO` varchar(100) DEFAULT NULL,
  `NUMPAGLIBRO` smallint(6) DEFAULT NULL,
  `CODERTORTAL` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
--
--
```

El comando **mysqldump -u usuario -p nombre_base_datos > fichero** crea un fichero de texto con las instrucciones de SQL necesarias para crear las tablas e insertar los datos necesarios en las mismas de tal modo que se consiga una copia exacta de la base de datos. Se ha elegido texto plano porque de este modo se puede replicar la base de datos original en cualquier motor de base de datos. Consulta el manual de referencia para ver las numerosas opciones del comando.

En el caso de que quisiéramos replicar la base de datos de nuevo en MySQL (se supone que existe la base de datos pero está vacía; si no es así, no cuesta nada modificar el fichero de texto para escribir los comandos adecuados):

```
C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql -u javi -p biblioteca < biblioteca.sql
Enter password: *****

C:\Program Files\MySQL\MySQL Server 5.1\bin>
```

Hay muchas formas de hacer esto mismo: el comando **mysqlimport**, las instrucción **load...**

7.5 Creación de Tablas

La forma de hacerlo es con la sentencia de SQL **CREATE**. Ya que es la primera tabla vamos a hacerla muy simple:

```

C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql -u javi -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.1.31-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use biblioteca;
Database changed
mysql> CREATE TABLE SOCIOS (
-> CODSOCIO INT,
-> NOMSOCIO VARCHAR(20),
-> AP1SOCIO VARCHAR(20),
-> AP2SOCIO VARCHAR(20),
-> TELSOCIO VARCHAR(10),
-> DIRSOCIO VARCHAR(50),
-> FECNACSOCIO DATE);
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+-----+
| Tables_in_biblioteca |
+-----+
| socios                |
+-----+
1 row in set (0.00 sec)

mysql> desc socios;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CODSOCIO   | int(11)       | YES  |     | NULL    |       |
| NOMSOCIO   | varchar(20)   | YES  |     | NULL    |       |
| AP1SOCIO   | varchar(20)   | YES  |     | NULL    |       |
| AP2SOCIO   | varchar(20)   | YES  |     | NULL    |       |
| TELSOCIO   | varchar(10)   | YES  |     | NULL    |       |
| DIRSOCIO   | varchar(50)   | YES  |     | NULL    |       |
| FECNACSOCIO | date          | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql>

```

Faltan claves, etc. para completar la definición de la tabla; Se verá en los apartados siguientes. Los tipos de datos empleados son los explicados en el apartado 2.1.1.2, “Tipos de datos de MySQL”, por lo que no los voy a repetir otra vez.

Para borrar la definición de la tabla junto con todos sus datos se usa el comando de SQL **DROP TABLE nombre_de_tabla**.

Hay un punto al que nunca se le da la importancia adecuada: **Los nombres empleados**. Puedes llamar a los campos y tablas como desees; Pero hay una diferencia fundamental con Access: Ya que no hay herramientas gráficas, tienes que escribirlo todo. Tienes que acordarte de ellos.

Si vas a crear unas quince tablas, y cada tabla tiene unos diez campos... demasiados nombres. **Usa un sistema de nombrado**, y úsalo hasta el fin. Por ejemplo, yo voy a nombrar las tablas siempre en plural y los campos en singular. Siempre emplearé tres letras para indicar el contenido del campo y a continuación el nombre de la tabla en singular... Hazlo como desees, pero siempre igual. Te ahorrarás un montón de errores.

7.5.1 Create, Alter y Drop Table

La sentencia para creación de tablas en SQL:

```
CREATE TABLE nombre_tabla (
Nombre_columna tipo_columna [NOT NULL | NULL ] [DEFAULT
valor_defecto] [AUTO_INCREMENT],...
PRIMARY KEY [nombre_índice](nombre_columna,...),
KEY [nombre_índice] (nombre_columna,...),
UNIQUE [nombre_índice] (nombre_columna,...),
FOREIGN KEY [nombre_índice] (nombre_columna,...) REFERENCES
nombre_tabla_referenciada (nombre_columna,...))
```

Todas las claves son opcionales, y se pueden poner varias en función del tipo: Obviamente sólo puede existir una clave principal, pero no hay límite para el resto. Tendrás que ver el apartado 7.7, “Claves” y 7.8, “Relaciones: Claves Foráneas” para comprender qué son exactamente y ver la sintaxis completa.

“Nombre_índice” es un nombre que podemos poner a las claves creadas. Si después queremos borrarlas será más sencillo que usar el nombre proporcionado por el sistema. Personalmente sólo lo hago para las claves foráneas. Y generalmente uso otra sintaxis que es equivalente:

```
...
CONSTRAINT [nombre_índice] FOREIGN KEY (nombre_columna,...)
REFERENCES nombre_tabla_referenciada (nombre_columna,...))
```

Excepto con KEY, se puede hacer lo mismo con el resto de claves.

Cuando se define una columna se debe indicar el nombre, el tipo y opcionalmente si se permiten o no valores NULL. También se puede indicar el valor por defecto que se asignará a ese campo al insertar una fila.

AUTO_INCREMENT sólo se aplica a campos numéricos. El sistema se compromete a escribir un número en el campo que siempre será diferente a todos los demás que se hayan usado en dicha columna (sencillamente tiene un contador al que le suma un uno cada vez que lo usa). Es ideal para claves primarias en las que no nos importa el contenido (código de autor, por ejemplo).

La sintaxis para borrar una tabla es más sencilla:

```
DROP TABLE nombre_tabla
```

No pregunta ni permite deshacer, así que cuidado al usarla. Si se infringen las reglas de integridad referencial (claves foráneas) no permitirá borrar la tabla.

Por último, para modificar una tabla ya creada:

```
ALTER TABLE nombre_tabla
ADD [COLUMN | INDEX | UNIQUE | PRIMARY KEY | FOREIGN KEY ] opciones
ALTER COLUMN [SET DEFAULT valor | DROP DEFAULT]
CHANGE COLUMN nombre_columna nombre_columna_nuevo definición_nueva
DROP [COLUMN | INDEX | UNIQUE | PRIMARY KEY | FOREIGN KEY ] opciones
```

Como en el caso de CREATE, he simplificado la forma de expresar la sintaxis. “Opciones” depende de lo que se esté modificando: Será la lista de campos que componen la clave principal, o si se trata de añadir una columna habrá que escribir el nombre del campo, el tipo, etc. Por ejemplo:

```
ALTER TABLE SOCIOS ADD COLUMN DNI CHAR(8);
ALTER TABLE SOCIOS CHANGE COLUMN DNI DNISOCIO VARCHAR(9);
ALTER TABLE SOCIOS DROP COLUMN DNISOCIO;
```

7.6 Operaciones con los datos

Vamos a ver como manipulamos datos de una tabla. Realmente ya no estamos estudiando MySQL, sino SQL. Ya lo hemos explicado en apartados anteriores al crear, borrar y modificar filas de la tabla USER; En este apartado vamos a repetir todo eso, pero con más detalle.

Para **añadir** socios a la tabla:

```
mysql> INSERT INTO SOCIOS
-> (CODSOCIO,NOMSOCIO,AP1SOCIO,AP2SOCIO,TELSOCIO,DIRSOCIO,FECNACSOCIO)
-> VALUES (1,"Javier","Rodriguez","Diez","900100010","VITORIA-GASTEIZ",
-> "1970-09-07");
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> INSERT INTO SOCIOS
-> (CODSOCIO,NOMSOCIO,AP1SOCIO,AP2SOCIO,TELSOCIO,DIRSOCIO,FECNACSOCIO)
-> VALUES (2,"Maria","Arresti","Perez",null,"BILBAO","1965-09-24");
Query OK, 1 row affected (0.03 sec)

mysql>
mysql> INSERT INTO SOCIOS
-> (CODSOCIO,NOMSOCIO,AP1SOCIO,AP2SOCIO,TELSOCIO,DIRSOCIO,FECNACSOCIO)
-> VALUES (3,"Juana","Arregui","Leioa","999112233","VITORIA-GASTEIZ",
-> "1979-11-23");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM SOCIOS;
```

CODSOCIO	NOMSOCIO	AP1SOCIO	AP2SOCIO	TELSOCIO	DIRSOCIO	FECNACSOCIO
1	Javier	Rodriguez	Diez	900100010	VITORIA-GASTEIZ	1970-09-07
2	Maria	Arresti	Perez	NULL	BILBAO	1965-09-24
3	Juana	Arregui	Leioa	999112233	VITORIA-GASTEIZ	1979-11-23

```
3 rows in set (0.00 sec)
```

Los textos y fechas se escriben entre comillas dobles y simples. Las fechas siempre se indican de mayor a menor: año, mes, día, hora, minutos, segundos.

Para **modificar** datos:

```
mysql> UPDATE SOCIOS SET DIRSOCIO='BILBAO';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 3 Changed: 2 Warnings: 0

mysql> UPDATE SOCIOS SET TELSOCIO="112233445" WHERE CODSOCIO=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM SOCIOS;
```

CODSOCIO	NOMSOCIO	AP1SOCIO	AP2SOCIO	TELSOCIO	DIRSOCIO	FECNACSOCIO
1	Javier	Rodriguez	Diez	900100010	BILBAO	1970-09-07
2	Maria	Arresti	Perez	112233445	BILBAO	1965-09-24
3	Juana	Arregui	Leioa	999112233	BILBAO	1979-11-23

```
3 rows in set (0.00 sec)
```

En el primer caso no he indicado a quién quería cambiar la dirección, por lo que se la he cambiado a todos los socios. Por supuesto, puedo cambiar varios campos a la vez, separándolos por comas. MySQL siempre indica cuántas columnas ha cambiado, cuántas se han visto afectadas, si ha habido algún problema, etc.

Para **borrar** filas:

```
mysql> DELETE FROM SOCIOS WHERE NOMSOCIO='Maria';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM SOCIOS;
```

CODSOCIO	NOMSOCIO	AP1SOCIO	AP2SOCIO	TELSOCIO	DIRSOCIO	FECNACSOCIO
1	Javier	Rodriguez	Diez	900100010	BILBAO	1970-09-07
3	Juana	Arregui	Leioa	999112233	BILBAO	1979-11-23

```
2 rows in set (0.00 sec)
```

No indico qué campos quiero borrar porque en una base de datos se borran siempre filas completas. Si no hubiera usado la cláusula WHERE habría borrado todas las filas de la tabla.

7.6.1 Insert, Update y Delete

La sentencia de SQL para añadir registros a una tabla:

```
INSERT INTO nombre_tabla VALUES(nombre_columna,...) (valores,...)
```

También admite esta otra sintaxis:

```
INSERT INTO nombre_tabla SET nombre_columna=valor,...
```

Inserta una nueva fila con los valores indicados. Si se trata de un texto o de una fecha deberá ir entre comillas dobles o simples.

En el caso de fechas el dato irá expresado de “mayor a menor” (año, mes, día...). Se pueden incluir o no separadores entre los elementos. Valores de fechas correctos serían:

```
"1970-09-7"
"1970/9/07"
"19700907"
```

la sentencia para modificar valores es más compleja, pues permite escribir condiciones para seleccionar qué filas queremos modificar:

```
UPDATE nombre_tabla SET nombre_columna=valor,... WHERE condiciones
```

Las cláusula WHERE es muy potente, Permite expresar multitud de condiciones distintas. Se verá con detalle en el apartado dedicado a la instrucción SELECT. Algunos ejemplos:

```
UPDATE SOCIOS SET AP1SOCIO="Desconocido" WHERE AP1SOCIO=NULL;
UPDATE SOCIOS SET DIRSOCIO="Vitoria" WHERE TELSOCIO LIKE "945%";
UPDATE SOCIOS SET NOMSOCIO="Maria" WHERE NOMSOCIO IN ("Ma", "Mari",
"M.", "M");
UPDATE SOCIOS SET NOMSOCIO="Incompleto" WHERE (NOMSOCIO IS NULL OR
AP1SOCIO IS NULL OR AP2SOCIO IS NULL);
```

Por último, para borrar filas:

```
DELETE FROM nombre_tabla WHERE condiciones
```

No se indica el nombre de las columnas porque en una base de datos siempre se borran filas completas. Las condiciones que podemos expresar son las mismas que en la instrucción UPDATE o SELECT.

7.7 Claves

Al igual que en Access, hay que indicar qué claves principales y secundarias queremos en las tablas. Por supuesto, tenemos que hacerlo usando SQL. Podemos hacerlo directamente en la sentencia **CREATE** o podemos usar la sentencia **ALTER** para modificar una tabla ya creada:


```
mysql> ALTER TABLE SOCIOS ADD PRIMARY KEY (CODSOCIO);
Query OK, 2 rows affected (0.09 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
mysql> CREATE TABLE LIBROS (
-> CODLIBRO INT,
-> TITLIBRO VARCHAR (100),
-> NUMPAGLIBRO SMALLINT,
-> CODEDRTORIAL INT,
-> PRIMARY KEY (CODLIBRO));
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> desc socios;
```

Field	Type	Null	Key	Default	Extra
CODSOCIO	int(11)	NO	PRI	0	
NOMSOCIO	varchar(20)	YES		NULL	
AP1SOCIO	varchar(20)	YES		NULL	
AP2SOCIO	varchar(20)	YES		NULL	
TELSOCIO	varchar(10)	YES		NULL	
DIRSOCIO	varchar(50)	YES		NULL	
FECNACSOCIO	date	YES		NULL	

7 rows in set (0.00 sec)

```
mysql> desc libros;
```

Field	Type	Null	Key	Default	Extra
CODLIBRO	int(11)	NO	PRI	0	
TITLIBRO	varchar(100)	YES		NULL	
NUMPAGLIBRO	smallint(6)	YES		NULL	
CODEDRTORIAL	int(11)	YES		NULL	

4 rows in set (0.00 sec)

Por supuesto, podemos crear todo tipo de claves secundarias:

```
mysql> ALTER TABLE SOCIOS ADD INDEX (AP1SOCIO, AP2SOCIO, NOMSOCIO);
Query OK, 2 rows affected (0.03 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
mysql> desc socios;
```

Field	Type	Null	Key	Default	Extra
CODSOCIO	int(11)	NO	PRI	0	
NOMSOCIO	varchar(20)	YES		NULL	
AP1SOCIO	varchar(20)	YES	MUL	NULL	
AP2SOCIO	varchar(20)	YES		NULL	
TELSOCIO	varchar(10)	YES		NULL	
DIRSOCIO	varchar(50)	YES		NULL	
FECNACSOCIO	date	YES		NULL	

7 rows in set (0.00 sec)

Hemos creado un índice múltiple (permite repeticiones) con los apellidos y nombres de los socios. La palabra reservada **INDEX** crea índices y claves con repeticiones. **UNIQUE** crea índices o claves únicos. En general, todo lo que pueda usarse con ALTER se puede indicar directamente en CREATE:

```
mysql> CREATE TABLE PRESTAMOS (
-> CODSOCIO INT,
-> CODLIBRO INT,
-> FECENTPRESTAMO DATETIME,
-> FECDEVPRESTAMO DATETIME,
-> PRIMARY KEY (CODSOCIO,CODLIBRO,FECENTPRESTAMO),
-> UNIQUE (CODSOCIO,CODLIBRO,FECDEVPRESTAMO));
Query OK, 0 rows affected (0.00 sec)
```


7.8 Relaciones: Claves Foráneas

Tenemos tres tablas: Socios, Libros y Préstamos. Hay una relación de uno a N entre Socios y Préstamos y entre Libros y Préstamos. Como todo lo que hemos hecho en MySQL, tenemos que especificarlas usando SQL; La forma de establecer relaciones, claves foráneas o **foreign keys** es de nuevo a través de ALTER o CREATE:

```
mysql> ALTER TABLE PRESTAMOS ADD FOREIGN KEY (CODLIBRO) REFERENCES LIBROS(CODLIBRO);
Query OK, 0 rows affected (0.33 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> show create table prestamos;
+-----+
| prestamos | CREATE TABLE `prestamos` (
  `CODSOCIO` int(11) NOT NULL DEFAULT '0',
  `CODLIBRO` int(11) NOT NULL DEFAULT '0',
  `FECENTPRESTAMO` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `FECDEVPRESTAMO` datetime DEFAULT NULL,
  PRIMARY KEY (`CODSOCIO`,`CODLIBRO`,`FECENTPRESTAMO`),
  UNIQUE KEY `CODSOCIO` (`CODSOCIO`,`CODLIBRO`,`FECENTPRESTAMO`,`FECDEVPRESTAMO`),
  KEY `CODLIBRO` (`CODLIBRO`),
  CONSTRAINT `prestamos_ibfk_1` FOREIGN KEY (`CODLIBRO`) REFERENCES `libros` (`CODLIBRO`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+
```

Las claves foráneas siempre se establecen en la parte N de la relación. Como al resto de claves de MySQL, el sistema les asigna un nombre si nosotros no le ponemos ninguno (por ejemplo para poder borrarlo). La instrucción para poder ver dichos nombres es **show create table nombre_de_tabla**.

A partir de ahora, no podemos referirnos a códigos de libros dentro de Préstamos que no existan en la tabla Libros. Como en el caso de Access, podemos decidir qué pasa en Préstamos si modificamos o borramos el libro. Voy a borrar la clave foránea y rehacerla con más opciones. De paso, le asignaré un nombre más significativo que “prestamos_ibfk_1”:

```
mysql> ALTER TABLE PRESTAMOS DROP FOREIGN KEY prestamos_ibfk_1;
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE PRESTAMOS ADD CONSTRAINT FK_PRESTAMOS_LIBROS
-> FOREIGN KEY (CODLIBRO) REFERENCES LIBROS (CODLIBRO)
-> ON DELETE CASCADE ON UPDATE CASCADE;
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
mysql> ALTER TABLE PRESTAMOS ADD CONSTRAINT FK_PRESTAMOS_SOCIOS
-> FOREIGN KEY (CODSOCIO) REFERENCES SOCIOS (CODSOCIO)
-> ON DELETE CASCADE ON UPDATE CASCADE;
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

He creado dos claves foráneas con los nombres “FK_PRESTAMOS_LIBROS” y “FK_PRESTAMOS_SOCIOS”. Además le he dicho que si se borran o modifican los códigos de libros de la tabla Libros, o los códigos de socios de la tabla Socios los cambios se apliquen en cascada en la tabla Préstamos. MySQL permite varias operaciones³¹:

- CASCADE. Borra o actualiza en cascada.
- RESTRICT, NO ACTION: Rechaza la operación de modificación o borrado. No puedes modificar la parte uno de la relación si antes no modificas el registro de la parte N.
- SET NULL: Si se cambia la parte uno de la relación, la parte N se establece a NULL, siempre que ese valor sea válido para dicha columna.

La sintaxis completa para expresar una clave foránea (ya sea usando CREATE O ALTER):

³¹ El formato de almacenamiento INNODB permite todas las opciones. Otros formatos de MySQL no...

```
...
[CONSTRAINT nombre_índice] FOREIGN KEY (nombre_columna,...)
REFERENCES nombre_tabla_referenciada (nombre_columna,...)
[ON DELETE CASCADE | RESTRICT | SET NULL]
[ON UPDATE CASCADE | RESTRICT | SET NULL]
```

7.9 Consultas. Select

Consisten en sentencias SELECT de SQL. La orden de “selección” de datos es la más potente y sin duda la más usadas del lenguaje de bases de datos. En este apartado veremos unos cuantos ejemplos de su uso, así como las funciones y operadores más habituales.

Al contrario que en Access, no hay diseñadores que nos ayuden a escribir las instrucciones. Sin embargo sí que podemos “guardar” las sentencias más usadas usando las **vistas**. Sirven para más cosas que simples alias de instrucciones SELECT, pero eso es parte de un curso más completo de MySQL.

Para los ejemplos, vamos a escribir unos cuantos datos en las tablas de Libros y Préstamos:

```
mysql> INSERT INTO LIBROS (CODLIBRO,TITLIBRO,NUMPAGLIBRO,CODEEDITORIAL) VALUES (1,"El Quijote I",640,NULL);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO LIBROS (CODLIBRO,TITLIBRO,NUMPAGLIBRO,CODEEDITORIAL) VALUES (2,"El Quijote II",540,NULL);
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO LIBROS (CODLIBRO,TITLIBRO,NUMPAGLIBRO,CODEEDITORIAL) VALUES (3,"Neuromante",456,NULL);
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO LIBROS (CODLIBRO,TITLIBRO,NUMPAGLIBRO,CODEEDITORIAL) VALUES (4,"El Tambor de Hojalata",837,NULL);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO LIBROS (CODLIBRO,TITLIBRO,NUMPAGLIBRO,CODEEDITORIAL) VALUES (5,"Pipo y la granja",36,NULL);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PRESTAMOS (CODSOCIO,CODLIBRO,FECENTPRESTAMO,FECDEVPRESTAMO) VALUES (1,2,"2009-12-4",NULL);
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO PRESTAMOS (CODSOCIO,CODLIBRO,FECENTPRESTAMO,FECDEVPRESTAMO) VALUES (1,1,"2009-12-4",NULL);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PRESTAMOS (CODSOCIO,CODLIBRO,FECENTPRESTAMO,FECDEVPRESTAMO) VALUES (2,1,"2009-12-1","2009-12-3");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PRESTAMOS (CODSOCIO,CODLIBRO,FECENTPRESTAMO,FECDEVPRESTAMO) VALUES (2,4,"2009-12-4","2009-12-5");
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO PRESTAMOS (CODSOCIO,CODLIBRO,FECENTPRESTAMO,FECDEVPRESTAMO) VALUES (2,5,"2009-12-4",NULL);
Query OK, 1 row affected (0.00 sec)
```

La sentencia SELECT es muy versátil. Permite seleccionar los datos de multitud de maneras, por lo que su sintaxis es extensa:

```
SELECT [DISTINCT] lista_columnas FROM lista_tablas
[WHERE condiciones]
[GROUP BY nombre_columna,...]
[HAVING condiciones]
[ORDER BY nombre_columna [ASC | DESC],...]
[LIMIT numero_filas]
```

Lista_columnas es la lista de nombres de columnas separadas por comas que queremos mostrar.

Lista_tablas es la lista de tablas de las cuales se quieren extraer los datos.

WHERE permite expresar las condiciones a través de las cuales filtraremos los datos.

GROUP BY nos permite agrupar los datos en función de ciertas columnas clave.

HAVING sólo se puede usar si se ha agrupado previamente. Es muy similar a WHERE, salvo que se aplica **después** de agrupar.

ORDER BY permite ordenar los datos finales mostrados como deseemos.

LIMIT permite limitar el número de filas mostradas.

7.9.1 Columnas

Se puede usar un **asterisco** para indicar que se quieren visualizar todas las columnas de cierta tabla. Siempre que se desee se puede usar la sintaxis **nombre_tabla.nombre_columna**. Es muy útil cuando se hacen búsquedas en varias tablas y coinciden los nombres de las columnas.

SQL permite cambiar el nombre de columna que se mostrará en la pantalla usando la cláusula AS:

```
SELECT CODSOCIO AS codigo, NOMSOCIO as nombre, APlSOCIO as apellido
FROM SOCIOS;
```

Asimismo, podemos modificar el resultado mostrado aplicando funciones de MySQL y SQL, así como diversos operadores. Realmente casi todas las funciones y operadores se pueden usar en cualquier sentencia de SQL, no sólo SELECT. Pero es en ésta donde se aplican habitualmente.

7.9.1.1 Operadores

Son los operadores de cualquier lenguaje de programación:

- **Aritméticos:** “+”, “-”, “*”, “/”, “%” (módulo o resto) y DIV (división entera).
- **Lógicos:** Todas las operaciones se evalúan como TRUE, FALSE o NULL. En MySQL se implementan internamente como 1, 0 y NULL. Como en toda base de datos, cualquier operación con NULL devuelve NULL. Los operadores son los de siempre: AND o “&”, OR o “||”, NOT o “!” y XOR.
- **De comparación:** Como en el caso anterior, todo se evalúa a TRUE, FALSE o NULL.
 Los de siempre: “=”, “!=” o “<>”, “<”, “<=”, “>”, “>=”
 comparación **ANY** (lista valores). Cierta si la comparación se cumple con cualquier elemento de la lista:

```
... WHERE CAMPO > ANY (elem1, elem2,...)
```


 expresión **IS** [NOT] TRUE | FALSE | NULL
 expresión [NOT] **BETWEEN** mínimo AND máximo
 expresión [NOT] **IN** (valor, valor2,...)
ISNULL(expresión)
INTERVAL (num, num1, num2,...) devuelve el número de intervalo en el que se encuentra num.
INTERVAL numero DAY | HOUR | SECOND... genera ese “tiempo” para sumarlo o restarlo de una fecha.
 expresión [NOT] **LIKE** patrón. Se verá con más detalle junto con WHERE.
 expresión [NOT] **REGEXP** expresión_regular. Se verá con más detalle junto con WHERE.
- **Otros.**
BINARY cadena_texto. hace que el texto se vea como una cadena de bytes, no de caracteres, por lo que distingue entre mayúsculas y minúsculas, acentos, etc.

7.9.1.2 Funciones

Podemos aplicar ciertas funciones a los datos mostrados. Mostrar el año de una fecha, contar o sumar los datos en vez de mostrarlos directamente en pantalla, etc. Muchas de estas funciones se usan en

conjunción con los agrupamientos de datos. Hay decenas de funciones disponibles. Consulta el manual de referencia para mas información.

Observa que no he dejado espacios entre el nombre de la función y los paréntesis que encierran los parámetros; En MySQL genera un error. Las mas usadas:

- **CHAR_LENGTH(texto)** devuelve el número de caracteres del texto. Los caracteres que ocupan varios bytes (saltos de línea en Windows...) cuentan como uno.
- **COMPRESS(texto), UNCOMPRESS(texto)** comprime o descomprime en formato ZIP el texto.
- **CONCAT(texto1,texto2,...)** concatena los textos en una única cadena de caracteres.
- **LCASE(texto), LOWER(texto)** devuelve el texto en minúsculas.
- **LEFT(texto, num)** devuelve el número de caracteres indicado del texto empezando por la izquierda.
- **LENGTH(texto)** da el número de bytes que ocupa el texto.
- **LTRIM(texto)** elimina los caracteres en blanco a la izquierda del texto. Muy útil cuando almacenamos en la base de datos textos provenientes de entradas por teclado del usuario.
- **RIGHT(texto, num)** devuelve el número de caracteres indicado del texto empezando por la derecha.
- **RTRIM(texto)** elimina los caracteres en blanco a la derecha del texto.
- **TRIM(texto)** elimina los caracteres en blanco que rodean al texto. Consulta el manual para ver su sintaxis completa.
- **UCASE(texto), UPPER(texto)** devuelve el texto en mayúsculas.
- **STRCMP(texto1, texto2)** compara los dos textos y devuelve -1,0 o 1.
- **Funciones matemáticas.** SQL posee las operaciones matemáticas típicas: ABS, SEN, COS, POW, RAND, ROUND, SQRT, etc.
- **ADDTIME(fecha, tiempo)** devuelve una nueva fecha a partir de los argumentos. Para expresar tiempo se puede usar el operador INTERVAL para expresar el tiempo.
- **CURDATE()** devuelve la fecha del sistema en formato YYYYMMDD.
- **CURTIME()** devuelve la hora del sistema en formato HHMMSS.
- **DATE(texto)** convierte el texto en una fecha.
- **DATE_FORMAT(fecha, formato)** devuelve un texto formateado. Consulta el manual para ver las opciones de formato.
- **DATEDIF(fecha1, fecha2)** devuelve la diferencia en días entre las fechas.
- **DAY(fecha)** da el día de la fecha.
- **DAYNAME(fecha)** devuelve el nombre del día de la semana de la fecha.
- **DAYOFWEEK(fecha)** devuelve el índice del día de la semana de la fecha.
- **HOUR(time)** la hora de la fecha.
- **LAST_DAY(fecha)** devuelve la fecha correspondiente al último día del mes de la fecha.
- **MINUTE(fecha).** Minutos de la fecha.
- **MOUNTH(fecha).** Número de mes.
- **MOUNTHNAME(fecha).** Nombre del mes.
- **NOW()** devuelve la fecha y hora del sistema.
- **SECOND(fecha).** Los segundos de la fecha.
- **SUBTIME(fecha, tiempo)** resta el tiempo de esa fecha.

- **TIME(texto)** devuelve la hora de la fecha.
- **TO_DAYS(fecha)** convierte la fecha a días.
- **WEEK(fecha)** devuelve el número de la semana. Tiene parámetros adicionales.
- **YEAR(fecha)**. El año de la fecha.

Las funciones que vienen a continuación son especiales. Sirven para agrupar los datos; En vez de una lista de importes se obtiene la suma de todos ellos, o la media. Se suelen usar junto a GROUP BY, aunque no es obligatorio.

- **AVG(expresión)** devuelve el valor medio.
- **COUNT([DISTINCT] expresión)** cuenta cuántas elementos no NULL ha encontrado. **COUNT(*)** cuenta el número de filas, independientemente de si contienen o no NULL. Si se usa DISTINCT sólo tiene en cuenta valores no repetidos.
- **MIN(expresión)**. El valor mínimo.
- **MAX(expresión)**. El valor máximo.
- **STD(expresión)**. La desviación estándar.
- **SUM([DISTINCT] expresión)**. La suma de los valores.
- **VARIANCE(expresión)** devuelve la varianza estándar.

7.9.2 Tablas

Depende de la sintaxis usada, a menudo en **lista_tablas** sólo se indica el nombre de una tabla aunque los datos mostrados estén en varias. Lo veremos más adelante cuando estudiemos la cláusula JOIN.

Permite hacer más cosas de lo que parece a simple vista. En este apartado podemos hacer un SELECT y hacer que se comporte como si fuera una nueva tabla:

```
mysql> SELECT SOCIOS.NOMSOCIO, PRES.* FROM SOCIOS,
-> (SELECT * FROM PRESTAMOS WHERE FECDEVPRESTAMO IS NULL) AS PRES
-> WHERE SOCIOS.CODSOCIO=PRES.CODSOCIO;
```

NOMSOCIO	CODSOCIO	CODLIBRO	FECENTPRESTAMO	FECDEVPRESTAMO
Javier	1	1	2009-12-04 00:00:00	NULL
Javier	1	2	2009-12-04 00:00:00	NULL
Maria	2	5	2009-12-04 00:00:00	NULL

3 rows in set (0.00 sec)

Por supuesto, hay maneras mucho mejores de conseguir este resultado; Es sólo un ejemplo de la versatilidad de la sentencia SELECT.

7.9.3 Where y Having. Condiciones

Las condiciones que podemos expresar con WHERE o HAVING son muchas. Básicamente podemos usar cualquier combinación de operadores lógicos y de comparación que se nos ocurra:

```

SELECT * FROM PRESTAMOS WHERE FECDEVPRESTAMO IS NULL AND
DATEDIFF(NOW(),FECENTPRESTAMO)>15;

SELECT * FROM SOCIOS WHERE NOMSOCIO="Javier" AND
YEAR(FECNACSOCIO)>1960;

SELECT NOMSOCIO,TELSOCIO FROM SOCIOS WHERE DIRSOCIO LIKE "%VITORIA%"
AND LEFT(TELSOCIO,3)<>"945";

```

Los operadores de comparación LIKE y REGEXP son muy útiles. **LIKE** permite comparar textos usando caracteres comodín. "%" equivale a "cualquier cosa" y "_" a "una letra":

```

/* Selecciono a todos los socios con nombres compuestos */
SELECT * FROM SOCIOS WHERE NOMBRE LIKE "% %";

SELECT * FROM LIBROS WHERE TITLIBRO LIKE "El Quijote%";

/* Libros cuyo título comienza por tres letrasy una Q*/
SELECT * FROM LIBROS WHERE TITLIBRO LIKE "___Q%";

```

REGEXP es más potente. Permite definir búsquedas usando **expresiones regulares**. Las expresiones regulares son un método estándar de definir patrones de caracteres. Muy a menudo no basta con "%" y "_". El conjunto de caracteres especiales que pueden usarse:

^	Comienzo de la cadena de texto
\$	Fin de la cadena de texto
*	Repetir el carácter o expresión de la izquierda de 0 a n veces
+	Repetir el carácter o expresión de la izquierda de 1 a n veces
?	Repetir el carácter o expresión de la izquierda 0 ó 1 vez
cad1 cad2	Concuerta con una de las dos cadenas
(expresion)	Agrupar expresiones para las repeticiones
{n}, {n,m}	Repetir el carácter o expresión de la izquierda n veces, o de n a m veces
[a-z], [^a-z]	Concuerta (o no, usando "^") con cualquier carácter del rango.
.(punto)	Cualquier carácter

Unos cuantos ejemplos:

```

/* Títulos que no tengan la letra "a" */
SELECT * FROM LIBROS WHERE TITLIBRO NOT REGEXP "a";

/* Títulos con alguna cifra */
SELECT * FROM LIBROS WHERE TITLIBRO REGEXP "[0-9]";

/* Libros con un palito en su título. BYNARY (cadenas binarias) hace
que distinga entre mayúsculas y minúsculas)
SELECT * FROM LIBROS WHERE TITLIBRO REGEXP BINARY "I";

```

La sintaxis de WHERE y HAVING es la misma. Pero se comportan de forma distinta. HAVING está diseñado para las agrupaciones. Antes de que se agrupen los datos con GROUP BY, primero se filtran con WHERE. Sólo se agruparán los datos que cumplan las condiciones impuestas por esa cláusula. Después de agruparse es cuando actúa HAVING. En el apartado siguiente veremos esto con más claridad.

Si es posible, el motor de la base de datos usará las claves que coincidan con los criterios de filtrado para acelerar las operaciones. El orden de las preguntas debe coincidir con el orden de las claves si quieres que esto suceda. Estudia las claves existentes antes de escribir las cláusulas WHERE.

7.9.4 Agrupando datos

A menudo no interesa el detalle de una búsqueda, sino los totales de la misma. Cuántos socios hay, cuánto suma el total de pagos, etc. Eso ya sabemos hacerlo; basta con usar las funciones COUNT, SUM, etc.:

```
SELECT COUNT(*) AS numero FROM SOCIOS;
SELECT SUM(NUMPAGLIBRO) FROM LIBROS;
```

Pero no suele ser bastante. No queremos el total de todos los pagos, sino que queremos los totales agrupados por clientes. O el número de libros prestados agrupados por socios:

```
mysql> SELECT COUNT(*) AS prestados FROM PRESTAMOS
-> GROUP BY CODSOCIO;
+-----+
| prestados |
+-----+
|          2 |
|          3 |
+-----+
2 rows in set (0.00 sec)
```

Sólo me interesan los libros que no han sido devueltos. Antes de agrupar, filtro los datos y me quedo con aquellos que todavía no han sido devueltos. La forma de hacerlo es establecer la condición con la cláusula WHERE:

```
mysql> SELECT CODSOCIO, COUNT(*) as NUM_PRESTADOS FROM PRESTAMOS
-> WHERE FECDEVPRESTAMO IS NULL
-> GROUP BY CODSOCIO;
+-----+-----+
| CODSOCIO | NUM_PRESTADOS |
+-----+-----+
|          1 |                2 |
|          2 |                1 |
+-----+-----+
2 rows in set (0.00 sec)
```

Supongamos ahora que me interesan únicamente aquellos socios con más de un libro prestado:

```
mysql> SELECT CODSOCIO, COUNT(*) as NUM_PRESTADOS FROM PRESTAMOS
-> WHERE FECDEVPRESTAMO IS NULL
-> GROUP BY CODSOCIO
-> HAVING NUM_PRESTADOS>1;
+-----+-----+
| CODSOCIO | NUM_PRESTADOS |
+-----+-----+
|          1 |                2 |
+-----+-----+
1 row in set (0.00 sec)
```

Como la pregunta debe ir **después** de hacer la agrupación (es después de agrupar cuando sé cuántos libros tiene cada socio), tengo que usar la cláusula HAVING. Lógicamente, sólo puedo referirme a campos que existan después de la agrupación: Las columnas que aparecen en el SELECT y en GROUP BY. Si intento usar columnas “no agrupadas” dará error.

7.9.5 Ordenar datos

Es al final de todas las operaciones cuando indicamos el orden en el que queremos presentar los datos. Se hace mediante ORDER BY, que siempre será la última cláusula de la instrucción SELECT. Su uso es muy sencillo:

```
SELECT * FROM SOCIOS ORDER BY AP1SOCIO, AP2SOCIO, NOMSOCIO;
```

Podemos indicar orden ascendente o descendente para cada campo:

```
SELECT * FROM SOCIOS ORDER BY AP1SOCIO DESC, AP2SOCIO, NOMSOCIO;
```

No es necesario que las filas por la que se ordena sean visibles:

```
SELECT NOMSOCIO, TELSOCIO FROM SOCIOS ORDER BY CODSOCIO;
```

Por supuesto, la velocidad de la ordenación dependerá de si existen claves que coincidan con lo que se ha pedido. Es una buena idea estudiar cómo entregaremos los datos habitualmente a la hora de escribir las claves secundarias.

7.9.6 Limitar las filas mostradas

LIMIT permite limitar el número de filas que se mostrarán. Depende de ORDER BY el seleccionar cuáles serán. Si se indica un número mostrará ese número de filas. Si se indican dos números, el primero indica a partir de qué número de registro (comienza en cero) se mostrarán cuántas filas:

```
mysql> SELECT * FROM LIBROS ORDER BY CODLIBRO LIMIT 2,3;
```

CODLIBRO	TITLIBRO	NUMPAGLIBRO	CODEEDITORIAL
3	Neuromante	456	NULL
4	El Tambor de Hojalata	837	NULL
5	Pipo y la granja	36	NULL

3 rows in set (0.00 sec)

```
mysql> SELECT * FROM LIBROS ORDER BY CODLIBRO LIMIT 0,3;
```

CODLIBRO	TITLIBRO	NUMPAGLIBRO	CODEEDITORIAL
1	El Quijote I	640	NULL
2	El Quijote II	540	NULL
3	Neuromante	456	NULL

3 rows in set (0.00 sec)

LIMIT también puede aplicarse a la sentencia UPDATE.

Relacionado con el tema, a veces nos interesa numerar las filas que devuelve el SELECT. Generalmente se hace para poder seleccionar cierto número de registros de la consulta, esto es, lo que hace LIMIT directamente. Sin embargo, si se desea numerar de todos modos:

```
mysql> SELECT @FILA:=@FILA+10 AS NUM, LIBROS.* FROM (SELECT @FILA:=0) DA_IGUAL, LIBROS;
```

NUM	CODLIBRO	TITLIBRO	NUMPAGLIBRO	CODEEDITORIAL
10	1	El Quijote I	640	NULL
20	2	El Quijote II	540	NULL
30	3	Neuromante	456	NULL
40	4	El Tambor de Hojalata	837	NULL
50	5	Pipo y la granja	36	NULL

5 rows in set (0.00 sec)

La sintaxis de MySQL es más potente de lo que parece... Consulta en el manual de referencia sobre "variables de usuario" si quieres saber más.

7.9.7 Consultas a Varias Tablas. Join.

En el capítulo 2 ya vimos que lo normal es mostrar información de varias tablas a la vez. En MySQL lo haremos usando la sentencia SELECT de SQL:


```
mysql> SELECT SOCIOS.CODSOCIO, SOCIOS.NOMSOCIO, SOCIOS.APISOCIO, PRESTAMOS.* FROM SOCIOS, PRESTAMOS;
```

CODSOCIO	NOMSOCIO	APISOCIO	CODSOCIO	CODLIBRO	FECENTPRESTAMO	FECDEVPRESTAMO
3	Juana	Arregui	1	1	2009-12-04 00:00:00	NULL
2	Maria	Arresti	1	1	2009-12-04 00:00:00	NULL
1	Javier	Rodriguez	1	1	2009-12-04 00:00:00	NULL
3	Juana	Arregui	1	2	2009-12-04 00:00:00	NULL
2	Maria	Arresti	1	2	2009-12-04 00:00:00	NULL
1	Javier	Rodriguez	1	2	2009-12-04 00:00:00	NULL
3	Juana	Arregui	2	1	2009-12-01 00:00:00	2009-12-03 00:00:00
2	Maria	Arresti	2	1	2009-12-01 00:00:00	2009-12-03 00:00:00
1	Javier	Rodriguez	2	1	2009-12-01 00:00:00	2009-12-03 00:00:00
3	Juana	Arregui	2	4	2009-12-04 00:00:00	2009-12-05 00:00:00
2	Maria	Arresti	2	4	2009-12-04 00:00:00	2009-12-05 00:00:00
1	Javier	Rodriguez	2	4	2009-12-04 00:00:00	2009-12-05 00:00:00
3	Juana	Arregui	2	5	2009-12-04 00:00:00	NULL
2	Maria	Arresti	2	5	2009-12-04 00:00:00	NULL
1	Javier	Rodriguez	2	5	2009-12-04 00:00:00	NULL

15 rows in set (0.00 sec)

Salen cosas absurdas: El producto cartesiano (todas las combinaciones posibles) de los registros de las dos tablas. Como vimos en su momento, el problema es que no hemos dicho la relación existente entre las tablas. Obviamente no queremos todas esas combinaciones, sino aquellas en las que los códigos de socio de ambas tablas coincidan.

La forma tradicional de hacerlo era con una sentencia parecida a esta:

```
mysql> SELECT SOCIOS.CODSOCIO, SOCIOS.NOMSOCIO, SOCIOS.APISOCIO, PRESTAMOS.* FROM SOCIOS, PRESTAMOS
-> WHERE SOCIOS.CODSOCIO=PRESTAMOS.CODSOCIO;
```

CODSOCIO	NOMSOCIO	APISOCIO	CODSOCIO	CODLIBRO	FECENTPRESTAMO	FECDEVPRESTAMO
2	Maria	Arresti	2	1	2009-12-01 00:00:00	2009-12-03 00:00:00
2	Maria	Arresti	2	4	2009-12-04 00:00:00	2009-12-05 00:00:00
2	Maria	Arresti	2	5	2009-12-04 00:00:00	NULL
1	Javier	Rodriguez	1	1	2009-12-04 00:00:00	NULL
1	Javier	Rodriguez	1	2	2009-12-04 00:00:00	NULL

5 rows in set (0.00 sec)

Usamos WHERE para filtrar la información que no deseamos. Pero esta operación es lenta, sobre todo si añadimos en el WHERE otras condiciones que no tengan que ver con la relación entre las tablas.

Por ese motivo se añadió otra palabra al lenguaje, **JOIN** o **INNER JOIN**³². Está diseñada para unir tablas. “Matemáticamente hablando” hace lo mismo que la instrucción que acabamos de ver, pero la base de datos sabe lo que queremos hacer y puede optimizar:

```
mysql> SELECT SOCIOS.CODSOCIO, SOCIOS.NOMSOCIO, SOCIOS.APISOCIO, PRESTAMOS.* FROM SOCIOS
-> INNER JOIN PRESTAMOS ON SOCIOS.CODSOCIO=PRESTAMOS.CODSOCIO;
```

CODSOCIO	NOMSOCIO	APISOCIO	CODSOCIO	CODLIBRO	FECENTPRESTAMO	FECDEVPRESTAMO
2	Maria	Arresti	2	1	2009-12-01 00:00:00	2009-12-03 00:00:00
2	Maria	Arresti	2	4	2009-12-04 00:00:00	2009-12-05 00:00:00
2	Maria	Arresti	2	5	2009-12-04 00:00:00	NULL
1	Javier	Rodriguez	1	1	2009-12-04 00:00:00	NULL
1	Javier	Rodriguez	1	2	2009-12-04 00:00:00	NULL

5 rows in set (0.00 sec)

Su sintaxis (muy simplificada, como siempre):

```
tabla_uno [INNER | LEFT | RIGHT] JOIN tabla_dos ON condiciones
```

JOIN permite tres modificadores:

³² Irónicamente, las últimas revisiones del lenguaje SQL permiten escribir la instrucción de la forma antigua: Internamente la base de datos lo convertirá en un JOIN. Pero a menos que estés seguro de que tu programa siempre se va a ejecutar en esas modernas bases de datos, apréndete la sintaxis de JOIN.

INNER. Si no se indica, es el que se aplica. Muestra los registros de las tablas que cumplan las condiciones; El comportamiento normal que esperamos.

LEFT. Similar al anterior, pero aparte de mostrar los registros comunes, en la relación aparecen todos los registros de la tabla de la izquierda que no han conseguido emparejarse con la tabla de la derecha. En esas filas, las columnas correspondientes a la tabla de la derecha aparecerán con valores NULL (no hay nada que mostrar). Es muy útil. Supongamos que quiero saber qué socios no han pedido nunca un libro prestado:

```
mysql> SELECT SOCIOS.CODSOCIO, SOCIOS.NOMSOCIO, PRESTAMOS.CODSOCIO FROM SOCIOS
-> LEFT JOIN PRESTAMOS ON SOCIOS.CODSOCIO=PRESTAMOS.CODSOCIO;
```

CODSOCIO	NOMSOCIO	CODSOCIO
3	Juana	NULL
2	Maria	2
2	Maria	2
2	Maria	2
1	Javier	1
1	Javier	1

6 rows in set (0.00 sec)

```
mysql> SELECT SOCIOS.CODSOCIO, SOCIOS.NOMSOCIO FROM SOCIOS
-> LEFT JOIN PRESTAMOS ON SOCIOS.CODSOCIO=PRESTAMOS.CODSOCIO
-> WHERE PRESTAMOS.CODSOCIO IS NULL;
```

CODSOCIO	NOMSOCIO
3	Juana

1 row in set (0.00 sec)

En la primera instrucción he pedido los datos usando LEFT JOIN. Al parecer Juana nunca ha pedido ningún libro. No tiene registros de PRESTAMOS asociados; Pero como he usado LEFT, sí que aparece en la relación. Lógicamente, el campo de la tabla PRESTAMOS contiene un NULL. En la segunda consulta uso este hecho para sacar sólo lo que necesito.

RIGHT. Como la anterior, pero con la tabla de la derecha.

¿Y si quiero datos de más de dos tablas? La mecánica es la misma, sólo que con más cláusulas JOIN y unos cuantos paréntesis. Supongamos que queremos ver datos de SOCIOS, PRESTAMOS Y LIBROS a la vez. Primero empezamos a construir el JOIN entre dos de ellas (que tengan relación directa, por supuesto), por ejemplo PRESTAMOS Y LIBROS:

```
PRESTAMOS JOIN LIBROS ON PRESTAMOS.CODLIBRO=LIBROS.CODLIBRO
```

Ahora lo encerramos entre paréntesis y hacemos un JOIN con la tabla que falta:

```
SOCIOS JOIN
(PRESTAMOS JOIN LIBROS ON PRESTAMOS.CODLIBRO=LIBROS.CODLIBRO)
ON SOCIOS.CODSOCIO=PRESTAMOS.CODSOCIO
```

Y por último completamos la sentencia SELECT con los campos que queremos mostrar, el FROM y las condiciones que necesitamos:

```
mysql> SELECT SOCIOS.NOMSOCIO, SOCIOS.AP1SOCIO, PRESTAMOS.FECENTPRESTAMO, LIBROS.TITLIBRO FROM
-> SOCIOS JOIN (PRESTAMOS JOIN LIBROS ON PRESTAMOS.CODLIBRO=LIBROS.CODLIBRO)
-> ON SOCIOS.CODSOCIO=PRESTAMOS.CODSOCIO
-> WHERE PRESTAMOS.FECDEVPRESTAMO IS NULL;
```

NOMSOCIO	AP1SOCIO	FECENTPRESTAMO	TITLIBRO
Javier	Rodriguez	2009-12-04 00:00:00	El Quijote I
Javier	Rodriguez	2009-12-04 00:00:00	El Quijote II
Maria	Arresti	2009-12-04 00:00:00	Pipo y la granja

3 rows in set (0.00 sec)

7.9.8 Union

SQL permite unir los resultados de varias sentencias SELECT, siempre que tengan el mismo número de columnas. Es muy útil para unir datos de diferentes fuentes:

```
mysql> SELECT NOMSOCIO FROM SOCIOS UNION SELECT TITLIBRO FROM LIBROS;
```

NOMSOCIO
Juana
Maria
Javier
El Quijote I
El Quijote II
Neuromante
El Tambor de Hojalata
Pipo y la granja

```
8 rows in set (0.00 sec)
```

Si la sentencia SELECT es compleja, puede encerrarse entre paréntesis para evitar ambigüedades. El nombre de las columnas será el que tengan en la primera de las sentencias SELECT.

7.9.9 Subconsultas

Dentro de una sentencia SELECT pueden escribirse otras sentencias SELECT. Es habitual usarlo para filtrar datos:

```
mysql> SELECT CODLIBRO,TITLIBRO FROM LIBROS
-> WHERE CODLIBRO IN (SELECT CODLIBRO FROM PRESTAMOS);
```

CODLIBRO	TITLIBRO
1	El Quijote I
2	El Quijote II
4	El Tambor de Hojalata
5	Pipo y la granja

```
4 rows in set (0.00 sec)
```

En este caso resultaría más rápido hacer un JOIN. Usa subconsultas sólo si las tablas no están directamente relacionadas, o si la subconsulta es una SELECT compleja:

```
mysql> SELECT CONCAT( NOMSOCIO, " ",APISOCIO) AS NOM_AP FROM SOCIOS WHERE CODSOCIO IN
-> (SELECT CODSOCIO FROM PRESTAMOS GROUP BY CODSOCIO HAVING COUNT(*)>1);
```

NOM_AP
Maria Arresti
Javier Rodriguez

```
2 rows in set (0.00 sec)
```

He impreso los nombres y apellidos de los socios con más de un libro prestado. Primero, en la subconsulta, he buscado los datos que me interesaban y después en la consulta he operado sólo con ellos.

7.10 Vistas

Las vistas son objetos de la base de datos creados mediante un SELECT. Son el equivalente de las consultas de Access. Como es lógico, la base de datos sólo guarda la definición de la vista, no los datos seleccionados. Cada vez que se pide, se calcula.

Se crean por dos motivos:

- Es más cómodo usar una vista (se comporta como una tabla) que escribir una y otra vez una larga sentencia SELECT.
- Por seguridad. A menudo los usuarios que manejan la base de datos no tienen permisos para acceder a toda la tabla; Sólo pueden trabajar con la vista que ha creado el diseñador. Consulta el manual de referencia si quieres saber más sobre GRANT y los permisos de MySQL.

La sintaxis para crear, modificar o borrar vistas es muy sencilla:

```
CREATE VIEW nombre_vista [(alias_columnas,...)] AS sentencia_select
```

```
ALTER VIEW nombre_vista [(alias_columnas,...)] AS sentencia_select
```

```
DROP VIEW nombre_vista
```

Sentencia_select puede ser cualquier instrucción SELECT válida. Si se quiere renombrar las columnas, hay que indicar tantas como tenga el SELECT original.

```
mysql> CREATE VIEW LIBROS_PRESTADOS AS
-> SELECT DISTINCT LIBROS.CODLIBRO,TITLIBRO FROM LIBROS
-> JOIN PRESTAMOS ON LIBROS.CODLIBRO=PRESTAMOS.CODLIBRO;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM LIBROS_PRESTADOS;
+-----+-----+
| CODLIBRO | TITLIBRO |
+-----+-----+
| 1 | El Quijote I |
| 2 | El Quijote II |
| 4 | El Tambor de Hojalata |
| 5 | Pipo y la granja |
+-----+-----+
4 rows in set (0.00 sec)
```

Si se quiere ver la sentencia SELECT que originó la vista puedes usar el comando **show create table nombre_vista**. Para ver la lista de vistas y tablas existentes, usa el comando **show tables**.